

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

The background of the book cover features a futuristic, glowing digital cityscape with floating data structures and light trails, set against a dark space-like environment.

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

About This E-Book

EPUB is an open, industry-standard format for e-books. However, support for EPUB and its many features varies across reading devices and applications. Use your device or app settings to customize the presentation to your liking. Settings that you can customize often include font, font size, single or double column, landscape or portrait mode, and figures that you can click or tap to enlarge. For additional information about the settings and features on your reading device or app, visit the device manufacturer's Web site.

Many titles include programming code or configuration examples. To optimize the presentation of these elements, view the e-book in single-column, landscape mode and adjust the font size to the smallest setting. In addition to presenting code and configurations in the reflowable text format, we have included images of the code that mimic the presentation found in the print book; therefore, where the reflowable format may compromise the presentation of the code listing, you will see a "Click here to view code image" link. Click the link to view the print-fidelity code image. To return to the previous page viewed, click the Back button on your device or app.

Oracle Database Problem Solving and Troubleshooting Handbook

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



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](#)

[Acknowledgments](#)

[About the Authors](#)

[About the Technical Reviewers and Contributors](#)

[Chapter 1 Troubleshooting and Tuning LOB Segment Performance](#)

[Introduction to the LOB Datatype](#)

[Fixing a LOB Problem: A Real-World Example](#)

[Another Real-World Example: HW Resolution](#)

[BASICFILE LOB Issues: Toward a More Perfect Fix](#)

[BASICFILE versus SECUREFILE LOBs](#)

[LOB New and Old Type Differences](#)

[Migrating BASICFILE LOBs to SECUREFILE LOBs](#)

[The Impact of PCTFREE on LOBs](#)

[Overcoming Poor INSERT Performance](#)

[Summary](#)

[Chapter 2 Overcoming Undo Tablespace Corruption](#)

[Overview of Undo Management](#)

[The Importance of UNDO_RETENTION](#)

[Tuning UNDO_RETENTION](#)

[DTP, XA, and Rollback Segments](#)

[Other Unusual Rollback and Undo Segment Issues](#)

[Recovering from Undo Tablespace Corruption](#)

[Preventing, Detecting, and Repairing Corruption](#)

[Handling Memory Corruption](#)

[Handling Logical Corruption](#)

[Overcoming Media Corruption](#)

[Summary](#)

[Chapter 3 Handling GC Buffer Busy Wait Events](#)

[Overview of Buffer Busy Wait Events](#)

[Leveraging the ORAchk Utility](#)

[Installing ORAchk](#)

[Results of ORAchk Execution: Sample Output](#)

[Isolating GC Buffer Busy Waits](#)

[Using ADDM to Find Event Information](#)

[Using AWR to Find Event Information](#)

[Using ASH to Find Event Information](#)

[Isolating GC Buffer Busy Wait Event Issues](#)

[Using ASH Views to Find Waiting Sessions](#)

[Quickly Isolating Performance Bottlenecks](#)

[Fixes for GC Buffer Busy Waits](#)

[Summary](#)

[Chapter 4 Adaptive Cursor Sharing](#)

[ACS Working Algorithm](#)

[Bind Sensitiveness with Range Predicate](#)

[Bind Sensitiveness with Equality Predicate and Histogram](#)

[Bind Sensitiveness with Partition Keys](#)

[ACS in Action](#)

[ACS Bind-Awareness Monitoring](#)

[BUCKET_ID and COUNT Relationship](#)

[Marking Cursors Bind Aware](#)

[The Bind-Aware Cursor](#)

[A Practical Case](#)

[Summary](#)

[Chapter 5 Stabilizing Query Response Time Using SQL Plan Management](#)

[Getting Started](#)

[Creating a SQL Plan Baseline](#)

[Capturing Plans Automatically](#)

[Loading Plans from the Cursor Cache](#)

[Faking Baselines](#)

[Oracle Optimizer and SPM Interaction](#)

[When the CBO Plan Matches the SQL Plan Baseline](#)

[When the CBO Plan Doesn't Match the SQL Plan Baseline](#)

[When SQL Plan Baseline Is Not Reproducible](#)

[SQL Plan Baseline Reproducibility](#)

[Renaming the Index](#)
[Changing the Index Type](#)
[Adding Trailing Columns to the Index](#)
[Reversing the Index](#)
[NLS_SORT and SQL Plan Baseline Reproducibility](#)
[ALL_ROWS versus FIRST_ROWS](#)
[Adaptive Cursor Sharing and SPM](#)
[ACS and SPM in Oracle 11g Release 11.2.0.3.0](#)
[ACS and SPM in Oracle Database 12c Release 12.1.0.1.0](#)
[Summary](#)

[Chapter 6 DDL Optimization Tips, Techniques, and Tricks](#)

[DDL Optimization Concept](#)
[The DDL Optimization Mechanism](#)
[Table Cardinality Estimation](#)
[C_DDL Column in a Virtual Column](#)
[C_DDL Column in a Column Group Extension](#)
[When the Default Value of C_DDL Changes](#)
[C_DDL Column and Indexes](#)
[DDL Optimization for NULL Columns](#)
[Summary](#)

[Chapter 7 Managing, Optimizing, and Tuning VLDBs](#)

[Overview of Very Large Databases](#)
[Optimal Basic Configuration](#)
[Data Warehouse Template](#)
[Optimal Data Block Size](#)
[Bigfile Tablespaces](#)
[Adequate SGA and PGA](#)
[Temporary Tablespace Groups](#)
[Data Partitioning](#)
[Index Partitioning: Local versus Global](#)
[Data Compression](#)
[Table Compression](#)
[Heat Map and Automatic Data Optimization](#)
[Advanced Index Partition Compression](#)

VLDB Performance Tuning Principles

Real-World Scenario

Limiting the Impact of Indexes on Data Loading

Maximizing Resource Utilization

Gathering Optimizer Statistics

Incremental Statistics Synopsis

Gathering Statistics Concurrently

Setting the ESTIMATE_PERCENT Value

Backup and Recovery Best Practices

Exadata Solutions

Utilizing a Data Guard Environment

Summary

Chapter 8 Best Practices for Backup and Recovery with Recovery Manager

A Perfect Backup and Recovery Plan

An Overview of RMAN

Tips for Database Backup Strategies

Full Backups and Incremental Backups

Compressed Backups

Incremental Backups

Faster Incremental Backups

Rewinding in Oracle Flashback Technology

Disk-Based Backup Solutions

Recover Forward Forever

Validating RMAN Backups

Backup Optimization and Tuning

Tuning Disk-Based Backup Performance

Using RMAN for RAC Databases

Retaining Data in a Recovery Catalog

Having a Robust Recovery Strategy

Leveraging the Data Recovery Advisor

Summary

Chapter 9 Database Forensics and Tuning Using AWR Analysis: Part I

What Is AWR?

Knowing What to Look For

[Header Section](#)

[Load Profile](#)

[Instance Efficiencies](#)

[Shared Pool Memory](#)

[Wait Events](#)

[Load Average](#)

[Instance CPU](#)

[Memory Statistics](#)

[RAC-Specific Pages](#)

[RAC Statistics \(CPU\)](#)

[Global Cache Load Statistics](#)

[Global Cache and Enqueue Services](#)

[Cluster Interconnects](#)

[Time Model Statistics](#)

[Operating System Statistics](#)

[Foreground Wait Events](#)

[Background Wait Events](#)

[Wait Event Histograms](#)

[Service-Related Statistics](#)

[The SQL Sections](#)

[Total Elapsed Time](#)

[Total CPU Time](#)

[Total Buffer Gets](#)

[Total Disk Reads](#)

[Total Executions](#)

[Parse Calls](#)

[Shareable Memory](#)

[Version Count](#)

[Cluster Wait Time](#)

[Instance Activity Statistics](#)

[Consistent Get Statistics](#)

[DB Block Get Statistics](#)

[Dirty Block Statistics](#)

[Enqueue Statistics](#)

[Execution Count](#)

[Free Buffer Statistics](#)
[Global Cache \(GC\) Statistics](#)
[Index Scan Statistics](#)
[Leaf Node Statistics](#)
[Open Cursors](#)
[Parse Statistics](#)
[Physical Read and Write Statistics](#)
[Recursive Statistics](#)
[Redo-Related Statistics](#)
[Session Cursor Statistic](#)
[Sort Statistics](#)
[Summed Dirty Queue Length](#)
[Table Fetch Statistics](#)
[Transaction Rollback](#)
[Undo Change Vector Statistic](#)
[User Statistics](#)
[Work Area Statistics](#)
[Instance Activity Statistics—Absolute Values](#)
[Instance Activity Statistics—Thread Activity](#)

[Summary](#)

[Chapter 10 Database Forensics and Tuning Using AWR Analysis: Part II](#)

[Tablespace I/O Statistics](#)
[Buffer Pool Statistics](#)
 [Buffer Pool Statistics](#)
 [Instance Recovery Statistics](#)
 [Buffer Pool Advisory Section](#)

[PGA Statistics](#)

[PGA Aggregate Summary](#)
 [PGA Aggregate Target Statistics](#)
 [PGA Aggregate Target Histogram](#)
 [PGA Memory Advisor](#)

[Shared Pool Statistics](#)

[Other Advisories](#)

[SGA Target Advisory](#)
 [Streams Pool Advisory](#)

[Java Pool Advisory](#)

[Buffer Waits Statistics](#)

[Enqueue Statistics](#)

[Undo Segment Statistics](#)

[Latch Statistics](#)

[Latch Activity](#)

[Latch Sleep Breakdown](#)

[Latches and Spin Count](#)

[Latch Miss Sources](#)

[Mutex Sleep Summary](#)

[Parent and Child Latches](#)

[Segment Access Areas](#)

[Library Cache Activity Sections](#)

[Dynamic Memory Components Sections](#)

[Process Memory Sections](#)

[Process Memory Summary](#)

[SGA Memory Summary](#)

[SGA Breakdown Difference](#)

[Streams Component Sections](#)

[Resource Limits Statistics](#)

[Initialization Parameter Changes](#)

[Global Enqueue and Other RAC Sections](#)

[Global Enqueue Statistics](#)

[Global CR Served Statistics](#)

[Global Current Served Statistics](#)

[Global Cache Transfer Statistics](#)

[Global Cache Transfer Times](#)

[Global Cache Transfer \(Immediate\)](#)

[Global Cache Times \(Immediate\)](#)

[Interconnect Ping Latency Statistics](#)

[Interconnect Throughput by Client](#)

[Interconnect Device Statistics](#)

[Summary](#)

[Chapter 11 Troubleshooting Problematic Scenarios in RAC](#)

[Troubleshooting and Tuning RAC](#)

[Start with ORAchk](#)

[Employ the TFA Collector Utility](#)

[Utilize the Automatic Diagnostic Repository](#)

[Check the Alert and Trace Log Files](#)

[Employ the Three A's](#)

[Check the Private Cluster Interconnect](#)

[Enable Tracing and Inspect the Trace Logs](#)

[Utilize the Cluster Health Monitor](#)

[Miscellaneous Tools and Utilities](#)

[Useful My Oracle Support Resources](#)

[A Well-Oiled RAC Ecosystem](#)

[Maximum Availability Architecture](#)

[Optimal and Efficient Databases in RAC](#)

[Troubleshooting RAC with OEM 12c](#)

[Utilities and Commands for Troubleshooting](#)

[Summary](#)

[Chapter 12 Leveraging SQL Advisors to Analyze and Fix SQL Problems](#)

[OEM 12c—SQL Advisors Home](#)

[SQL Tuning Advisor](#)

[Running SQL Tuning Advisor in OEM 12c](#)

[Running SQL Tuning Advisor Manually in SQL*Plus](#)

[SQL Access Advisor](#)

[Running SQL Access Advisor in OEM 12c](#)

[Running SQL Access Advisor Manually in SQL*Plus](#)

[SQL Repair Advisor](#)

[SQL Performance Analyzer](#)

[Summary](#)

[Chapter 13 Extending Data Pump for Data and Object Migration](#)

[Using Data Pump](#)

[Copying Objects](#)

[Data Pump Modes](#)

[Working with Private and Public Objects](#)

[Saving and Restoring Database Links](#)

[Exporting Public Database Links and Synonyms](#)

[Verifying Content of the Export Dump File](#)
[Finding Valid INCLUDE and EXCLUDE Values](#)
[Exporting Subsets of Data](#)
[Changing Object Properties](#)
[Importing Partitioned Tables as Nonpartitioned](#)
[Importing Table Partitions as Individual Tables](#)
[Masking Data](#)
[Renaming Tables or Different Tablespaces](#)
[Using Default Storage Parameters](#)
[Resizing Tablespaces during Import](#)
[Consolidating Multiple Tablespaces](#)
[Using PL/SQL API with Data Pump](#)
[Monitoring and Altering Resources](#)
[Improving Performance](#)
[Upgrading Databases](#)
[Summary](#)

Chapter 14 Strategies for Migrating Data Quickly between Databases

[Why Bother Migrating?](#)
[Determining the Best Strategy](#)
[Real-Time versus Near Real-Time Migration](#)
[Read-Only Tolerance](#)
[Reversibility](#)

[Considering What Data to Migrate](#)

[Data Migration Methods](#)
[Transactional Capture Migration Methods](#)
[Nontransactional Migration Methods](#)
[Piecemeal Migration Methods](#)

[Summary](#)

Chapter 15 Diagnosing and Recovering from TEMPFILE I/O Issues

[Overview of Temporary Tablespaces](#)
[Read-Only Databases](#)
[Locally Managed Temporary Tablespaces](#)
[Temporary Tablespace Groups](#)
[Global Temporary Tables](#)

Correcting TEMPFILE I/O Waits

Undersized PGA

Inappropriate TEMPFILE Extent Sizing

Inappropriate Use of GTTs

Summary

Chapter 16 Dealing with Latch and Mutex Contention

Overview of Latch and Mutex Architecture

What Are Latches?

What Are Mutexes?

Latch and Mutex Internals

Measuring Latch and Mutex Contention

Identifying Individual Latches

Drilling into Segments and SQLs

Latch and Mutex Scenarios

Library Cache Mutex Waits

Library Cache Pin

Shared Pool Latch

Cache Buffers Chains Latch

Other Latch Scenarios

Intractable Latch Contention

Fine Tuning Latch Algorithms

Summary

Chapter 17 Using SSDs to Solve I/O Bottlenecks

Disk Technologies: SSD versus HDD

The Rise of Solid-State Flash Disks

Flash SSD Latency

Economics of SSD

SLC, MLC, and TLC Disks

Write Performance and Endurance

Garbage Collection and Wear Leveling

SATA versus PCIe SSD

Using SSD Devices in an Oracle Database

The Oracle Database Flash Cache

Free Buffer Waits

Configuring and Monitoring DBFC

[Using the FLASH_CACHE Clause](#) [Flash Cache Performance Statistics](#)

[Comparing SSD Options](#)

[Indexed Reads](#)

[OLTP Read/Write Workload](#)

[Full Table Scan Performance](#)

[SSD Native Caches and Full Table Scans](#)

[Disk Sort and Hash Operations](#)

[Redo Log Optimization](#)

[Storage Tiering](#)

[Using Partitions to Tier Data](#)

[Flash and Exadata](#)

[Creating Flash-Backed ASM Disk Groups on Exadata](#)

[Summary](#)

[Chapter 18 Designing and Monitoring Indexes for Optimal Performance](#)

[Types of Indexes](#)

[B-Tree Indexes](#)

[Bitmap Indexes](#)

[Partitioned Indexes](#)

[Other Index Types](#)

[Multiple Indexes on Identical Columns](#)

[Index Performance Issues](#)

[Index Statistics](#)

[The Impact of a Low Clustering Factor](#)

[Operational Considerations for Indexes](#)

[Hiding Unselective Indexes](#)

[Index Performance Issues in RAC Databases](#)

[Summary](#)

[Chapter 19 Using SQLT to Boost Query Performance](#)

[Installing SQLT](#)

[Using the XTRACT Method](#)

[Using the XECUTE Method](#)

[Leveraging Other SQLT Methods](#)

[A Real-World Example](#)

[Summary](#)

[**Chapter 20 Dealing with XA Distributed Transaction Issues**](#)

[Repairing Common Distributed Transaction Issues](#)

[Repairing Ghost Distributed Transactions](#)

[Information Exists, but Transaction Missing](#)

[ORA-1591 Has No Corresponding Information](#)

[Transaction Hangs after COMMIT or ROLLBACK](#)

[Monitoring Distributed Transactions](#)

[Summary](#)

[Index](#)

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 11g and Oracle Database 12c 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.

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 Brain-Surface 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 11g; an Oracle Certified Professional (9*i*, 10g, 11g, and 12c); an Oracle RAC 10g and 11g Certified Specialist; an Oracle DBA 10g 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 8i, 9i, 10g, and 11g, 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 12c, 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 Applications* (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).

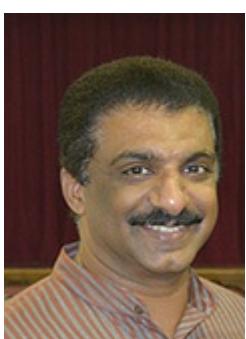


Anju Garg is an Oracle ACE Associate with more than thirteen years of experience in the IT industry in various roles. Anju is a certified expert in Oracle RAC, Oracle Database Performance Tuning, and SQL Statement Tuning. Since 2010, she has been involved in teaching and has trained more than a hundred 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 (9i, 10g, 11g), 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 8i, all versions including Oracle Database 12c OCA. Biju blogs at www.bijoos.com/oraclenotes, and you can follow him on Twitter

(@biju_thomas) and Facebook (oraclenotes) for daily Oracle Tidbits.

1. Troubleshooting and Tuning LOB Segment Performance

The large object (LOB) datatype allows us to hold and manipulate unstructured and semistructured data such as documents, graphic images, video clips, sound files, and XML files. The `DBMS_LOB` package was designed to manipulate LOB datatypes. Starting in Oracle Database 12c, LOBs can store large amounts of data with a maximum size of 128 TB depending on the database block size; a single table can have one or more columns of LOB datatypes, such as binary large object (BLOB), character large object (CLOB), national character large object (NCLOB), and BFILE. This chapter describes some of the problems that can happen when you have LOB segments in your environment and how to mitigate these problems.

Introduction to the LOB Datatype

When creating and designing your database’s LOB objects, remember to carefully review the *Oracle Database SecureFiles and Large Objects Developer’s Guide* (Oracle, 2016) especially [Chapter 14](#), “Performance Guidelines.” This guide updates you on latest recommendations for creating your LOBs depending on what types of data will reside within these table columns.

It’s important to remember that whenever a LOB column is created in a table, two different segments are actually created: one `LOBSEGMENT` and one `LOBINDEX`. The `LOBINDEX` is the one that points to the LOB “chunks” that are stored in the corresponding `LOBSEGMENT`. In some cases, LOBs may be stored “inline”—that is, inside the table segment—but inline storage is usually used for LOB data that is fairly small (less than about 4000 bytes) or is `NULL`. In those cases, LOB values are stored directly inside the table segment.

Now let’s discuss what causes the database to generate a *high-watermark* (HW) enqueue event. When a session wants to access a database resource, the lock that coordinates the access to this resource is an *enqueue*. When an enqueue event happens, it means that a session is waiting for another session to free up this resource. The event always appears with the name of the enqueue in the format `enq: <enqueue_type>-<details>`; each enqueue type will have different details. The `P1`, `P2`, and `P3` columns of dynamic views `V$SESSION` and `V$SESSION_WAIT` are also helpful in giving us more details about where the waiting session is currently held up and what is causing the lock. Those values can assume different meanings depending on the enqueue, as shown in the corresponding `P1TEXT`, `P2TEXT`, and `P3TEXT` columns. [Listing 1.1](#) illustrates just a few of Oracle Database 12c’s more than 600 enqueue events.

Listing 1.1 Enqueue Events

[Click here to view code image](#)

```

select
  distinct name
from
  v$event_name
where
  name like '%enq%'
order by 1;

enq: AB - ABMR process initialized
enq: AB - ABMR process start/stop
enq: AC - acquiring partition id
enq: AD - allocate AU
enq: AD - deallocate AU
enq: AD - relocate AU
enq: AE - lock
...

```

The HW enqueue is responsible for serializing the allocation of space beyond the high-watermark of a segment. The recommended action to handle an unexpected HW enqueue is to manually allocate more extents for the LOB's segments; however, in later sections we also discuss proactive methods to help avoid unexpected HW enqueues by following recommended best practices for LOB configuration.

Fixing a LOB Problem: A Real-World Example

In this real-world example, the database of an e-commerce company is completely hung. The following steps will quickly identify and fix this issue:

1. Create an automatic workload repository (AWR) report to determine if the enq: HW event is one of the top five wait events. Log in to your database as SYSDBA and run the following command. When prompted, choose the two most recent AWR snapshot IDs to analyze just the last two snapshots:

```
$> ?/rdbms/admin/awrrpt.sql
```

2. Review the top five wait events listed in the report:

[Click here to view code image](#)

Event (s)	Time (ms)	Waits	Time
		Wait Class	
<hr/>			
<hr/>			
enq:HW			
contention		249,725	3,289
User I/O direct			13
path			90.0
write		168,486	103
User I/O DB CPU			1
PX qref			2.8

latch	6,392,581	40	0	1.1
Other				
PX Deq: Slave Session				
Stats	18	1	51	.0 Other

- 3.** If the event is constantly occurring in the database, just run the following query to identify the sessions encountering the HW contention:

[Click here to view code image](#)

```
SELECT sid, event
  FROM gv$session_wait
 WHERE event LIKE '%contention%';
```

SID	EVENT
9426	enq: HW - contention
13050	enq: HW - contention

- 4.** Run the following query to isolate the object that is experiencing HW enqueue contention:

[Click here to view code image](#)

```
SELECT
    DBMS_UTLILITY.DATA_BLOCK_ADDRESS_FILE(id2) file#
    DBMS_UTLILITY.DATA_BLOCK_ADDRESS_BLOCK(id2) block#
  FROM gv$lock
 WHERE type = 'HW';
```

FILE#	BLOCK#
19	195

- 5.** Using the datafile number (19) and block ID (195) obtained from the prior query, discover the object being locked by this event via the following query:

[Click here to view code image](#)

```
SELECT
    owner,
    segment_type,
    segment_name
  FROM
    dba_extents
 WHERE
    file_id = 19
  AND
    195 between block_id
  AND
    block_id + blocks - 1;
```

OWNER	SEGMENT_TYPE	SEGMENT_NAME
-------	--------------	--------------

```
----- ----- -----  
ORABPEL      LOBSEGMENT      SYS_LOB0000181226C00029$$
```

6. Next, isolate the table name that references this LOB segment using the following query:

[Click here to view code image](#)

```
SELECT  
    owner,  
    table_name,  
    segment_name  
    FROM  
    dba_lobs  
WHERE  
    segment_name='SYS_LOB0000181226C00029$$';
```

OWNER	TABLE_NAME	SEGMENT_NAME
ACOM_BPEL_AQ	INSERT_SITE_ORDER_BI_TBL	SYS_LOB0000181226C00029\$

7. Now that you have isolated exactly which LOB segment is causing the HW enqueue, it's time to alleviate the root cause of this problem by manually adding a new extent to the segment. It's important to know if the object resides on a tablespace whose extents are either AUTO ALLOCATED or of UNIFORM size. If its extents are defined as UNIFORM, then you simply need to add a new extent of the same size; otherwise, you would add a new extent sized the same as the biggest extent for this segment:

- a. Determine the biggest extent size of this object:

[Click here to view code image](#)

```
SELECT  
    DISTINCT bytes  
    FROM dba_extents  
    WHERE segment_name = 'SYS_LOB0000181226C00029$$'  
    AND owner = 'ORABPEL';
```

BYTES
1048576
8388608
65536

- b. Add some extents to this LOB segment:

[Click here to view code image](#)

```
ALTER TABLE orabpel.insert_site_order_bi_tbl  
MODIFY LOB ('SYS_LOB0000055018C00004$$')  
(ALLOCATE EXTENT (SIZE 8M));
```

Congratulations! With this simple action, you have just saved your company millions of dollars in orders that would be seriously delayed or even lost from customer activity on its e-commerce website.

Another Real-World Example: HW Resolution

If your database is a repository for applications such as Oracle Transportation Management (OTM), Oracle Business Process Execution Language (BPEL), and Oracle E-Business Suite (EBS), it is likely that you will eventually encounter problems regarding LOB segments. For example, we recently ran into an issue with OTM related to the `I_TRANSMISSION` table when we applied 150,000,000 `INSERTs` and `DELETEs` plus 500,000,000 `UPDATEs` against this table's `XML_BLOB` column in a single month. Following are the steps used to verify and repair this problem quickly:

Note

You can also use the following approach with the previous `enq: HW` example. This example simply shows an alternative for detecting and solving the issue.

1. Run an AWR report and check the Top 5 Timed Events section. If you are indeed encountering problems with LOB segments, you will see something like the following report:

[Click here to view code image](#)

Top 5 Timed Events				Avg	Wait	Tot
Event	Waits	Time				
Wait						
db file sequential						
read	2,580,474	35,544	14	77.4	User I/O	
SQL*Net more data from client		659,140		5,513		8
** 12.0 Network						
CPU time				4,540		9.9
enq: HW - contention			88,910	2,890	33	
* 6.3 Configuration						
log file						
sync	777,146	1,688	2			3.7
Commit						

2. In the same AWR report, check the Top Enqueue Activity section for output that's similar to what follows:

[Click here to view code image](#)

Enqueue Activity Snaps: 1234-1235
 -> only enqueues with waits are shown
 -> Enqueue stats gathered prior to 10g should not be compared
 with 10g data
 -> ordered by Wait Time desc, Waits desc

Wait	Gets	Gets	Wait		
Enq Type	Reg	Succ	Failed	Waits	Time
(s)	Time (ms)				
<hr/>					
<hr/>					
HW-Segment					
High Water					
Mark	93,860	93,862	0	88,226	2,961
*					
TX-Transaction (row lock contention)	272	272	0	209	570
**					
TX-Transaction (index contention)	4,564	4,564	0	4,144	34
TX- Transaction	793,989	794,042	0	97	0

3. Verify the Wait Events section of the AWR report; you will likely see something like this:

[Click here to view code image](#)

-> s -second					
-> ms - millisecond - 1000th of a second					
-> ordered by wait time desc, waits desc (idle events last)					
<hr/>					
<hr/>					
Time	Total	Wait	Avg	Waits	%
Event				Waits	-
outs	Time (s)	Wait (ms)	/txn		
<hr/>					
<hr/>					
db file sequential					
read		2,580,474	.0	35,544	14
SQL*Net more data from					
client	**	659,140	.0	5,513	8 C
enq: HW -					
contention			88,910	.0	2,890
log file					
sync		777,146	.0	1,688	
read by other					
session		103,140	.0	929	

SQL*Net break/reset to client				114,782
.0	813	7	0.1	
enq: TX - row lock contention				
***	380	43.4***		557
log file parallel				1466*
write		552,663	.0	394
latch: cache buffers				1
chains		55,203	.0	382
				7

4. Now that you have gathered all this information, find the object that is experiencing this wait event of enq: HW. Run the following query to isolate the objects that are encountering any enq: HW contention between the AWR snapshot interval chosen in your AWR report:

[Click here to view code image](#)

```

SELECT
  sql_id,
  event,
  event_id,
  time_waited,
  current_obj#,
  current_file#,
  current_block#
FROM dba_hist_active_sess_history
WHERE snap_id BETWEEN 1072 AND 1076
AND event LIKE '%HW%CONTENTION%'
AND time_waited > 0
AND current_obj# <> -1
ORDER BY time_waited, event, sql_id;

```

5. Capture the values for current_obj# from the previous query and supply that value to the following query:

[Click here to view code image](#)

```

SELECT
  owner,
  object_name,
  object_type
FROM dba_objects
WHERE object_id = [current_obj# of query above];

```

The following alternative query shows crucial information when you need to isolate the object name, object type, and (most important) the SQL identifier corresponding to the statements that are suffering from enq: HW contention:

[Click here to view code image](#)

```

SQL> col object_name format a30
SQL> col program format a30
SQL> col event format a30
SQL> SELECT DISTINCT

```

```

CURRENT_OBJ#,o.object_name,o.owner,o.object_type,CURRENT_BLOCK#,s
from v$active_session_history a, dba_objects o
where a.current_obj# = o.object_id
and a.event like 'enq%HW%';

```

CUR_OBJ#	OBJ_NAME	OWN	OBJECT_TYPE	CUR_BLOCK#	SESS	SQL_HW -
235738	MLOG\$_ENI_OLTP_	ENI	TABLE	100747	WAITING	0ghs
612464	ITEM_STAR HIST_PEDIDOS	B2W	TABLE	394731	WAITING	9phv

- 6.** Before adding extents to this object, it is advisable to check the object's extent sizes:

[Click here to view code image](#)

```

SELECT COUNT(*), bytes/1024/1024 "MB"
  FROM dba_extents
 WHERE segment_name = 'HIST_PEDIDOS' AND owner='B2W'
 GROUP BY bytes;

COUNT(*)    BYTES
-----
1969801      131072

```

- 7.** Now that you know the largest extent size for the object, you simply add several new extents for it. Use SQL to construct the necessary MODIFY PARTITION commands based on a partition that is already known:

[Click here to view code image](#)

```

SELECT
  'alter table'||table_owner||'.'||table_name||' modify
partition'||partition_name||' lob ('||column_name||')
(allocate extent (size 131072));'
  FROM dba_lob_partitions
 WHERE table_name = 'HIST_PEDIDOS'
   AND partition_name like '%2014%';

```

In our experience, after enough new extents are added—we recommend adding at least 20!—the enq: HW contention will simply disappear from the database's wait events and applications will start running faster again.

While adding extents manually is a good way to relieve this contention *temporarily*, it should be noted that it's not a permanent solution. Because the enq: HW enqueue occurs when we want to extend the high-watermark of the LOB faster than the foreground process can acquire and format the new LOB chunks, we are simply doing

the work proactively for the foreground processes by allocating the new extents manually. However, depending on how much space we need to grow for the incoming workload, 30 or even 50 extents will give the application some time to fulfill that space before it begins to encounter the HW enqueue again. So depending on the situation and expected size and duration of the application workload, allocating extents manually will only temporarily fix the issue.

Also, it's important to note that, as is the case with almost all data definition languages (DDLs), manually allocating an extent does require the database to obtain an exclusive lock on the segment; therefore, depending on the level of concurrency, you won't be able to work around it unless there is sufficient downtime. Thus, it's a good idea to capture these query results into an alert script that would monitor these events in your database and raise a corresponding alert should this situation arise.

BASICFILE LOB Issues: Toward a More Perfect Fix

Here are some other possible solutions to permanently increase the throughput of LOB chunks allocated to the LOB segment:

- Move the LOB segment to a tablespace with a larger UNIFORM extent size. This has proved to be the most effective method because large UNIFORM extents provide more chunks per HW operation.
- Increase the LOB chunk size. First, determine the average size of LOB data via procedure DBMS_LOB.GETLENGTH, and then set the LOB's chunk size to between 120 and 150 percent of the average size of the LOB data.
- Increase the size of chunk size reclamation by setting event 44951 TRACE NAME CONTEXT FOREVER, LEVEL 1024. Once a LOB hits its high-watermark, it will try to reclaim space inside the segment first by purging old images of LOB data based on the setting for PCTVERSION or RETENTION parameters. Setting this event increases how many chunks Oracle will try to reclaim in a single operation, thus making it more effective when enqueue HW waits are encountered. This strategy is complementary with the proper sizing of the LOB's chunk size, as previously described.
- Make sure that Automatic Segment Space Management (ASSM) has been activated for the tablespace in which the LOB resides. This is a prerequisite for SECUREFILE LOBs as well.

Finally, it's important to note that even in Oracle Database Release 11.2.0.1 there was a nasty bug that could corrupt the segment header (thus making the whole segment corrupt) when a manual ALLOCATE EXTENT operation was interrupted, especially for LOB segments. See My Oracle Support (MOS) Note 1229669.1, "Segment header corruption if extent allocation operation is interrupted," for complete information.

BASICFILE versus SECUREFILE LOBs

Oracle Database 11g's new SECUREFILE option for storing LOB datatypes offers

better options for managing LOB datatypes, including LOB *deduplication*, *encryption*, and *compression*. It's therefore strongly recommended that you migrate your BASICFILE LOBs to SECUREFILE LOBs if your database is being upgraded from Oracle 9i or Oracle 10g to Oracle 11g; by doing so, you are likely to improve LOB performance and manageability while simultaneously overcoming several of the problems we discussed in the previous section. However, your database's transition to SECUREFILE LOBs will not necessarily be seamless, as the following scenarios demonstrate.

One issue you may encounter will be readily obvious when an application first inserts data into this new LOB that has been migrated to a SECUREFILE format. If you have an 11.2.0.1 or 11.2.0.2 database, you could have some serious problems when inserting data into this LOB field regardless of whether it's in BASICFILE or SECUREFILE format. (Note that this particular bug has been repaired in 11.2.0.3.)

It's also important to recognize how a LOB's storage parameters affect its performance; here is a brief summary of the most crucial ones:

- CHUNK specifies the smallest unit of LOBSEGMENT and is always a multiple of the DB_BLOCK_SIZE parameter. So if your database's DB_BLOCK_SIZE is 16 KB and you insert 2 KB worth of data into a LOB column, 14 KB is simply wasted space. Because the maximum chunk size is 32 KB, it's important to specify CHUNK so that space isn't being wasted unnecessarily.
- The CACHE directive tells Oracle to retain LOB data in the database buffer cache, whereas the NOCACHE setting never brings LOB data into the buffer cache. The end result is that Oracle will use direct read/writes for NOCACHED data (indicated by the direct path read/write wait event) and perform reads/writes from the database buffer cache for CACHED data (indicated by the db file sequential read wait event). Also note that for LOB data that will only be read and never written, a third option—CACHE_READS—brings LOB data into the buffer cache only during reads, not during writes.
- The LOGGING option enables the logging of changed LOB data to the online redo logs. If you want to improve performance for data manipulation language (DML) executed against your LOB data and know that your application's recovery requirements don't require logging of changed LOB data, consider setting the LOB to NOLOGGING. Note that if you specify NOLOGGING for an in-line LOB, any changes to its data will still be logged in the online redo logs.

LOB New and Old Type Differences

To illustrate the difference between these two LOB formats, let's create two tables, one with a BASICFILE LOB and the other with a SECUREFILE LOB:

1. Create table test1 so that it contains a BASICFILE LOB column. Remember that if you are performing these tests in Oracle Database 12c, you must specify the parameter BASICFILE to use this type of storage because, starting in that release, the default option during LOB creation is now SECUREFILE:

[Click here to view code image](#)

```
CREATE TABLE test1 (col1 CLOB,col2 number)
LOB(col1) STORE AS SECUREFILE(CACHE)
tablespace TS_GG_DATA;
```

When a table containing a LOB segment is created, it creates two different segment types: LOBSEGMENT and LOBINDEX. As its name suggests, LOBINDEX is an index used to access the pages or “chunks” of its corresponding LOBSEGMENT. The following query illustrates:

[Click here to view code image](#)

```
SYS@ORCL AS SYSDBA> SELECT COUNT(*), segment_name, segment_type
  FROM dba_extents
 WHERE segment_name = 'TEST1'
 GROUP BY segment_name, segment_type;
-----  
COUNT(*)    SEGMENT_NAME    SEGMENT_TYPE  
-----  
24    TEST1          TABLE
```

2. Create the test2 table using the SECUREFILE LOB option:

[Click here to view code image](#)

```
CREATE TABLE test2 (col1 CLOB,col2 number)
LOB(col1) STORE AS BASICFILE
tablespace TS_GG_DATA;
```

3. Run the following query to validate the tables and their corresponding LOB datatypes using the DBA_LOBS view:

[Click here to view code image](#)

```
set lines 200
col column_name for a30
SELECT
  table_name,
  column_name,
  segment_name,
  securefile
FROM dba_lobs
WHERE table_name like 'TEST%';
```

TABLE_NAME	COLUMN_NAME	SEGMENT_NAME
TEST1	COL	SYS_LOB0000088
YES		
TEST2	COL1	SYS_LOB0000088
NO		

- 4.** Using a simple loop in an anonymous Procedural Language/Structured Query Language (PL/SQL) block, run the following test to concatenate a value to create different values to insert 1 million rows into both types of LOB segments. Of course, be sure that there is sufficient space in the LOB segments' tablespaces and corresponding Automatic Storage Management (ASM) disk group before attempting this test:

[Click here to view code image](#)

```
SET TIME ON
SET TIMING ON
TRUNCATE TABLE test1;
TRUNCATE TABLE test2;

-- Load TEST1
TT@ORCL > BEGIN
    FOR v_Count_1 IN 1..1000000 LOOP
        INSERT INTO TEST1(col1) VALUES
('testInsertColLOBTest2'||v_Count_1);
        commit;
    END LOOP;
END;
/
```

PL/SQL procedure successfully completed.

Elapsed: 00:03:00.40

-- Load TEST2

```
TT@ORCL > BEGIN
    FOR v_Count_1 IN 1..1000000 LOOP
        INSERT INTO TEST2(col1) VALUES
('testInsertColLOBTest2'||v_Count_1);
        commit;
    END LOOP;
END;
/
```

22:00:55 2 22:00:55 3 22:00:55 4 22:00:55 5 22:00:55

PL/SQL procedure successfully completed.

Elapsed: 00:09:11.61

As these tests prove, inserting into a SECUREFILE LOB is over *three times* more efficient (180 s vs. 551 s) than when inserting the same data into a SECUREFILE LOB on an Oracle 11g Release 2 database.

- 5.** Run the following, and you'll see that there is no significant difference in performance for update statements between BASICFILE and SECUREFILE

LOBs:

[Click here to view code image](#)

```
TT@ORCL > BEGIN
  FOR v_Count_2 IN 1..1000 LOOP
    update TEST2 set col1 = 'testInsertColLOBTest2'||v_Count_2
    where rownum <100;
    commit;
  END LOOP;
END;
/
```

PL/SQL procedure successfully completed.

Elapsed: **00:00:03.93**

```
TT@ORCL > BEGIN
  FOR v_Count_2 IN 1..1000 LOOP
    update TEST1 set col1 = 'testInsertColLOBTest2'||v_Count_2
    where rownum <100;
    commit;
  END LOOP;
END;
/
```

PL/SQL procedure successfully completed.

Elapsed: **00:00:03.62**

6. However, it is important to remember that after data in a LOB segment has been updated and/or deleted, it's possible that the LOB could become seriously fragmented. It's therefore advisable to shrink the LOB using the following command:

[Click here to view code image](#)

```
-- For Oracle Database 10.2 and above:
ALTER TABLE <table name>
  MODIFY LOB (<lob column name>) (SHRINK SPACE [CASCADE]);

-- For Oracle Database 10.1 and below:
ALTER TABLE <table name>
  MOVE LOB (<lob column name>) STORE AS (TABLESPACE
<tablespace name>);
```

Migrating BASICFILE LOBs to SECUREFILE LOBs

Migrating a BASICFILE LOB to a SECUREFILE LOB can be done using one simple command, as follows:

[Click here to view code image](#)

```
TT@ORCL > ALTER TABLE test2 MOVE LOB (col1) STORE AS SECUREFILE  
(TABLESPACE users);
```

Table altered.

Elapsed: 00:00:23.54

However, one disadvantage of this simple migration strategy is that the LOB will be inaccessible for any DML activity while the migration completes. Another way to migrate a BASICFILE to a SECUREFILE LOB in ONLINE mode is to use the DBMS_REDEFINITION package, as our next example shows:

1. If the user account that's going to be used for the LOB migration doesn't have SYSDBA privileges, run the following to grant specific privileges to that account so that it can perform the migration:

[Click here to view code image](#)

```
-- Create the migrating user (if it doesn't yet exist) ...  
grant dba to tt identified by tt123;  
  
-- ... or grant the existing user account the necessary specific  
privileges  
-- so it can use DBMS_REDEFINITION  
grant execute on dbms_redefinition to tt;  
grant alter any table to tt;  
grant drop any table to tt;  
grant lock any table to tt;  
grant create any table to tt;  
grant select any table to tt;  
grant create session to tt;
```

2. Create the tables for this example. This includes the table that will be converted (test3) as well as the table that will be used in the migration (test4) that's known as the interim table:

[Click here to view code image](#)

```
CREATE TABLE test3 (  
    col1 NUMBER PRIMARY KEY,  
    col2 CLOB  
) ; 23:03:10    2  23:03:10    3  23:03:10    4
```

Table created.

Elapsed: 00:00:00.11

3. Insert some example data into test3 for demonstration purposes:

[Click here to view code image](#)

```
TT@ORCL > BEGIN  
FOR v_Count_2 IN 1..10000 LOOP
```

```

        INSERT INTO TEST3(col1,col2) VALUES
(v_count_2,'testInsertColLOBTest3'||v_count_2);
      commit;
    END LOOP;
END;
/
PL/SQL procedure successfully completed.

```

Elapsed: 00:00:01.82

- 4.** Create the interim table (**test4**), which will act as a staging table that will store data being inserted, deleted, or updated while the online migration proceeds:

[Click here to view code image](#)

```

TT@ORCL > CREATE TABLE test4 (
  col1 NUMBER NOT NULL,
  col2 CLOB
) LOB(col2) STORE AS SECUREFILE (NOCACHE);

```

Table created.

Elapsed: 00:00:00.05

- 5.** Start the redefinition of table **test3** using the **START_REDEF_TABLE** procedure of package **DBMS_REDEFINITION**:

[Click here to view code image](#)

```

23:07:09 TT@ORCL > DECLARE
  col_mapping VARCHAR2(1000);
BEGIN
  col_mapping := 'col1 col1, || col2 col2';
  DBMS_REDEFINITION.START_REDEF_TABLE('tt', 'test3', 'test4',
  col_mapping);
END;
/

```

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.97

- 6.** To insure that all of the constraints are copied from the original table to the interim table, invoke the **COPY_TABLE_DEPENDENTS** procedure of **DBMS_REDEFINITION** as follows:

[Click here to view code image](#)

```

23:07:42 TT@ORCL > DECLARE
  error_count pls_integer := 0;
BEGIN
  DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS(
  uname=> 'tt',
  orig_table=> 'test3',

```

```

int_table=> 'test4',
copy_indexes=> DBMS_REDEFINITION.CONS_ORIG_PARAMS,
copy_triggers=> TRUE,
copy_constraints=> TRUE,
copy_privileges=> TRUE,
copy_statistics=> FALSE,
num_errors=> error_count);
DBMS_OUTPUT.PUT_LINE('errors := ' || TO_CHAR(error_count));
END;
/

```

PL/SQL procedure successfully completed.

Elapsed: 00:00:05.89

7. Finish the redefinition of table test3 to table test4 using the FINISH_REDEF_TABLE procedure of DBMS_REDEFINITION:

[Click here to view code image](#)

```

23:08:10 TT@ORCL > EXEC
DBMS_REDEFINITION.FINISH_REDEF_TABLE('tt', 'test3', 'test4');

```

PL/SQL procedure successfully completed.

Elapsed: 00:00:02.52

8. Verify that the column in table test3 that was originally a BASICFILE LOB is now a SECUREFILE LOB via the following query against DBA_LOBS:

[Click here to view code image](#)

```

23:18:33 TT@ORCL > select
owner,table_name,column_name,securefile
      from dba_lobs where table_name='TEST3';

```

OWNER	TABLE_NAME	COLUMN_NAME	SEC
TT	TEST3	COL2	YES

Elapsed: 00:00:00.05

23:18:53 TT@ORCL >

Another way to perform an online, high-availability migration between BASICFILE and SECUREFILE LOBs is to use CTAS (create table as select) in conjunction with Oracle GoldenGate on the same source and target databases to replicate data. This makes it possible to sync up the original and new tables' contents after the CTAS operation that transforms the tables is completed. Once the migration is complete, there will be only a brief window of application downtime while original and new tables are

renamed.

The Impact of PCTFREE on LOBs

Setting an appropriate value for the PCTFREE parameter is critical to avoiding wasted space while creating tables with in-line LOB columns. PCTFREE specifies the minimum percentage of free space in a block that the Oracle database reserves when updating existing rows in a table, so it's mainly designed to inhibit unnecessary row migration when a row piece no longer fits within a block after the row piece is updated and grows in length.

The default value for PCTFREE is 10 percent, but it's not unusual to see it set to a higher value to prevent row migration. For example, if a table's PCTFREE value has been set to 30, Oracle will ensure that 30 percent of the block is reserved for row-length growth after the rows have been updated. When a new row piece is inserted into this block and 70 percent of the space has been used, then the block is marked as full and the next row will be inserted into another block with sufficient free space.

However, a larger-than-normal value for PCTFREE can also mean that huge amounts of free space can be inadvertently wasted when a table contains in-line LOB columns. Our next example demonstrates just how much space can be saved in this situation:

1. Create a simple table with a LOB column and PCTFREE equal to 40 percent, insert 100,000 rows into this new table, and then verify the row count and gather optimizer statistics:

[Click here to view code image](#)

```
SYS@ORCL AS SYSDBA> create table tt.test_pctfree (col1 clob)
PCTFREE 40;
```

Table created.

```
SYS@ORCL AS SYSDBA> BEGIN
  FOR v_Count_2 IN 1..100000 LOOP
    INSERT INTO tt.test_pctfree
      VALUES ('testInsertColLOBTest2' || v_Count_2);
    COMMIT;
  END LOOP;
END;
/
```

```
SYS@ORCL AS SYSDBA> select count(*) from tt.test_pctfree;
```

```
COUNT(*)
```

```
-----
```

```
100000
```

```
SYS@ORCL AS SYSDBA> EXEC DBMS_STATS.GATHER_TABLE_STATS ('TT',
```

```
'TEST_PCTFREE') ;  
PL/SQL procedure successfully completed.
```

2. Check the number of blocks and the average size of each block on this table:

[Click here to view code image](#)

```
SYS@ORCL AS SYSDBA>  
select  
    blocks,  
    avg_space,  
    pct_free  
from  
    dba_tables  
where  
    table_name='TEST_PCTFREE';  
  
BLOCKS      AVG_SPACE  PCT_FREE  
-----  
      1504        3362       40
```

3. Truncate the table and change its PCTFREE of example table to 0 (zero):

[Click here to view code image](#)

```
SYS@ORCL AS SYSDBA> truncate table tt.test_pctfree;  
Table truncated.  
Elapsed: 00:00:00.07  
  
SYS@ORCL AS SYSDBA> alter table tt.test_pctfree pctfree 0;  
Table altered.  
Elapsed: 00:00:00.01
```

4. Insert the same quantity of rows again and regather statistics:

[Click here to view code image](#)

```
SYS@ORCL AS SYSDBA> BEGIN  
    FOR v_Count_2 IN 1..100000 LOOP  
        INSERT INTO tt.test_pctfree VALUES  
        ('testInsertColLOBTest2'||v_Count_2);  
        COMMIT;  
    END LOOP;  
END;  
/  
PL/SQL procedure successfully completed.
```

Elapsed: 00:00:14.13

```
SYS@ORCL AS SYSDBA> EXEC DBMS_STATS.GATHER_TABLE_STATS ('TT',
'TEST_PCTFREE');

PL/SQL procedure successfully completed.
```

Elapsed: 00:00:00.22

```
SYS@ORCL AS SYSDBA>
```

5. Verify the new values of blocks and average space that each block consumes:

[Click here to view code image](#)

```
SYS@ORCL AS SYSDBA> select
  blocks,
  avg_space,
  pct_free
from
  dba_tables
where
  table_name='TEST_PCTFREE';
      2      3      4      5      6      7      8
BLOCKS      AVG_SPACE      PCT_FREE
-----
  874          303            0
```

Elapsed: 00:00:00.00

As this demonstration shows, the number of blocks consumed is much lower: 874 for a PCTFREE of 0 percent versus 1,504 when PCTFREE is 40 percent, or an improvement of about 41 percent. Using a high value for PCTREE in tables with LOB columns can represent a huge waste of space, and this is why some LOB tablespaces become incredibly huge, which tends to make their reorganization difficult. If a table that contains LOBs has not yet been partitioned, there's an excellent chance that an Oracle DBA could be held hostage to this table's space demands because it may become nearly impossible to find a sufficient window of time to recreate the table, even when using DBMS_REDEFINITION.

It is important to emphasize that reducing wasted space within a table with LOBs applies just to inline LOBs. As this example showed, the data inserted was quite small, and most of it was stored within the table itself; for this reason, it was significantly affected by the PCTFREE value of the table. For out-of-line LOBs, however, the table's PCTFREE setting is not a factor because LOB space is allocated and maintained in units of chunks (which are several blocks long) and not on a block-by-block basis as with a table segment.

While LOB datatypes (BLOB, CLOB, NCLOB, and BFILE) do not use the PCTFREE storage parameter or free lists to manage free space, the same strategy to manage LOB space growth can be applied to chunk size when dealing with out-of-line LOBs. Chunk size should therefore be set to a larger value than the average size of LOB data if

updates are expected to increase their size; otherwise, LOB data migration might occur, and application performance may be significantly degraded, because each LOB retrieval will be forced to retrieve twice as many LOB chunks.

One final but crucial caveat to remember is that the chunk size also impacts the size of the redo generated during DML against LOB data. A redo record must be generated for the *entire* chunk, so the size of the chunk has a direct impact on how much redo will be generated, and there will be a definite corresponding impact on DML performance. It's therefore important to be sure to set chunk size no larger than necessary.

Overcoming Poor INSERT Performance

If your application code should encounter unexpectedly poor performance during INSERTs after migrating from BASICFILE to SECUREFILE LOBs in Oracle Database 11g Release 2, be aware that there is a rather pernicious bug in release 11.2.0.1 that is documented in MOS Note 1323933.1, “Securefiles performance appears slower than basicfile LOB.”

To verify that this bug is really what is causing poor INSERT performance (as well as provide a quick workaround for this problem), use the following command to change the `_kdli_sio_fileopen` parameter to an appropriate value at the session level before continuing to perform INSERTs against the table that contains one or more SECUREFILE LOB columns:

[Click here to view code image](#)

```
SQL> alter session set "_kdli_sio_fileopen"='nodsync';
```

The MOS note strongly suggests applying an appropriate patch for your database version to fix the problem permanently. Finally, note that as of release 11.2.0.3, this bug has been repaired.

Summary

From this chapter, you have learned that the more tables with LOB columns you have, the more likely you are to encounter problems regarding LOB performance and maintenance. The recommendation you must always follow is that whenever creating a new table with LOB columns, try to use the best storage parameters that fit exactly the behavior of its corresponding table:

- For tables that will never be updated, be sure to set PCTFREE to a value of zero (0).
- If you do not really need to use LOB storage, consider using VARCHAR or RAW instead.

2. Overcoming Undo Tablespace Corruption

Undo data is a vital component of a database environment, and when something goes wrong with undo tablespaces where this data is stored, the database administrator (DBA) must take quick action to correct the problem. Otherwise, the database could become damaged, and valuable information might be permanently lost. This chapter explains how to manage some undo problems that may occur in real-world scenarios.

Overview of Undo Management

Undo tablespaces were introduced in Oracle Database 9*i*, but not until Oracle Database 11g was the undo management mode set to AUTO by default. Automatic undo management means that Oracle DBAs no longer need to manage rollback segments manually because the Oracle database itself will manage rollback segments in a single tablespace.

As soon as an Oracle database instance starts, it looks for the UNDO_TABLESPACE parameter to determine which undo tablespace should be used; if it does not find the specified undo tablespace, the database will not start and an error will be generated in the database's alert log. If the UNDO_TABLESPACE parameter is not being used, the database will try to find *any* undo tablespace in the database, and if it cannot find one, it will start to use rollback segments stored in the SYSTEM tablespace. This is not an optimal situation for database performance and is definitely not recommended. Again, in this situation, an alert message will be written to the database instance's alert log.

The Importance of UNDO_RETENTION

When a transaction begins, it is assigned a specific undo segment in the undo tablespace. That undo segment, like all segments in a tablespace, consists of several undo extents. Until the transaction is either committed or rolled back, the undo extents for a transaction have a status of ACTIVE. However, once the transaction is completed, if *other* statements need those extents to maintain a read-consistent view of the data as of the system change number (SCN) at which the statement began, then the related undo extents are marked as UNEXPIRED and are retained until they are no longer needed to maintain read consistency. If there are no related statements that need to maintain a read-consistent view once the transaction has completed, then the undo extents will be marked as EXPIRED and other new transactions can reuse them.

The UNDO_RETENTION initialization parameter specifies the time in seconds that Oracle will *attempt* to retain undo extents before they can be overwritten by transactions that need undo space. When a transaction has already been committed and its execution time is longer than the value of UNDO_RETENTION, the status of the transaction's undo extent changes to EXPIRED so that other transactions can reuse them. Otherwise, if the transaction itself was of short duration, then the undo extents will remain in UNEXPIRED status until the amount of time since the transaction was committed

exceeds the time specified by `UNDO_RETENTION`.

The `UNDO_RETENTION` parameter also interacts with the undo tablespace differently depending on whether the tablespace's underlying datafiles are autoextensible. If the undo tablespace's datafiles are set to a fixed size, then Oracle will ignore the `UNDO_RETENTION` parameter because it cannot guarantee the value specified for the retention period without having to increase the size of the undo tablespace. On the other hand, if the undo tablespace's datafiles are set to `AUTOEXTEND` instead, then Oracle will try to gracefully honor the `UNDO_RETENTION` parameter value by increasing the size of the undo tablespace's datafiles instead of overwriting unexpired undo transactions.

The ultimate impact of the `UNDO_RETENTION` parameter and its interaction with its tablespace can be verified by checking the value for the `TUNED_UNDORETENTION` column of the `V$UNDOSTAT` dynamic view. This value is calculated and adjusted on the fly. The way it's calculated depends on the `MAXQUERYLENGTH` of the query captured within the time interval, as well as on the total space usage of the undo tablespace. If your database's application workload consists of long-running report-like queries, and its undo tablespace's datafiles are relatively large and of fixed sized, it is unlikely to encounter a very high value for `TUNED_UNDORETENTION`.

This situation can sometimes cause problems for database application performance. For example, if `TUNED_UNDORETENTION` is set relatively high, then depending on the database's application workload, it is possible that a lot of `UNEXPIRED` undo extents will result, and the database will most likely reuse them during peak data manipulation language (DML) execution periods. Reusing `UNEXPIRED` undo extents is potentially more expensive in terms of latches and internal work than reusing `EXPIRED` segments. In this situation, it may be advantageous to disable autotuning of retention values by setting the hidden initialization parameter `_UNDO_AUTOTUNE` to a value of `FALSE`; alternatively, the `_HIGHTHRESHOLD_UNDORETENTION` parameter can be used to specify an upper bound value for `TUNED_UNDORETENTION`.

Note

Be sure to review MOS Note 742035.1, "Contention under auto-tuned undo retention," and its notes about Bug 7291739 before implementing this solution because it identifies a particularly pernicious bug for Oracle databases prior to release 11.2.0.1.

Tuning `UNDO_RETENTION`

The `DBMS_UNDO_ADV` package provides some excellent feedback to help determine the best value for the `UNDO_RETENTION` parameter:

[Click here to view code image](#)

```
SQL> SELECT DBMS_UNDO_ADV.getLongest_query(SYSDATE-1/24, SYSDATE)
   AS best_undo_time FROM dual;
```

```
BEST_UNDO_TIME
```

```
-----  
845
```

This output indicates that the longest-running query on this instance took 845 seconds. Based on this output, then, 845 seconds is a good starting point for the undo retention parameter. Another query that can help identify an appropriate value for UNDO_RETENTION follows:

[Click here to view code image](#)

```
SQL> SELECT DBMS_UNDO_ADV.REQUIRED_RETENTION(SYSDATE-30, SYSDATE)  
AS reqd_retn FROM dual;
```

```
REQD_RETN  
-----  
1699
```

This query leverages the REQUIRED_RETENTION procedure of the DBMS_UNDO_ADV package to capture the execution time of the longest-running query in the last 30 days. If you use the same input parameter in the LONGEST_QUERY procedure of the first example, you will see that the same value will be returned.

Note

DBMS_UNDO_ADVISOR is an undocumented procedure. For more information about using it, see MOS Note 1580225.1, “What is the Undo Advisor and how to use it through the DBMS_UNDO_ADV package.” Much of this information is also available via the Undo Advisor in Enterprise Manager Database Console, Grid Control, and Cloud Control.

DTP, XA, and Rollback Segments

Database applications that connect to multiple databases via database links or that use XA (eXtended Architecture) for distributed transaction processing (DTP) may occasionally experience issues with pending transactions. When such issues arise, it's not unusual to encounter extreme contention, often to the point that the database instance “hangs,” which may cause all database DML operations to halt until the problem is resolved. The following checklist and corresponding scripts will help to ascertain if an Oracle database instance is in danger of hanging due to issues with DTP via XA:

1. Run the following query to determine if there are any pending distributed transactions and, if so, build a set of commands to force any pending distributed transactions to commit:

[Click here to view code image](#)

```
SQL> SET HEADING OFF  
SELECT 'commit force '''||local_tran_id||''';' FROM
```

```

dba_2pc_pending;
SQL> SQL>
commit force '151.23.987365';
commit force '155.29.1615583';
commit force '231.10.1069716';
commit force '237.18.648972';
commit force '238.15.811599';
commit force '36.5.1329177';
commit force '393.41.746115';
commit force '4733.28.915649';
commit force '613.17.686683';

9 rows selected.

```

This output identifies nine pending distributed transactions that must be resolved by forcing them to explicitly commit. While it's also acceptable to roll back these transactions, it makes much better sense to commit these transactions because committing them diminishes the chance of losing any transaction data.

2. Next, execute the commands previously generated:

[Click here to view code image](#)

```

SQL> commit force '151.23.987365';
commit force '155.29.1615583';
commit force '231.10.1069716';
commit force '237.18.648972';
commit force '238.15.811599';
commit force '36.5.1329177';
commit force '393.41.746115';
commit force '4733.28.915649';
commit force '613.17.686683';
Commit complete.
Commit complete.
...
Commit complete.

```

3. Run this next query to build a set of transactions to execute on this instance to purge any lost transactions from the database:

[Click here to view code image](#)

```

SQL> SELECT 'Execute
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('' || local_tran_id || '')';cor
FROM dba_2pc_pending;

Execute
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('151.23.987365');commit;
Execute
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('155.29.1615583');commit;
Execute
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('231.10.1069716');commit;

```

```

Execute
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('237.18.648972');commit;
Execute
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('238.15.811599');commit;
Execute
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('36.5.1329177');commit;
Execute
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('393.41.746115');commit;
Execute
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('4733.28.915649');commit;
Execute
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('613.17.686683');commit;

9 rows selected.

```

Simply copy and paste the output from the previous command into a SQL*Plus session to clean up any additional pending distributed transactions. However, if the distributed transactions persist, it may be necessary to investigate and resolve one last possibility: the status of the rollback segments associated with these purged transactions may have changed from ONLINE to PARTLY AVAILABLE.

4. The PARTLY AVAILABLE status means that the rollback segment contains data for an in-doubt transaction or a recovered distributed transaction. The database instance's background recoverer process (RECO) should automatically resolve in-doubt or pending transactions; however, it still may be necessary for the Oracle DBA to intervene to reset the affected rollback segments to their proper status of ONLINE. Doing so may cause the database to hang if transactions other than the purged distributed transactions were using those rollback segments.

To correct this situation, run the following query to locate any rollback segments still marked as partly available:

[Click here to view code image](#)

```

SQL> SELECT
  'ALTER ROLLBACK SEGMENT "'||segment_name||'" ONLINE;'
  FROM dba_rollback_segs
 WHERE status LIKE 'PARTLY_AVAILABLE';

```

5. If executing this command fails to bring the rollback segments back online, some additional remedial action is required. Take the rollback segment offline and then verify that its status is now OFFLINE instead of PARTLY AVAILABLE using the following commands:

[Click here to view code image](#)

```

SQL> ALTER ROLLBACK SEGMENT "_SYSSMU168$" OFFLINE;

SQL> SELECT status
  FROM dba_rollback_segs
 WHERE SEGMENT_NAME = '_SYSSMU168$';

```

- Once the rollback segment is taken offline successfully, drop it and then immediately re-create it, as follows:

[Click here to view code image](#)

```
SQL> DROP ROLLBACK SEGMENT "_SYSSMU168$";  
  
SQL> CREATE ROLLBACK SEGMENT '_SYSSMU168$' TABLESPACE UNDOTBS1;
```

This step should resolve any rollback segment-related issues, and database application user sessions should now be able to proceed without any danger of halting or causing the database instance to hang.

Other Unusual Rollback and Undo Segment Issues

Sometimes, a session can hang when the DBA tries to force a transaction to commit or roll back (COMMIT FORCE or ROLLBACK FORCE). In the database, we can see the session hang as it waits on the event “free global transaction table entry.” Other rare instances can occur when underlying metadata about the transaction is corrupted. In this case, it will most likely require manually deleting and re-inserting the transaction information within the appropriate internal tables. See MOS Note 401302.1, “How to resolve stranded DBA_2PC_PENDING entries,” for more information.

Recovering from Undo Tablespace Corruption

If you have been an Oracle DBA for any length of time, it’s quite likely that you’ve encountered an ORA-1578, ORA-7445, or ORA-600 error during your career. These are indications of a symptom that all DBAs love to hate: *corruption*. Fortunately, corruption within Oracle database structures doesn’t occur very often, but when it does, it must be dealt with carefully and immediately. It is also crucial to determine the true root cause (or causes) of the corruption and employ proactive measures to ensure it will not reoccur.

The simplest technical definition of corruption in an Oracle database is that it is an inconsistency in either a data block or buffer cache memory structure. Fortunately, there are well-known methods to prevent, detect, and repair corruption depending on the type of corruption encountered. The remainder of this chapter discusses the appropriate approaches to detecting, repairing, and then proactively preventing corruption in rollback segments in the SYSTEM tablespace and undo segments in an undo tablespace.

Preventing, Detecting, and Repairing Corruption

First, it’s important to distinguish two broad types of corruption: memory corruption and block corruption.

An ORA-7445 error is generally indicative of memory corruption, and the root causes of this error are legion: It could be caused by a malfunction in the database server’s dynamic random-access memory (DRAM), storage network, I/O controller memory, or even the internal disk drive’s memory cache. This type of error requires careful testing

and retesting to establish the root cause, and it may even require intervention by the appropriate hardware technicians to diagnose, isolate, and repair the ultimate source of the problem.

An ORA-600 or ORA-1578 error typically indicates either logical or physical corruption of a database block or memory buffer. *Physical* corruption means that the database cannot recognize the block or buffer correctly; perhaps the block header has been corrupted, or its actual size no longer matches its expected size. *Logical* corruption implies that even though the block or buffer *structure* is still intact, its *contents* are essentially gibberish and are no longer legible to the database. As these issues are mostly within an Oracle DBA's jurisdiction, the remainder of this section focuses on how to prevent, detect, and repair this type of corruption.

Prevention. A smart Oracle DBA spends her time anticipating a problem before it happens. In the case of corruption, she follows recommended best practices, including avoiding single points of failure in the database's hardware and network, activating appropriate monitoring of the hardware and the database using tools such as Oracle Enterprise Manager 12c Cloud Control, and activating appropriate values for specific initialization parameters to limit or eliminate database corruption.

Detection. Unfortunately, detection of database corruption often first appears when a user reports a strange error message on his screen, usually followed by a frantic search of the corresponding database's alert log to find the ORA-1578 error that matches the database block that's been corrupted. However, it's also possible to detect block corruption outside of application errors in the following ways:

- During regularly scheduled Recovery Manager (RMAN) backup processing
- After running the RMAN BACKUP VALIDATE command against a database, tablespace, or datafile, either with or without the optional CHECK LOGICAL clause to identify logical corruption
- Running the DBMS_REPAIR package against a specific database segment
- Executing the database verification utility, DBVerify, against a specific datafile

In these cases, it's likely that the V\$DATABASE_BLOCK_CORRUPTION dynamic view will contain a list of the corrupted blocks and that information can be leveraged effectively for repair of the corrupted blocks.

Repair. Once corruption has been detected, it's the job of an Oracle DBA to repair the damage as quickly as possible. The appropriate method to repair the damage depends on the type of corruption detected, the criticality of the object to the database application, the amount of downtime that the corruption is causing or may cause, the availability of RMAN backups on either disk or tape, and even the specific disaster recovery environment that has been implemented. For example, a corrupted database block on disk in the oldest historical partition of a partitioned table containing data from 20 years ago is obviously of less immediate concern than a block in the hottest partition of the same table.

Handling Memory Corruption

When blocks are being read into an Oracle database's buffer cache, Oracle background processes will verify the blocks being read from disk. The background processes compare the incarnation number (INC) and sequence number (SEQ) in the header of the data block with the INCSEQ structure in the database server's baseboard to make sure that the block versions are exactly the same. This verification should prevent an Oracle database from reading a block from disk that contains a corrupted block header.

Even so, it is still possible for memory corruption to occur. Here are several possible vectors:

- Oracle software bugs
- Operating system software bugs
- Non-Oracle programs writing to the same memory address being used by Oracle
- Hardware problems, including failing DRAM components

Prevention. One of the best ways to prevent unnecessary corruption of Oracle database blocks is to set the `DB_BLOCK_CHECKSUM` initialization parameter to its recommended setting of `FULL`. The Oracle database will then verify that a block is logically consistent before it's written back to disk. Be aware, however, that there is a potential performance penalty of up to 10 percent if the application workload is skewed heavily toward DML because of the additional verification processing this block checking entails.

Starting with Oracle Database 11g, it's even simpler to set the appropriate values to guard against corruption via the `DB_ULTRA_SAFE` initialization parameter. This parameter accepts just three possible values:

- As might be expected, `OFF` sets the lowest acceptable values: `DB_CLOCK_CHECKING`, `DB_BLOCK_CHECKSUM`, and `DB_LOST_WRITE_PROTECT` are deactivated (but can be set separately to appropriate values if desired).
- `DATA_ONLY` sets `DB_CLOCK_CHECKING` to `MEDIUM`, `DB_LOST_WRITE_PROTECT` to `TYPICAL`, and `DB_BLOCK_CHECKSUM` to `FULL`.
- `DATA_AND_INDEX` sets `DB_BLOCK_CHECKING` and `DB_CLOCK_CHECKSUM` to `FULL` and `DB_LOST_WRITE_PROTECT` to `TYPICAL`.

Detection. Memory corruption is typically detected by monitoring the database instance alert log for ORA-600 errors; however, it's just as likely that they will be reported when database application users unexpectedly receive an ORA-600 error or when error messages that are related to memory corruption are reported to the Oracle DBA.

Repair. Restarting a database instance is one potential method to clear at least some memory corruption errors. However, the Oracle hanganalyze utility is an excellent tool for collecting more information, and it's likely that Oracle Support will require more

detailed information when opening a service request to solve the memory corruption issues. Collecting hanganalyze information is relatively straightforward, and there are two methods to choose from: SQL*Plus and Oradebug.

Invoking Hanganalyze via SQL*Plus

One of the best methods to gather information about database performance problems when the database is hanging is to use the HANGANALYZE procedure as follows:

1. Start a SQL*Plus session as SYSDBA and run the following command:

[Click here to view code image](#)

```
ALTER SESSION SET EVENTS 'immediate trace name HANGANALYZE level 3';
```

2. Wait 1 minute and run this same command again:

[Click here to view code image](#)

```
ALTER SESSION SET EVENTS 'immediate trace name HANGANALYZE level 3';
```

3. Wait 1 more minute and again run the same command:

[Click here to view code image](#)

```
ALTER SESSION SET EVENTS 'immediate trace name HANGANALYZE level 3';
```

4. Wait 1 more minute and then run this command to set another event:

[Click here to view code image](#)

```
ALTER SESSION SET EVENTS 'immediate trace name SYSTEMSTATE level 266';
```

5. Wait 1 more minute, and run the same event command again:

[Click here to view code image](#)

```
ALTER SESSION SET EVENTS 'immediate trace name SYSTEMSTATE level 266';
```

6. Finally, wait 1 more minute and run the same command once more:

[Click here to view code image](#)

```
ALTER SESSION SET EVENTS 'immediate trace name SYSTEMSTATE level 266';
```

7. Exit your SQL*Plus session, gather all trace files generated in your database's USER_DUMP directory, and send them on to Oracle Support for expert analysis.

Using Hanganalyze without a Persistent SQL*Plus Connection

If you are unable to log in to the affected database with SQL*Plus at the exact moment

that the hang occurs, this method will work instead:

1. Log in to SQLPLUS using the following command:

```
sqlplus -prelim / as sysdba
```

2. Set the pid of your session using the following command:

```
oradebug setmypad
```

3. Set the ulimit of this session:

```
oradebug unlimit
```

4. Run the HANGANALYZE command:

```
oradebug hanganalyze 3
```

5. Wait 3 minutes and run it again:

```
oradebug hanganalyze 3
```

6. Wait 3 more minutes and run it once more:

```
oradebug hanganalyze 3
```

7. Wait 3 more minutes, then run this command:

```
oradebug dump systemstate 10
```

8. Now wait 7 minutes and run the same command again:

```
oradebug dump systemstate 10
```

9. Wait 7 minutes and run the same command once more:

```
oradebug dump systemstate 10
```

10. Cancel the tracing, stop the session, and send the appropriate files from `USER_DUMP_DEST` to Oracle Support.
-

Note

If you are analyzing an issue with an Oracle Real Application Cluster (RAC) database, be sure to invoke `oradebug setinst all` to capture all instance information properly!

Handling Logical Corruption

Poorly designed applications can cause logical corruptions because they do not check integrity of data. Some bugs related to the Oracle optimizer could also lead to logical corruptions.

Prevention. Excessively test application code before deploying it to a production

environment. Also, be sure to keep your Oracle database patched to the most recent patch set release and strive to upgrade databases to the most stable version for your current release.

Detection. Logical corruptions caused by application errors are difficult to detect; the best professionals to find errors of this kind are the developers of the application in question.

Repair. The same professional who finds the error—usually, the application developer—will be the one to fix it.

Overcoming Media Corruption

Media corruption has several potential vectors, including (but not limited to) the following:

- Hardware problems in the database server, storage area network, or physical storage components
- Misconfigured parameters in the operating system storage area network
- I/O controller issues
- Logical volume issues
- User errors (e.g. a user inadvertently replaced or deleted a datafile)
- Unexpected bugs in Oracle database or Automatic Storage Management (ASM) software
- Bugs in the database server's operating system software

Prevention. Depending on the cause, here are some recommended best practices to prevent media corruption:

- Practice the principle of least privilege to ensure that no one can inadvertently overwrite or delete a datafile.
- Have a good backup policy, and always test a full restore at regular intervals to ensure full database recoverability can be performed when it is least expected.
- Install a monitoring tool such as Oracle Grid Control 11g or Cloud Control 12c so alerts can be raised whenever an ORA error occurs, or deploy shell scripts to monitor the alert log.
- Always make changes to your database in a test environment before deploying any application changes that might cause corruption.

Detection. ORA errors will be found in alert log files, and sometimes users accessing the database will report these errors. In addition, DBVerify can be used to find physical corrupted blocks in datafiles.

Repair. Datafile block corruption can be repaired in many different ways, and the appropriate method depends on the type of block that needs recovery.

File Header Block Corruption

As the name implies, the header block keeps track of information about the datafile as well as the status of the file and the last time that a RESETLOG operation was performed. Also, every time a checkpoint occurs, the file header block is updated with checkpoint information.

Detection. Fortunately, there are several ways to detect file header block corruption:

- Review the database's alert logs for any ORA errors.
- Run DBVerify against datafiles.
- Issue the `ALTER SYSTEM CHECK DATAFILES` command, and then check the database's alert log afterwards to see if any errors were generated.
- Issue `SELECT FILE#, ONLINE_STATUS` from `V$RECOVER_FILE` to check for any datafile that needs recovery, and then review the output to find any information about corrupted blocks.
- The error ORA-1578 is always generated when a file header becomes corrupted, and this error message will display the block number and data file number of the affected datafile.

Repair. The only repair option in this situation is to restore the block from a valid datafile backup.

Data Dictionary Object Block Corruption

Data dictionary object blocks are crucial because they belong to objects that reside in the `SYSTEM` tablespace and contain information that the database cannot live without—the objects that comprise the database itself.

Detection. Again, a couple of methods are available to detect block corruption:

- Running DBVerify against the `SYSTEM` tablespace's datafiles will identify which blocks are corrupted.
- The database's alert log file will show ORA-1578 errors related to the `SYSTEM` tablespace's datafiles.

Repair. As the `SYSTEM` tablespace is absolutely crucial to the database's operation, this situation must be dealt with immediately upon detection of corruption.

The best method is to restore the `SYSTEM` tablespace from a valid backup and recover the block/datafile. However, remember that this recovery will require shutting down the database instance and restarting it in `MOUNT` mode before recovery can continue.

If you have a disaster recovery site provided by either Data Guard physical replication or GoldenGate logical replication, you may consider switching over to that system until the data dictionary corruption is repaired.

Undo Header and Undo Block Corruption

Transactions utilize undo segments in two cases:

- If the database detects that a transaction has failed, it will automatically roll back

that transaction to maintain data integrity.

- When an application user explicitly cancels the transaction through a web browser or other interface, the application issues the ROLLBACK command to undo everything that was committed within the current transaction.

Undo segments are therefore crucial for transaction consistency, so any undo tablespace corruption issues must be handled with extreme urgency.

Detection. Again, several methods are available to detect block corruption:

- Run DBVerify on the undo tablespace's datafile to discover any corruption.
- The database's alert log will register ORA errors when undo corruption is detected. If the corrupted undo segment is not offline, an ORA-1545 is typically raised, while a query that is unable to locate the appropriate version of a block from an undo segment for read consistency will typically raise an ORA-1578 error.
- Query the V\$ROLLSTAT dynamic view; if any undo segment shows a status of NEEDS RECOVERY, then undo corruption is likely.

Repair. Recovering from undo segment corruption is a bit trickier than other corruption recovery scenarios.

If an undo segment is found to contain a corrupted block, a good first attempt to solve the problem is to simply take the undo segment offline and then drop the segment.

If that method doesn't work, or if it is impossible to drop the undo segment, then it will be necessary to restore and recover the undo tablespace from a valid backup. However, as with SYSTEM tablespace recovery, doing so will require shutting down the database instance and restarting it in mount mode before recovery of the undo tablespace can continue.

One *unsupported* repair method involves the following procedure:

1. If the corrupted undo block is part of a table, index, or cluster, then activate the following event. It will write information about the object number into a trace file in the USER_DUMP_DEST directory for the current session:

[Click here to view code image](#)

```
ALTER SYSTEM SET EVENT "10015 trace name context forever, level 10";
```

2. Review the trace file to find the object number, and then run the following query:

[Click here to view code image](#)

```
SQL> SELECT owner, object_name, object_type, status
      FROM dba_objects
     WHERE object_id = <object# from trace file>;
```

3. Drop the object and restore only this object from a valid backup.

If the block belongs to the undo header, then the repair is much more tedious:

1. List the corrupted undo segment by setting unsupported parameter `_CORRUPTED_ROLLBACK_SEGMENTS` to a value of rollback segments' names.
2. Run a full database export.
3. Create an empty database.
4. Run a full database import.

Summary

As this chapter demonstrates, the undo tablespace is a vital part of any Oracle database after Oracle Database 10g Release 1. It's therefore crucial for an Oracle DBA to be able to detect, diagnose, and repair any problem related to the undo tablespace and its corresponding undo segments and extents. If the DBA ignores any of the potential issues this chapter has described, database applications may encounter unacceptable response times, the database instance may unexpectedly hang, and the database may even stop operating completely or be impossible to open or reopen. Following are the best practices mentioned in this chapter:

- Be sure that you understand the intricacies of how an Oracle database uses its undo tablespace, including how the `UNDO_RETENTION` parameter setting determines the potential size of an undo tablespace as well as how long an unexpired undo extent will be retained for purposes of read consistency.
- Use the Undo Advisor tool. Whether accessed via Oracle Enterprise Manager or the undocumented `DBMS_UNDO_ADVISOR` package, it can be invaluable for tuning undo tablespace usage and related initialization parameter settings.
- Leverage the `hanganalyze` utility to accurately ascertain the cause of an apparent database hang and decide which approach to overcome the hang is most warranted.
- Learn how to distinguish the types and potential causes of block media corruption, and be prepared to detect, diagnose, and correct corruption in any circumstance because this type of corruption can cause an Oracle database to hang and may even require database downtime to repair the damage.

3. Handling GC Buffer Busy Wait Events

Oracle Database is the most complete and simultaneously complex database in today's database marketplace. Every time Oracle launches a new release, a lot of new features are made available, and anyone who works as an Oracle database administrator (DBA) knows how difficult it is to stay current on all feature sets. It is entirely possible that an Oracle DBA may be required to manage multiple databases that are running on completely different releases of Oracle. To add to this complexity, each database may have several distinct features enabled, including various versions of table and index partitioning, Oracle Advanced Compression, replication between databases using either Oracle Streams or Oracle GoldenGate, Oracle Real Application Clusters (RAC), and many others. A RAC database is one of the most difficult environments to administer because its different architecture exists mainly to offer high availability for database applications. This chapter therefore focuses on the global cache (gc) buffer busy wait event, one of the most commonly encountered wait events in a RAC database.

Overview of Buffer Busy Wait Events

As its name implies, any buffer busy wait event in an Oracle database simply means that at least one session is waiting for a buffer in the instance's database buffer cache to become available. In Oracle Database versions prior to Release 10.1, there was just one event called *buffer busy wait*, but starting with that release a new event, *gc buffer busy*, was added to the mix for RAC databases to help monitor buffer busy waits related to cache fusion.

Starting with Oracle Database 12c Release 1, the *Oracle Database 12c Reference Guide* (Oracle, 2016) now describes four wait events to consider when discussing this type of wait event:

- **Buffer busy:** A session cannot pin the buffer in the buffer cache because another session has pinned the buffer.
- **Read by other session:** A session cannot pin the buffer in the buffer cache because another session is reading the buffer from disk.
- **GC buffer busy acquire:** A session cannot pin the buffer in the buffer cache because another session is reading the buffer from the cache of another database instance in the RAC cluster database.
- **GC buffer busy release:** A session cannot pin the buffer in the buffer cache because another session on another database instance in the RAC cluster database is bringing that buffer from a different instance's buffer cache into its own cache so it can be pinned.

This chapter focuses on some important topics related to the gc buffer busy wait event:

- How to leverage the ORAchk utility to help keep a RAC database healthy
- How to use Automatic Workload Repository, Automatic Database Diagnostic

Monitor, and Active Session History to find important information regarding gc buffer busy wait events

- How to investigate whether a RAC database is experiencing an unnecessarily high frequency of the gc buffer busy wait event, and how to handle and repair this situation

Leveraging the ORAchk Utility

The original RACcheck tool was renamed to ORAchk in early 2015, and for an excellent reason: it now not only works for RAC databases but also supports a lot of new features and products, such as Oracle GoldenGate, Oracle E-Business Suite, Oracle Sun Systems, and Oracle Enterprise Manager Cloud Control. ORAchk has thus been transformed into a serious audit tool for all Oracle database environments.

The following examples show how simple it is to deploy and use the ORAchk tool within a real-world Exadata X4-2 full rack environment. A full rack contains eight database nodes and 14 Exadata storage cells, but in this example, the full rack has been subdivided between two RAC databases, one using six nodes and the other using the remaining two nodes. Both databases leverage the Automatic Storage Management (ASM) disk groups resident across all 14 storage cells.

Installing ORAchk

Complete details about how to download, install, and leverage the ORAchk utility can be found in MOS Note 1268927.2, “ORAchk—Health checks for the Oracle stack.”

1. Download the file described in the MOS note previously mentioned, transfer the file to one of the Exadata database nodes, and execute the following commands while logged in as the root user or a user with root permissions. (ORAchk requires that a user with root permissions must install the utility.)

[Click here to view code image](#)

```
[root@ex01dbadm01 tmp]# cd  
[root@ex01dbadm01 ~]# mkdir ORAchk  
[root@ex01dbadm01 ~]# cd ORAchk/  
[root@ex01dbadm01 ORAchk]# unzip /tmp/orachk.zip  
Archive: /tmp/orachk.zip  
  inflating: UserGuide.txt  
  inflating: collections.dat  
  inflating: rules.dat  
  ...  
  inflating: .cgrep/lcgrep6  
  inflating: .cgrep/profile_only.dat  
  inflating: .cgrep/auto_upgrade_check.pl  
  inflating: .cgrep/diff_collections.pl  
  inflating: CollectionManager_App.sql  
  inflating: orachk
```

2. After unzipping all Exachk files, review the appropriate readme file to see if there

are any new instructions pertaining to installing and configuring Exachk.

3. Run the ORAchk utility and answer all questions, as shown. Note that the warning about the binary being older than 120 days can be ignored if the file has just been downloaded:

[Click here to view code image](#)

```
[root@ex01dbadm01 ORAchk]# ./orachk
This version of orachk was released on 09-Oct-2014 and its older
than 120 days. No new version of orachk is available in
RAT_UPGRADE_LOC. It is highly recommended that you download the
latest version of orachk from my oracle support to ensure the
highest level of accuracy of the data contained within the
report.
```

Do you want to continue running this version? [y/n] [y]

CRS stack is running and CRS_HOME is not set. Do you want to set
CRS_HOME to /u01/app/11.2.0.4/grid? [y/n] [y]

...

4. You can deploy the ORAchk utility so that it will automatically verify whether a newer version is available, as well as send an email to the appropriate account after each regularly scheduled execution, as follows:

[Click here to view code image](#)

```
[root@ex01dbadm01 ORAchk]#
./orachk -set "AUTORUN_SCHEDULE=3 1 *
*;NOTIFICATION_EMAIL=paulo.portugal@f2c.com.br"

Created AUTORUN_SCHEDULE for ID[orachk.default]

Created NOTIFICATION_EMAIL for ID[orachk.default]

[root@ex01dbadm01 ORAchk]#
```

The AUTORUN_SCHEDULE can be interpreted as follows:

[Click here to view code image](#)

```
AUTORUN_SCHEDULE * * * *           :- Automatic run at specific time
in daemon mode.

- - - -
? ? ? ?
? ? ? +----- day of week (0 - 6) (0 to
6 are Sunday to Saturday)
? ? +----- month (1 - 12)
? +----- day of month (1 -
31)
+----- hour (0 - 23)
```

5. To verify whether ORAchk has been customized in any way, run the following command:

[Click here to view code image](#)

```
[root@ex01dbadm01 ORAchk]# ./orachk -get all

ID: orachk.default
-----
AUTORUN_SCHEDULE = 3 1 * *
NOTIFICATION_EMAIL = paulo.portugal@f2c.com.br

[root@ex01dbadm01 ORAchk]#
```

6. Finally, you can configure ORAchk to start automatically after any server reboot by running the following command:

```
./orachk -initsetup
```

Results of ORAchk Execution: Sample Output

After the ORAchk utility finishes its execution, it generates a report in HTML format that lists all information regarding the environment in which the tool was executed. [Figure 3.1](#) shows the main page of this output that lists any findings with a status of FAILED or WARNING that should be investigated.

Findings Needing Attention

FAIL, WARNING, ERROR and INFO finding details should be reviewed in the context of your environment.

NOTE: Any recommended change should be applied to and thoroughly tested (functionality and load) in one or more non-production environments before applying the change to a production environment.

Database Server

Status	Type	Message	Status On	Details
FAIL	OS Check	Database server disk devices tune2fs check interval should be set to 0	ex01dbadm03	View
FAIL	ASM Check	You do not have enough space to reestablish redundancy in at least one disk group after disk failure	All ASM Instances	View
FAIL	SQL Check	Table AUD5[FGA_LOGS] should use Automatic Segment Space Management for ACERPRO	All Databases	View
FAIL	OS Check	Database control files are not configured as recommended	All Database Servers	View
FAIL	SQL Check	Some bigfile tablespaces do not have non-default maxbytes values set	All Databases	View
FAIL	OS Check	Database parameter Db_create_online_log_dest_n is not set to recommended value	All Database Servers	View

Figure 3.1 Summary section of an ORAchk report

Reviewing the details for one of the steps marked as FAILED is as simple as clicking on the related link, as shown in [Figure 3.2](#).

Infiniband Switch counters on all switches

	<p>Benefit / Impact:</p> <p>Verifying that there are no high, persistent InfiniBand network error counters helps to maintain the InfiniBand network at peak efficiency.</p> <p>The impact of verifying there are no InfiniBand network errors is minimal.</p> <p>Risk:</p> <p>Without verifying the InfiniBand network error counters, there is a risk that a component will degrade the InfiniBand network performance, yet may not be sending an alert or error condition.</p> <p>Action / Repair:</p> <p>Linux</p> <p>To verify there are no InfiniBand network errors, use the version appropriate command shown below as the "root" userid on one of the database servers:</p> <pre>For image version >= 11.2.3.3.0 (ofed version >= 1.5.5): ibqueryerrors.pl -rR -s PortRcvSwitchRelayErrors,PortXmitDiscards,PortXmitWait,VL15Dropped For image verison < 11.2.3.3.0: ibqueryerrors.pl -rR -s RcvSwRelayErrors,XmtDiscards,XmtWait,VL15Dropped The output should be similar to (image version >= 11.2.3.3.0): ## Summary: 172 nodes checked, 0 bad nodes found ## 1023 ports checked, 0 ports have errors beyond threshold ## Thresholds: ## Suppressed: PortRcvSwitchRelayErrors PortXmitDiscards PortXmitWait VL15Dropped - OR (image version < 11.2.3.3.0) - Suppressing: RcvSwRelayErrors XmtDiscards XmtWait VL15Dropped There may or may not be additional output. Please refer to the "NOTE"s at the end of this section. Solaris Use the command shown below as the "root" userid on one of the database servers: ibqueryerrors.pl -rR -s PortRcvSwitchRelayErrors,PortXmitDiscards,PortXmitWait,VL15Dropped The output should be similar to: Suppressing: PortRcvSwitchRelayErrors PortXmitDiscards PortXmitWait VL15Dropped</pre>
Recommendation	

Figure 3.2 Detail section of an ORAchk report

As this ORAchk detailed report section shows, there are issues that the Oracle DBA can research to isolate other potential causes behind a gc buffer busy wait event. In this case, there is an issue with the configuration of the Exadata's InfiniBand network components that may be contributing to poor application performance.

Isolating GC Buffer Busy Waits

The three key tools that every Oracle DBA who has worked on a database since Oracle 10g should know about—Automatic Database Diagnostic Monitor (ADDM), Automatic Workload Repository (AWR) reports, and Active Session History (ASH) reports—are also extremely useful for detecting performance issues related to the gc buffer busy wait event. The next sections explain exactly how to locate crucial information about the statements, user sessions, and database objects that are causing an Oracle database to perform poorly because of high occurrences of the gc buffer busy wait event.

Using ADDM to Find Event Information

ADDM is one of the fastest methods that an Oracle DBA can leverage to find specific recommendations about a database application workload that has been executed over a specific period of time, including which SQL statements are encountering a performance bottleneck. ADDM can draw on information retained within the AWR; it can also be executed in real time, in which case the most recent set of ASH data will be used for its analysis.

[Listing 3.1](#) shows an excerpt from an ADDM report that was run against the database during the time that excessive gc buffer busy wait events were encountered.

Listing 3.1 ADDM Report Showing the SQL Text

[Click here to view code image](#)

```
...
RECOMMENDATION 1: Schema, 84.4% benefit (17609 seconds)
    ACTION: Consider partitioning the INDEX
    "MID_B2W_ADMIN.STM_LOG_DATA_IDX"
        with object id 131712 in a manner that will evenly
        distribute
            concurrent DML across multiple partitions.
    RELEVANT OBJECT: database object with id 131712
    RATIONALE: The INSERT statement with SQL_ID "fv4un8f4w6zg8"
was
    significantly affected by "buffer busy" waits.
    RELEVANT OBJECT: SQL statement with SQL_ID fv4un8f4w6zg8
        insert into STM_LOG (NM_LOGIN, DS_ROLES, DS_OPERATION,
        DT_CRIACAO,
        CD_MARCA, CD_LOG) values (:1, :2, :3, :4, :5, :6)
...

```

Notice that ADDM was intelligent enough to isolate the specific performance issue to a particular SQL statement and connect that statement to the gc buffer busy wait event. In fact, ADDM even offered a suggestion to partition the affected database object to potentially alleviate the issue. However, we can confirm this suggestion through additional means—AWR and ASH reports—as the next sections demonstrate.

Using AWR to Find Event Information

Creating an AWR report is extremely simple; it can be done by issuing just one command from within SQL*Plus and then responding to the prompts that define which time period(s) the AWR report should span:

```
$> sqlplus / as sysdba
SQL> ?/rdbms/admin/awrrpt
...
```

Typical AWR report output usually contains an incredible amount of information about

an Oracle database's application workload behavior. When a database instance is suffering from a gc buffer busy wait event during the time period chosen for the AWR report, however, that event will usually surface as one of the Top 5 Timed Events, as shown in [Figure 3.3](#).

Top 5 Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
CPU time		13,285		53.4	
db file sequential read	2,295,269	3,525	2	14.2	User I/O
gc buffer busy	53,874	2,306	43	9.3	Cluster
direct path read	7,956,080	1,112	0	4.5	User I/O
gc current block 2-way	482,556	820	2	3.3	Cluster

Figure 3.3 Top 5 Timed Events in an AWR report

As [Figure 3.3](#) shows, the gc buffer busy event is the third-most frequently occurring wait event; this situation obviously is not optimal because it means the database instance is having to wait excessively for cache fusion to be handled properly. The excessive wait time may indicate that the root cause of this wait event may be a serious issue with the performance of the private interconnect network itself.

Another helpful source of information is the Segments by Global Cache Buffer Busy report, shown in [Figure 3.4](#).

Segments by Global Cache Buffer Busy

- % of Capture shows % of GC Buffer Busy for each top segment compared
- with GC Buffer Busy for all segments captured by the Snapshot

Owner	Tablespace Name	Object Name	Subobject Name	Obj. Type	GC Buffer Busy	% of Capture
MID_B2W_ADMIN	TS_MID_B2W_ADMIN_INDEX_M_E32M	STM_LOG_DATA_IDX		INDEX	16,933	36.65
MID_B2W_ADMIN	TS_MID_B2W_ADMIN_DATA_M	PRC_ITEM		TABLE	14,267	30.88
MID_B2W_ADMIN	TS_MID_B2W_ADMIN_DATA_M_E2M	BL_RECUSA_PAGAMENTO		TABLE	3,769	8.16
MID_B2W_ADMIN	TS_MID_B2W_ADMIN_INDEX_M_E32M	SYS_C009545		INDEX	3,499	7.57
MID_B2W_ADMIN	TS_MID_B2W_ADMIN_INDEX_M	STM_STATUS_ATUALIZACAO_IDX		INDEX	2,756	5.96

Figure 3.4 Segments waiting for gc buffer busy in an AWR report

In this example, two segments—STM_LOG_DATA_IDX and PRC_ITEM—are experiencing the largest amount of gc buffer busy waits. These segments should therefore be investigated to determine why they are experiencing almost 77 percent of all waits in this category.

[Figure 3.5](#) shows the Global Cache and Enqueue Services – Workload Characteristics report. It indicates that this database instance is experiencing significant interconnect problems because, on average, it is taking almost 0.5 seconds to receive a single buffer across the private interconnect.

Global Cache and Enqueue Services - Workload Characteristics

Avg global enqueue get time (ms):	120.5
Avg global cache cr block receive time (ms):	457.4

Figure 3.5 Problems in global cache receive time

This event is calculated using the following formula:

[Click here to view code image](#)

```
gc cr block receive time=
Time to send message to a remote LMS process by FG
+ Time taken by LMS to build block (statistics: gc cr block build
time)
+ LMS wait for LGWR latency (statistics:gc cr block flush time)
+ LMS send time (Statistics: gc cr block send time)
+ Wire latency
```

The AWR report sections therefore offer concrete evidence that something is seriously wrong with this database's environment. We will present some queries later in this chapter to show how to locate the SQL_ID, statement, block, and other information about the database server process that is causing this problem. It's important to remember that the root cause of this problem must be verified first at the hardware level—a malfunctioning network interface card (NIC), a failing network switch, misconfigured networking parameters, among many other possibilities—before pointing a finger at any other layers of the application and system as potential root causes of the problem.

Using ASH to Find Event Information

Generating an ASH report can help you locate the specific SQL statements that are experiencing performance problems related to the gc buffer busy wait event. For example, the Top User Events report section shown in [Listing 3.2](#) from the generated ASH report illustrates that wait event is definitely an issue within the selected 15-minute reporting period.

Listing 3.2 Top User Events with GC Buffer Busy

[Click here to view code image](#)

Top User EventsDB/Inst: BWMDPR/BWMDPR1 (Feb 25 17:40 to 17:55)			
Activity	Sessions	Event Class	Avg %
CPU + Wait for			
CPU	CPU		51.49
db file sequential read		User	2.49
I/O	13.17		0.64
gc buffer			

busy		Cluster	5.63	0.27
direct path read		User		
I/O	3.61	0.17		
db file scattered read		User		
I/O	3.17	0.15		
<hr/>				
<hr/>				

Another part of the same ASH report also proves that the database is encountering gc buffer busy waits. [Listing 3.3](#) shows the Top Blocking Sessions section that identifies which sessions are blocking other sessions and the event that these sessions were most commonly waiting for.

Listing 3.3 Top Blocking Sessions

[Click here to view code image](#)

Top Blocking Sessions		DB/Inst: BWMDPR/BWMDPR1 (Feb 25
17:40 to 17:55)		
-> Blocking session activity percentages are calculated with respect to		
waits on enqueue, latches and "buffer busy" only		
-> '% Activity' represents the load on the database caused by a particular blocking session		
-> '# Samples Active' shows the number of ASH samples in which the blocking session was found active.		
-> 'XIDs' shows the number of distinct transaction IDs sampled in ASH		
when the blocking session was found active.		
 Blocking Sid % Activity Event Caused %		
Event		
<hr/>		
-		
User	Program	# Samples
Active	XIDs	
<hr/>		
<hr/>		
5074, 38287	1.15 gc buffer	
busy	0.80	
MID102_B2W_WL_APP		165/901 [
18%]	0	
<hr/>		
4375, 33093	1.13 read by other	
session	0.85	

This report identifies that session ID 5074 with serial 38287 is one of the sessions that is waiting for the gc buffer busy wait event and is also blocking or making other sessions

wait for that task to finish so that the resource can become available again to another session.

Finally, [Listing 3.4](#) displays which object is responsible for generating the gc buffer busy wait event.

Listing 3.4 TOP DB Objects

[Click here to view code image](#)

Top DB Objects 17:40 to 17:55)	DB/Inst: BWMDPR/BWMDPR1 (Feb 25
-> With respect to Application, Cluster, User I/O and buffer busy waits only.	
Object ID % Activity Event	%
Event	
-----	-----
-	
Object Name (Type)	Tablespace
-----	-----
-----	-----
131712 4.37 gc buffer	
busy 2.62	
MID_B2W_ADMIN.STM_LOG_DATA_IDX	
(INDEX) TS_MID_B2W_ADMIN_INDEX_M_	
gc current block	
busy 1.36	

While ADDM, AWR, and ASH reports are valuable for both historical and real-time analysis, it's also possible to isolate this information in real time when gc buffer busy wait events are occurring without leveraging these tools, as the final sections of this chapter demonstrate.

Isolating GC Buffer Busy Wait Event Issues

Almost every Oracle DBA has experienced a situation when having just the right query at her fingertips at just the right time has saved her IT organization from utter disaster. The next sections present several queries that can quickly help isolate and analyze the root causes of—as well as the objects most affected by—the gc buffer busy wait event.

Using ASH Views to Find Waiting Sessions

Various dynamic views offer a look deep within an Oracle database's wait events from the perspective of ASH. The query in [Listing 3.5](#) shows how to identify the top 10 wait events in the database for the past 1 hour using the total amount of time waited for single events. In this particular scenario, it is apparent that the top wait event is gc buffer busy.

Listing 3.5 Finding Top 10 Wait Events in Last Hour via ASH View

[Click here to view code image](#)

```
SELECT * FROM (
  SELECT
    h.event "Wait Event",
    SUM(h.wait_time + h.time_waited)/1000000 "Total Wait Time"
    FROM v$active_session_history h,
         v$event_name e
   WHERE h.sample_time < (SELECT MAX(sample_time)
                           FROM v$active_session_history)
        AND h.sample_time > (SELECT MAX(sample_time) - 1/24
                           FROM v$active_session_history)
        AND h.event_id = e.event_id
        AND e.wait_class <>'IDLE'
  GROUP BY h.event
  ORDER BY 2 DESC)
 WHERE ROWNUM <10;
```

Wait Event	Total Wait Time
gc buffer busy	40.23931
enq: TX - row lock contention	32.385347
enq: TX - index contention	28.62571
gc current block busy	25.963209
db file sequential read	14.387571
LNS wait on SENDREQ	13.18233
gc cr multi block request	12.478076
reliable message	5.038086
cr request retry	4.887495

Now that we have identified that the gc buffer busy wait event is definitely the key to this application workload's poor performance, the next step is to use this information to locate the particular statement, user session, database object, and even individual database block if necessary that is suffering from this contention. The query in [Listing 3.6](#) shows how to obtain all necessary information, including the affected object ID, object name, object type, its datafile, and—most important—the SQL_ID of the session that is suffering gc buffer busy waits.

Listing 3.6 Finding the Database Object and SQL_ID for gc Buffer Busy Waits

[Click here to view code image](#)

```
COL object_name FORMAT A30
COL program      FORMAT A30
COL event        FORMAT A30
SELECT DISTINCT
```

```

current_obj#,
o.object_name,
o.owner,
o.object_type,
current_file#,
session_state,
sql_id,
event
FROM v$active_session_history a,
      dba_objects o
WHERE a.current_obj# = o.object_id
  AND a.event LIKE '%gc buffer busy%';

```

CURR_OBJ#	OBJECT_NAME	OWNER	OBJ_TYPE	CURR_FILE#	SESSION	SQL_ID
104830	STM_DETALHE_BUFFER_BUSY	MID_B2W	INDEX	445	WAITING	39j77bgam9506
79829	STM_ITEM_BUFFER_BUSY	MID_B2W	TABLE	140	WAITING	0ba26mnwv
55681	STM_ITEM_BUFFER_BUSY	MID_B2W	TABLE	141	WAITING	9apqp7fw1
131712	STM_LOG_GROUP_BUFFER_BUSY	MID_B2W	INDEX	445	WAITING	fv4un8f4v
	DATA_IDX	_ADMIN				

It is now obvious which objects are affected by gc buffer busy waits. After running the following query to identify the matching SQL text for SQL_ID 39j77bgam9506, it is evident that the statement is actually performing an INSERT:

[Click here to view code image](#)

```
SQL> select sql_text from v$sqltext where
  sql_id='39j77bgam9506'  order by piece;
```

SQL_TEXT

insert into STM_DETALHE_LOG (CD_LOG, DS_ATRIBUTO, DS_VALOR, TP_DETALHE_LOG, CD_DETALHE_LOG) values (:1, :2, :3, :4, :5)

It's therefore likely that this operation will benefit from a change to the index structure, such as re-creating the affected index as a hash-partitioned index rather than a standard nonpartitioned index. For instance, if this index were re-created with 64 partitions, the concurrency demands placed on the index would be significant because now the INSERTs would be spread across 64 separate index partition branches rather than across just one index branch as before.

Quickly Isolating Performance Bottlenecks

This next real-world example demonstrates several queries that can be executed within SQL*Plus to quickly isolate exactly what is causing a performance bottleneck within a database, including the capture of which database user account, session, and statement is causing the problem.

1. Verify the interval that will be chosen when analyzing the database in the pursuit of the database bottleneck at that specific time. The following query checks whether the AWR snapshot timeframe between AWR snapshot IDs is the one that is really needed for the analysis about to be performed:

[Click here to view code image](#)

```
col min  for a30
col max  for a30
SQL> SELECT
      MIN(begin_interval_time) min,
      MAX(end_interval_time) max
     FROM dba_hist_snapshot
    WHERE snap_id BETWEEN 54657 AND 54658;

MIN                                MAX
-----                            -----
28-FEB-15 09.00.15.104 AM        28-FEB-15 11.00.04.693 AM
```

2. Verify that the defined AWR snapshot interval indeed does encapsulate the class of wait events that as a whole are exhibiting the longest waits for the instance as of that AWR snapshot timeframe:

[Click here to view code image](#)

```
SQL> SELECT
      wait_class_id,
      wait_class,
      COUNT(*) cnt
     FROM dba_hist_active_sess_history
    WHERE snap_id BETWEEN 54657 AND 54659
   GROUP BY wait_class_id, wait_class
  ORDER BY 3;
```

WAIT_CLASS_ID	WAIT_CLASS	CNT
4166625743	Administrative	23
3290255840	Configuration	23
3386400367	Commit	111
4217450380	Application	147
2000153315	Network	233
4108307767	System I/O	236
3875070507	Other	544
1893977003	Cluster	633
1740759767	User I/O	1019

11 rows selected.

From this query's output, it's obvious that the Concurrency wait event class is encountering the highest waits for this time period.

3. Armed with this information, use the following query to find the event inside the wait class that is waiting the longest; the gc buffer busy is the worst one:

[Click here to view code image](#)

```
SELECT
    event_id,
    event,
    COUNT(*) cnt
FROM dba_hist_active_sess_history
WHERE snap_id BETWEEN 54657 AND 54659
    AND wait_class_id = 3871361733
GROUP BY event_id, event
ORDER BY 3;
```

EVENT_ID	EVENT	CNT
-		
...		
2277737081	gc current grant	
busy		147
3046984244	gc cr block 3-	
way		194
737661873	gc cr block 2-	
way		318
111015833	gc current block 2-	
way		451
2701629120	gc current block	
busy		473
1478861578	gc buffer	
busy		1333

24 rows selected.

This query's output corroborates the fact that the gc buffer busy wait event is encountering the highest waits for this time period.

4. Run this next query to isolate the SQL_ID of statements that were run within sessions that are waiting for the gc buffer busy wait event:

[Click here to view code image](#)

```
SELECT
    sql_id,
    COUNT(*) cnt
FROM dba_hist_active_sess_history
```

```

WHERE snap_id BETWEEN 54657 AND 54659
    AND event_id = 1478861578
GROUP BY sql_id
HAVING COUNT(*)>1
ORDER BY 2;

```

SQL_ID	CNT
...	
0s34c5d0n7577	48
5bwdfzr1s4cx0	70
Gppjjfhgxnbxx	94
fv4un8f4w6zg8	690

52 rows selected.

5. Locate the SQL text that corresponds to the identified SQL_ID via this simple query:

[Click here to view code image](#)

```

SQL> SELECT
      DISTINCT sql_text
        FROM gv$sqltext
       WHERE sql_id = 'fv4un8f4w6zg8';

```

SQL_TEXT
insert into STM_LOG (NM_LOGIN, DS_ROLES, DS_OPERATION, DT_CRIACAO, CD_MARCA, CD_LOG) values (:1, :2, :3, :4, :5, :6)

Like the example obtained from running an ASH report in prior sections, this report points to a potential issue: an `INSERT` statement is executing during the same timeframe that those same indexes are being accessed from too many user sessions simultaneously. This is the most likely culprit for generating high wait counts for the `Concurrency` wait class and, more specifically, the `gc buffer busy` wait event.

Fixes for GC Buffer Busy Waits

Since every Oracle database is certainly unique, it is neither possible nor particularly desirable to implement the same repairs for the `gc buffer busy` events in all cases. However, there are several well-known avenues for potential investigation and repair of the scenarios we have presented so far:

- If the object experiencing the `gc buffer busy` waits is an index, verify the possibility of re-creating the index using a *HASH partitioning* methodology and/or consider re-creating the index as a *reverse key* index.
- If the database was just migrated from a single instance environment to a RAC environment, consider creating a new database service that will isolate the application to using only one node of the RAC database, thus isolating the application workload so that it accesses just *one* of the RAC database instances.

This approach should significantly reduce global cache waits and is probably the simplest workaround to repair this issue; however, this approach also effectively limits the possibility of load-balancing the affected application workloads between multiple nodes. If the application workload is part of Oracle's E-Business Suite (EBS), it is possible to create separate database services within each EBS module (AP, OM, IVN, and so forth).

- A smaller database block size *may* help to improve or even eliminate block contention and the gc buffer busy wait event because fewer blocks will need to be handled when locating those needed to complete a transaction successfully. Of course, using an absurdly small block size just to overcome this problem may be equally counterproductive. Remember that a 2 KB block size has already yielded approximately 10 percent of its space to its block header and significantly reduces the amount of data that can be stored within a single block for the segments that are suffering from wait events; it could also contribute to significant row chaining and/or row migration as well as slower performance for full table scans for larger tables. And don't forget that the only way to use a different block size than the database's default block size is to create a tablespace with that smaller block size and then move the appropriate objects to that tablespace—definitely not a trivial exercise.
- If Automatic Segment Space Management (ASSM) is *not* in use, be sure to check the configuration of the segment's *free lists* for recommended proper settings; better yet, be sure to slate these objects to an early migration to a tablespace that already employs ASSM as soon as possible.

Summary

This chapter illustrated several different methods to detect and repair problems related to the gc buffer busy wait event. It demonstrated how to use the main reports that Oracle Database provides automatically through Oracle tools such as ADDM, AWR, and ASH to locate the bottleneck, as well as some valuable SQL*Plus queries that can quickly help analyze a database's performance related to this wait event. Some final points to remember:

- Every Oracle database is different; even if you isolate a problem and it appears to be the exact same issue you just encountered on another database, be sure to check all dependent objects before making any change. The change recommended may cause no negative or positive impact, but tools such as Oracle Database Replay and Oracle SQL Performance Analyzer can assist to ascertain if there are any unexpected side effects, however minimal.
- In a reactive situation regarding gc buffer busy waits, it is important to capture all necessary data—ADDM, AWR, and ASH reports—and retain it for as long as possible (perhaps via snapshots in Oracle 11.2.0).
- Don't forget to proceed methodically when diagnosing and alleviating the true root cause(s) of gc buffer busy waits. While it may be tempting to seize upon a single

potential cause, there may actually be multiple root causes—for example, a malfunctioning NIC in concert with poorly written application code combined with an inappropriate block size for the application’s data.

4. Adaptive Cursor Sharing

When a new query is issued, the database server makes a syntactic check to determine the legality of the SQL statement. If this new query is found to be semantically equivalent to an existing one already hashed and currently available in the library cache, it is executed using the execution plan of the earlier query.

Such a sharing mechanism is possible when using bind variables (or literals with `CURSOR_SHARING` set to `FORCE`). However, cursor sharing and SQL optimization might be diametrically opposed. Whereas bind variables avoid reoptimization by sharing the existing child cursor, they are not necessarily going to do the same amount of work and henceforth they might create a performance issue.

Oracle Database 11g introduces adaptive cursor sharing (ACS; also known as extended cursor sharing) to address this conflicting issue between sharing resources and optimizing SQL. Multiple optimal execution plans per SQL statement can be generated depending on the value of the peeked bind variable and certain other criteria presented in this chapter. The chapter first examines the prerequisites for ACS, its working mechanism, and dynamic views to monitor it. Next, it outlines the secret sauce used to mark a cursor bind aware. Finally, it shows a practical case taken from a running system where a bind-aware cursor becomes dramatically nonperformant, generating a high number of child cursors and thereby damaging considerably the library cache of the application database.

ACS Working Algorithm

[Figure 4.1](#) depicts an algorithm for triggering ACS. When a SQL query using bind variables is launched, if its underlying parent cursor is bind sensitive (this cursor property is explained shortly), the query is monitored by Oracle via the ACS feature so that an optimal execution plan is generated and used to execute the query. This cursor is then said to be adaptive because it fulfills two conditions: it is bind sensitive and it is bind aware.

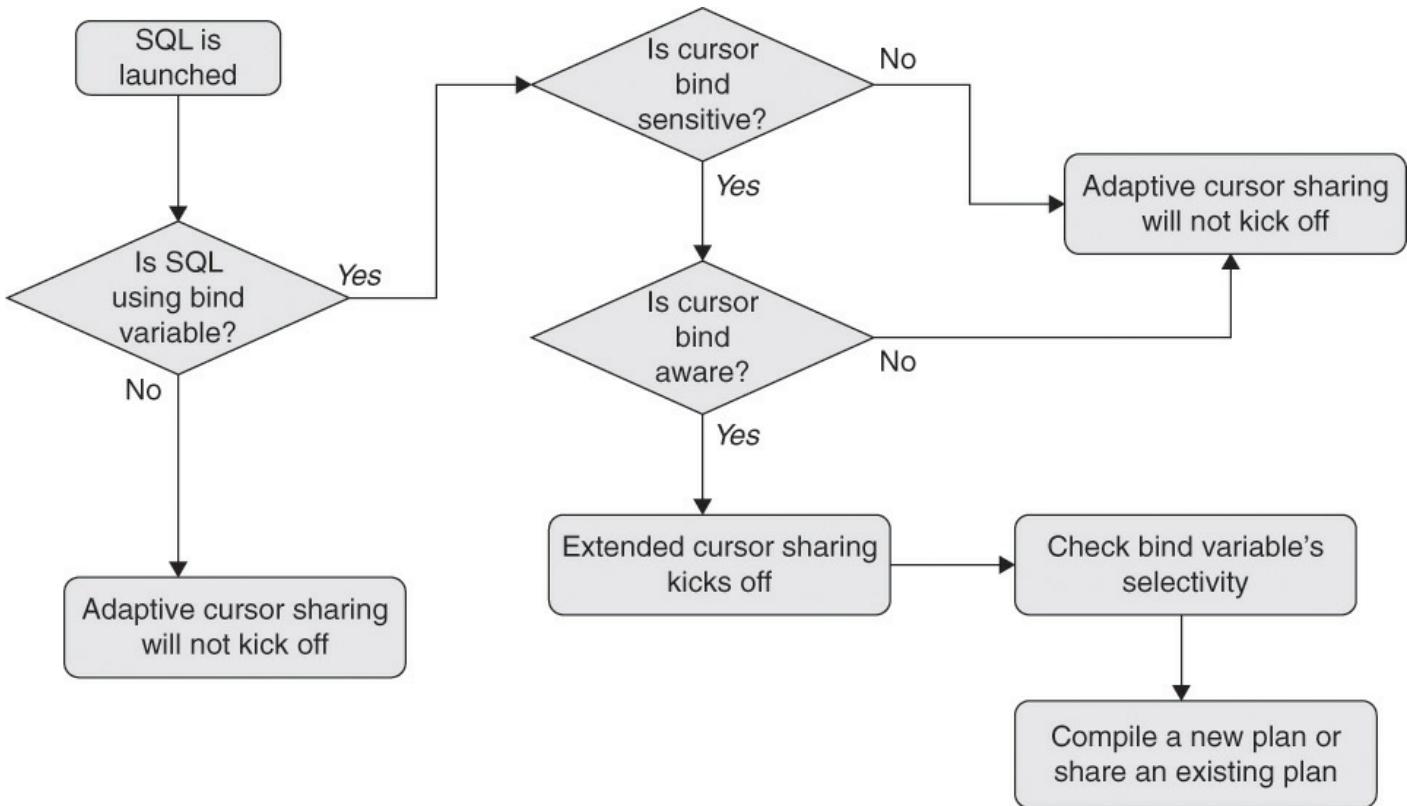


Figure 4.1 Simple ACS triggering algorithm

The next section presents the bind-sensitive property and its prerequisites. Then we show how a cursor transits from this property to a bind-aware status in which its underlying SQL query will start to be executed via an optimal execution plan depending on its bind variable value's selectivity.

Bind Sensitiveness with Range Predicate

The model used in this chapter is based primarily on a simple two-column heap table that has a B-tree single-column index. Basic statistics for the table, without histograms, have also been gathered, as shown in the following:

[Click here to view code image](#)

```

SQL> select * from v$version;

BANNER
-----
-----
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit
Production
PL/SQL Release 12.1.0.1.0 - Production
CORE    12.1.0.1.0      Production
TNS for 64-bit Windows: Version 12.1.0.1.0 - Production
NLSRTL Version 12.1.0.1.0 - Production

SQL> create table t_acs(n1 number, n2 number);

SQL> BEGIN
      for j in 1..1200150 loop
  
```

```

if j = 1 then
    insert into t_acs values (j, 1);
elsif j>1 and j<=101 then
    insert into t_acs values(j, 100);
elsif j>101 and j<=1101 then
    insert into t_acs values (j, 1000);
elsif j>10001 and j<= 110001 then
    insert into t_acs values(j,10000);
else
    insert into t_acs values(j, 1000000);
end if;
end loop;
commit;
END;
/
SQL> create index t_acs_il on t_acs(n2);

SQL> BEGIN
    dbms_stats.gather_table_stats
        (user
         , 't_acs'
         , method_opt => 'for all columns size 1'
         , cascade => true
         , estimate_percent => dbms_stats.auto_sample_size
        );
END;
/

```

The data in column N2 is highly skewed, as shown in the following:

[Click here to view code image](#)

```

SQL> select n2, count(1) from t_acs group by n2 order by 2;

      N2      COUNT(1)
----- -----
          1            1
        100           100
       1000          1000
      10000         100000
    1000000        1099049

```

We explore the effect of skewness of data in the N2 column on the bind sensitiveness of its underlying cursor later in this chapter.

When a cursor uses a bind variable in a range predicate of a `where` clause, it is marked bind sensitive as follows:

[Click here to view code image](#)

```

SQL> var ln2 number;
SQL> exec :ln2 := 100;

```

```

SQL> select count(1) from t_acs where n2 <= :ln2;
      COUNT(1)
-----
      101

SQL> select * from table(dbms_xplan.display_cursor);

-----
| Id  | Operation          | Name   | Rows  | Bytes |
-----| 0  | SELECT STATEMENT  |         |        |        |
| 1  |  SORT AGGREGATE   |         |       1 |       3 |
|* 2 |  TABLE ACCESS FULL| T_ACS  | 397K | 1165K|
-----
```

Predicate Information (identified by operation id):

```
2 - filter("N2" <=:LN2)
```

```

SQL> select
      sql_id
    , child_number
    , is_bind_sensitive
  from
    v$sql
 where
  sql_id = 'ct0yv82p15jdw';
```

SQL_ID	CHILD_NUMBER	IS_BIND_SENSITIVE
ct0yv82p15jdw	0	Y

As you can see, the corresponding child cursor is bind sensitive.

Thus, although column N2, when used in a range predicate, doesn't need a histogram to be bind sensitive, it nevertheless requires having simple statistics gathered on it. To demonstrate, the following simply flushes the shared pool, deletes the statistics, and requeries using the same value of the bind variable:

[Click here to view code image](#)

```

SQL> alter system flush shared_pool;

SQL> exec dbms_stats.delete_table_stats(user,'t_acs');

SQL> select count(1) from t_acs where n2 <= :ln2;

SQL> select * from table(dbms_xplan.display_cursor);

SQL_ID  ct0yv82p15jdw, child number 0
-----
```

```

| Id | Operation          | Name      | Rows | Bytes |
-----+
|   0 | SELECT STATEMENT   |           |       |        |
|   1 |   SORT AGGREGATE   |           |     1 |    13 |
| * 2 |   INDEX RANGE SCAN | T_ACS_I1 |  101 | 1313 |
-----+

```

Predicate Information (identified by operation id):

```
2 - access ("N2"<=:LN2)
```

Note

```
- dynamic sampling used for this statement (level=2)
```

```
SQL> select
      sql_id
    ,child_number
    ,is_bind_sensitive
  from
    v$sql
 where
   sql_id = 'ct0yv82p15jdw';
```

SQL_ID	CHILD_NUMBER	IS_BIND_SENSITIVE
ct0yv82p15jdw	0	N

Notice that the child cursor 0 has not been marked bind sensitive in this case because column N2 does not have any statistics at all. The note about dynamic sampling should normally give you a clue as to why your cursor is not bind sensitive when you think it should be.

Bind Sensitiveness with Equality Predicate and Histogram

A cursor can also be marked bind sensitive when the column appears in an equality predicate and has a histogram, as shown in the following:

[Click here to view code image](#)

```
SQL> BEGIN
      dbms_stats.gather_table_stats
        (user
         , 't_acs'
         , method_opt => 'for all columns size auto'
         , cascade => true
         , estimate_percent => dbms_stats.auto_sample_size
         );
END;
/
SQL> SELECT
```

```

        column_name,
        histogram
FROM user_tab_col_statistics
WHERE table_name = 'T_ACS'
AND column_name = 'N2';

```

COLUMN_NAME	HISTOGRAM
N2	FREQUENCY

```
SQL> select count(1) from t_acs where n2 = :ln2;
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

```
SQL_ID  f2pmwazy1rnfd, child number 0
```

Id	Operation	Name	Rows	Bytes
0	SELECT STATEMENT			
1	SORT AGGREGATE		1	3
* 2	INDEX RANGE SCAN T_ACS_I1	1372	4116	

```
Predicate Information (identified by operation id):
```

```
2 - access("N2"=:LN2)
```

```
SQL> select
      sql_id
    ,child_number
    ,is_bind_sensitive
  from
    v$sql
 where
   sql_id = 'f2pmwazy1rnfd';
```

SQL_ID	CHILD_NUMBER	IS_BIND_SENSITIVE
f2pmwazy1rnfd	0	Y

Having gathered a histogram on column N2, a new query was executed using an equality predicate, and a bind-sensitive property of the corresponding child cursor was set to Y (yes).

Bind Sensitiveness with Partition Keys

Finally, a cursor can be marked bind sensitive when a SQL query uses a partition key in its predicate part. To demonstrate, the following creates a range partitioned table,

populates it with appropriate data, and collects statistics without a histogram:

[Click here to view code image](#)

```
SQL> create table t_acs_part
  (n1 number, n2 number)
partition by range (n2)
(partition p1 values less than (100)
,partition p2 values less than (1000)
,partition p3 values less than (10000)
,partition p4 values less than (100000)
,partition p5 values less than (1000000)
,partition p6 values less than (10000000)
);

SQL> BEGIN
  for j in 1..1200150 loop
    if j = 1 then
      insert into t_acs_part values (j, 1);
    elsif j>1 and j<=101 then
      insert into t_acs_part values(j, 100);
    elsif j>101 and j<=1101 then
      insert into t_acs_part values (j, 1000);
    elsif j>10001 and j<= 110001 then
      insert into t_acs_part values(j,10000);
    else
      insert into t_acs_part values(j, 1000000);
    end if;
  end loop;
  commit;
END;
/

SQL> BEGIN
  dbms_stats.gather_table_stats
  (user
   , 't_acs_part'
   , method_opt => 'for all columns size 1'
   , cascade => true
   , estimate_percent => dbms_stats.auto_sample_size
   );
END;
/

SQL> SELECT
  column_name,
  histogram
FROM user_tab_col_statistics
WHERE table_name = 'T_ACS_PART'
AND column_name = 'N2';
```

```
COLUMN_NAME HISTOGRAM
-----
N2          NONE
```

A query against this table using the partition key (N2) in the where clause causes the cursor to be marked bind sensitive:

[Click here to view code image](#)

```
SQL> select count(1) from t_acs_part where n2 = :ln2;
```

```
COUNT(1)
-----
100
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

```
SQL_ID byztzuffb65n9, child number 0
-----
---  

| Id  | Operation           | Name      | Rows | Pstart |  

Pstop |
-----  

---  

|   0 | SELECT              |          |      |        |  

STATEMENT          |          |          |      |        |  

|   1 | SORT AGGREGATE     |          |      |        | 1  

|       |                   |          |      |        |  

|   2 | PARTITION RANGE SINGLE |          | 100 |        | KEY  

| KEY  |                   |          |      |        |  

|*  3 | TABLE ACCESS FULL    | T_ACS_PART | 100 |        | KEY  

| KEY  |                   |          |      |        |  

-----  

---
```

Predicate Information (identified by operation id):

```
3 - filter("N2"=:LN2)
```

```
SQL> select
      sql_id
    ,child_number
    ,is_bind_sensitive
  from
    v$sql
  where
    sql_id = 'byztzuffb65n9';
```

```
SQL_ID      CHILD_NUMBER IS_BIND_SENSITIVE
-----
byztzuffb65n9          0 Y
```

In this case, too, the cursor has been marked bind sensitive.

ACS in Action

Now that we know about various prerequisites a SQL cursor should fulfill in order to be marked bind sensitive, let's look at how Oracle generates multiple optimal execution plans for the same cursor using different bind variable values.

Let's first see again how the data in column N2 is distributed in the nonpartitioned table t_acs:

[Click here to view code image](#)

```
SQL> select n2, count(1) from t_acs group by n2 order by 2;
```

N2	COUNT (1)
1	1
100	100
1000	1000
10000	100000
1000000	1099049

The following demonstration has been carried out on the nonpartitioned table t_acs with skewed data distribution in column N2, as shown earlier. Since this column will be used in a query having an equality predicate, histograms have been gathered on column N2. This query uses two bind variable values in the equality predicate: 100 and 1000000. The value 100 favors an index range scan path, while the value 1000000 favors a full table scan. Let's start the experiment:

[Click here to view code image](#)

```
SQL> alter system flush shared_pool;
SQL> exec :ln2 := 100
SQL> select count(1) from t_acs where n2 = :ln2;
COUNT(1)
-----
100
SQL> select * from table(dbms_xplan.display_cursor);
SQL_ID  f2pmwazy1rnfd, child number 0
-----
| Id  | Operation          | Name      | Rows  | Bytes |
|     | SELECT STATEMENT   |           |        |        |
|     |  SORT AGGREGATE    |           |       1 |        3 |
|*  2 |  INDEX RANGE SCAN | T_ACS_I1 | 1372  | 4116  |
-----
Predicate Information (identified by operation id):

```

```
-----  
2 - access ("N2"=:LN2)
```

As expected, the optimizer has generated a plan employing an index range scan for the bind variable value of 100.

Let's now change the bind variable value to 1000000 for which we expect to have a full table scan plan:

[Click here to view code image](#)

```
SQL> exec :ln2 := 1000000
```

```
SQL> select count(1) from t_acs where n2 = :ln2;
```

```
COUNT(1)  
-----
```

```
1099049
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

```
SQL_ID  f2pmwazy1rnfd, child number 0  
-----
```

Id	Operation	Name	Rows	Bytes
0	SELECT STATEMENT			
1	SORT AGGREGATE		1	3
* 2	INDEX RANGE SCAN T_ACS_I1	1372	4116	

```
-----
```

```
Predicate Information (identified by operation id):  
-----
```

```
2 - access ("N2"=:LN2)
```

Despite the numerous additional rows for the new value of the bind variable, we still use the same index range scan execution plan as we used earlier. Let's check first whether this cursor is bind sensitive:

[Click here to view code image](#)

```
SQL> select  
      sql_id  
    ,child_number  
  ,is_bind_sensitive  
  ,is_bind_aware  
  ,is_shareable  
  from  
    v$sql  
 where  
   sql_id = 'f2pmwazy1rnfd';
```


SQL_ID	CHILD_NUMBER	IS_BIND_SENSITIVE
IS_BIND_AWARE		IS_SHAREABLE

```
-----  
-----  
f2pmwazy1rnfd
```

```
0 Y
```

```
N
```

```
Y
```

As you can see, the cursor is bind sensitive but not bind aware yet. Let's execute the same query again with the same bind variable value (1000000):

[Click here to view code image](#)

```
SQL> select count(1) from t_acs where n2 = :ln2;
```

```
COUNT (1)
```

```
-----  
1099049
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

```
SQL_ID f2pmwazy1rnfd, child number 1
```

Id	Operation	Name	Rows	Bytes
0	SELECT STATEMENT			
1	SORT AGGREGATE		1	3
*	TABLE ACCESS FULL	T_ACS	1096K	3212K

```
Predicate Information (identified by operation id):
```

```
-----  
2 - filter("N2"=:LN2)
```

Finally, after two executions, Oracle Optimizer has generated a new execution plan that better suits the current bind variable value. If we look at the child cursor properties, we will find that a new child cursor 1 has been generated and is marked bind aware. Also, the existing child cursor 0 has been assigned a “not shareable” status signaling that it can be aged out away from the library cache under space pressure, as shown in the following:

[Click here to view code image](#)

```
SQL> select  
      sql_id  
      ,child_number  
      ,is_bind_sensitive  
      ,is_bind_aware  
      ,is_shareable  
  from  
    v$sql  
 where  
   sql_id = 'f2pmwazy1rnfd';
```

```
SQL_ID      CHILD_NUMBER IS_BIND_SENSITIVE  
IS_BIND_AWARE  IS_SHAREABLE
```

```

-----
-----
```

f2pmwazy1rnfd	0	Y	N	N
f2pmwazy1rnfd	1	Y	Y	Y

Now that the cursor has been marked bind aware, if we assign the value of 1000 to the bind variable, which normally favors an index range scan, would Oracle produce the expected execution plan? Let's see:

[Click here to view code image](#)

```

SQL> exec :ln2 := 1000

SQL> select count(1) from t_acs where n2 = :ln2;

COUNT(1)
-----
1000

SQL> select * from table(dbms_xplan.display_cursor);

SQL_ID  f2pmwazy1rnfd, child number 2
-----
| Id  | Operation          | Name      | Rows  | Bytes |
|-----|
|   0 | SELECT STATEMENT   |           |        |        |
|   1 |   SORT AGGREGATE   |           |       1 |       3 |
|*  2 |   INDEX RANGE SCAN | T_ACS_I1 | 2747 | 8241 |
|-----|
```

Predicate Information (identified by operation id):

```
2 - access("N2"=:LN2)
```

```

SQL> select
      sql_id
    ,child_number
    ,is_bind_sensitive
    ,is_bind_aware
    ,is_shareable
  from
    v$sql
 where
    sql_id = 'f2pmwazy1rnfd';
```

SQL_ID	CHILD_NUMBER	IS_BIND_SENSITIVE		
IS_BIND_AWARE		IS_SHAREABLE		

```

-----
-----
```

f2pmwazy1rnfd	0	Y	N	N
---------------	---	---	---	----------

f2pmwazy1rnfd	1 Y	Y	Y
f2pmwazy1rnfd	2 Y	Y	Y

Indeed, Oracle has compiled a new bind-aware execution plan (cursor child number 2) based on an optimal index range scan access path.

The preceding demonstrates, in a nutshell, how the ACS feature works. When a cursor is bind sensitive for any of the reasons explained earlier (range predicate, equality predicate with histogram and partition key), all subsequent cursor executions will be monitored so that, after a warm-up period, the cursor becomes bind aware and an optimal plan is generated (or shared depending on the bind variable selectivity) for each bind variable value.

However, how many executions does the warm-up period need during which the cursor has to execute suboptimally before an optimal plan will be generated? This question is answered in the next section, which shows how these executions are monitored and what Oracle uses to mark a cursor bind aware.

ACS Bind-Awareness Monitoring

The `/*+ bind_aware */` undocumented hint has the effect of making the cursor immediately bind aware, as shown via the following simple example:

[Click here to view code image](#)

```

SQL> create table t1
  as select
        rownum n1
     ,trunc((rownum -1)/3) n2
   from dual
 connect by level <=1e3;

SQL> exec dbms_stats.gather_table_stats(user, 't1');

SQL> select /*+ bind_aware */
      count(*)
    from t1
   where n2 = :ln2;

          COUNT(*)
-----
          0
SQL> select * from table(dbms_xplan.display_cursor);

SQL_ID  5gz8nu7ru5gh0, child number 0
-----
| Id  | Operation          | Name | Rows  | Bytes |
-----+
|   0 | SELECT STATEMENT   |       |        |        |
|   1 |  SORT AGGREGATE    |       |     1  |      4  |
| * 2 | TABLE ACCESS FULL| T1   |     1  |      4  |

```

```

-----
Predicate Information (identified by operation id):
-----
2 - filter("N2"=:LN2)

SQL> SQL> select
      sql_id
      ,child_number
      ,is_bind_sensitive
      ,is_bind_aware
      ,is_shareable
   from
     v$sql
  where
    sql_id = '5gz8nu7ru5gh0';

SQL_ID          CHILD_NUMBER IS_BIND_SENSITIVE
IS_BIND_AWARE      IS_SHAREABLE
-----
-----
5gz8nu7ru5gh0           0  Y                   Y
                                         Y

```

But, if you don't use this hint, you may find that your initial cursor needs a certain number of executions before reaching the bind-awareness status. How many such initial executions does a cursor need in its warming-up period? And how are these executions monitored? Those are the two questions answered in this section.

Let's continue using the `t_acs` table that was engineered with a special data pattern to make the monitoring explanation obvious. As shown earlier and reproduced here, the data distribution in column `N2` is highly skewed:

[Click here to view code image](#)

```

SQL> select n2, count(1) from t_acs group by n2 order by 2;

      N2      COUNT(1)
-----
        1          1
      100         100
     1000        1000
    10000       100000
  1000000      1099049

```

BUCKET_ID and COUNT Relationship

Oracle provides three views that are used to monitor a cursor that is a candidate for an ACS feature:

[Click here to view code image](#)

```
SQL> desc v$sql_cs_statistics
```

	Name	Null?	Type
1	ADDRESS		RAW
2	HASH_VALUE		NUMI
3	SQL_ID		VARCHAR
4	CHILD_NUMBER		NUMI
5	BIND_SET_HASH_VALUE		NUMI
6	PEEKED		VARCHAR
7	EXECUTIONS		NUMI
8	ROWS_PROCESSED		NUMI
9	BUFFER_GETS		NUMI
10	CPU_TIME		NUMI
11	CON_ID		NUMI

The V\$SQL_CS_STATISTICS view lists the statistics of the number of processed rows by the corresponding child cursor, the number of executions it has undergone so far, and the number of consumed buffers and CPU time. Although this view was reporting something in Oracle 11g, it seems it has not been used since Oracle 12c, since no rows were discovered when investigating the bind awareness in Oracle 12c. Even in Oracle 11g, the number of rows processed is not updated before a cursor is bind aware.

The second ACS monitoring view is V\$SQL_CS_HISTOGRAM:

[Click here to view code image](#)

	Name	Null?	Type
1	ADDRESS		RAW
2	HASH_VALUE		NUMI
3	SQL_ID		VARCHAR
4	CHILD_NUMBER		NUMI
5	BUCKET_ID		NUMI
6	COUNT		NUMI
7	CON_ID		NUMI

The V\$SQL_CS_HISTOGRAM view stores the number of cursor executions per bucket and is used to decide when it is time to mark a cursor bind aware, as we will detail later in this section.

The third and last ACS monitoring view is V\$SQL_CS_SELECTIVITY:

[Click here to view code image](#)

	Name	Null?	Type
1	ADDRESS		RAW
2	HASH_VALUE		NUMI
3	SQL_ID		VARCHAR

4	CHILD_NUMBER	NUMI
5	PREDICATE	VARC
6	RANGE_ID	NUMI
7	LOW	VARC
8	HIGH	VARC
9	CON_ID	NUMI

The V\$SQL_CS_SELECTIVITY view begins to be useful only when a cursor has been marked bind aware. It contains information about the selectivity of the bind variable's values, including a low and high value range per child cursor.

The goal of this section is to define the relationship that exists between the number of rows processed by a child cursor and the couple (BUCKET_ID, COUNT) of the V\$SQL_CS_HISTOGRAM view. The most important columns are highlighted in bold. Let's first flush the shared pool and issue the same query using three different bind variable values: 100, 10000, and 1000000. For each execution, let's check the content of the V\$SQL_CS_HISTOGRAM monitoring view:

[Click here to view code image](#)

```
SQL> alter system flush shared_pool;
SQL> exec :ln2 := 100;
SQL> select count(1) from t_acs where n2 = :ln2;
COUNT(1)
-----
100
```

If we check the V\$SQL_CS_STATISTICS view, it does not report any records in Oracle 12.1.0.1.0, although it did report something in Oracle 11g, as shown in the following:

[Click here to view code image](#)

```
SQL> select
      child_number
    , executions
    , rows_processed
  from v$sql_cs_statistics
 where sql_id = 'f2pmwazy1rnfd' ;
no rows selected
```

It seems that this view has been kept only for backward compatibility, which is why it is not discussed further. Even in Oracle 11g, this view did not report a correct value in the ROWS_PROCESSED column prior to the corresponding cursor becoming bind aware.

The V\$SQL_CS_HISTOGRAM view, however, is showing interesting information. The child cursor 0, used to honor the query, has three BUCKET_IDS: 0, 1, and 2. And it can be seen that for the child cursor 0, which has processed 100 rows, the COUNT of

BUCKET_ID 0 has been incremented to 1:

[Click here to view code image](#)

```
SQL> select
      child_number
      ,bucket_id
      ,count
  from
    v$sql_cs_histogram
 where  sql_id = 'f2pmwazylrnfd' ;
```

CHILD_NUMBER	BUCKET_ID	COUNT
0	0	1 → incremented because processed rows <1000
0	1	0
0	2	0

If we execute the same query twice more, the COUNT of the BUCKET_ID 0 increments to 3, as follows:

[Click here to view code image](#)

```
SQL> select count(1) from t_acs where n2 = :ln2;
SQL> select count(1) from t_acs where n2 = :ln2;
SQL> select
      child_number
      ,bucket_id
      ,count
  from
    v$sql_cs_histogram
 where  sql_id = 'f2pmwazylrnfd' ;
```

CHILD_NUMBER	BUCKET_ID	COUNT
0	0	3 → incremented because processed rows <1000
0	1	0
0	2	0

Now, we will change the bind variable value to 10000 with 100,000 rows, run the same query again, and check the contents of V\$SQL_CS_HISTOGRAM. This time, the COUNT of BUCKET_ID 1 is incremented to 1:

[Click here to view code image](#)

```
SQL> exec :ln2 := 10000;
SQL> select count(1) from t_acs where n2 = :ln2;
   COUNT(1)
-----
  100000
```

```

SQL> select
      child_number
    , bucket_id
    , count
  from
    v$sql_cs_histogram
  where sql_id = 'f2pmwazylrnfd' ;

```

CHILD_NUMBER	BUCKET_ID	COUNT
0	0	3
0	1	1 → incremented because 1000<=processed rows <1e6
0	2	0

Finally, if we execute the same query using the bind variable value 1000000, it returns 1099049 rows. If we check V\$SQL_CS_HISTOGRAM, we will see that Oracle increments the COUNT of BUCKET_ID 2:

[Click here to view code image](#)

```

SQL> exec :ln2 := 1000000;

SQL> select count(1) from t_acs where n2 = :ln2;

COUNT(1)
-----
1099049

SQL> select
      child_number
    , bucket_id
    , count
  from
    v$sql_cs_histogram
  where sql_id = 'f2pmwazylrnfd' ;


```

CHILD_NUMBER	BUCKET_ID	COUNT
0	0	3
0	1	1
0	2	1 → incremented because processed rows >=1e6

The preceding experiments show us clearly that Oracle is using the following heuristic methods to increment the COUNT of the three BUCKET_IDS belonging to any compiled child cursor:

- If the number of processed rows is between 0 and 1000, then increment the COUNT of BUCKET_ID 0.

- If the number of processed rows is between 1000 and 1 million, then increment the COUNT of BUCKET_ID 1.
- If the number of processed rows is greater than 1 million, then increment the COUNT of BUCKET_ID 2.

Marking Cursors Bind Aware

The relationship previously discussed (BUCKET_ID, COUNT) has paved the way to uncovering the secret sauce Oracle uses to mark a cursor bind aware. Let's continue using the pair (BUCKET_ID, COUNT) to figure out that secret sauce. The following sections look at three test cases:

- Case 1 looks at two adjacent buckets that have a COUNT greater than 0, while the COUNT of the remaining bucket is equal to 0.
- Case 2 involves two nonadjacent BUCKET_IDS (BUCKET_ID 0 and 2) that have a COUNT greater than 0, while the COUNT of BUCKET_ID 1 is equal to 0.
- Case 3 provides a scenario in which all BUCKET_IDS have a COUNT greater than 0.

Case 1: Adjacent Buckets (0 and 1 or 1 and 2) with a COUNT Greater Than Zero

This is the simplest case. We execute the initial query five times using the first bind variable value 100 and then five more times using the second bind variable value 10000. The goal is to observe when the cursor is marked bind aware.

[Click here to view code image](#)

```
SQL> alter system flush shared_pool;
SQL> exec :ln2 := 100
SQL> select count(1) from t_acs where n2 = :ln2;
COUNT(1)
-----
100

SQL> -- repeat 4 times
SQL> select
      child_number
     ,bucket_id
     ,count
   from
     v$sql_cs_histogram
  where  sql_id = 'f2pmwazy1rnfd' ;
CHILD_NUMBER    BUCKET_ID        COUNT
-----          -----        -----
0                  0            5 → incremented 5 times
```

```

0          1          0
0          2          0

SQL> exec :ln2 := 10000

SQL> select count(1) from t_acs where n2 = :ln2;

COUNT(1)
-----
100000

SQL> -- repeat 4 times

SQL> select
      child_number
     ,bucket_id
     ,count
   from
     v$sql_cs_histogram
  where  sql_id = 'f2pmwazylrnfd' ;

CHILD_NUMBER  BUCKET_ID      COUNT
-----  -----  -----
0            0            5
0            1            5 → incremented 5 times
0            2            0

```

After five executions using a different bind variable value, which processes a high number of records and causes the COUNT of BUCKET_ID 1 to rise to 5, we are still sharing the same child cursor 0 (only one child cursor in V\$SQL_CS_HISTOGRAM). Let's try a sixth execution with the current bind variable value of 10000:

[Click here to view code image](#)

```

SQL> select count(1) from t_acs where n2 = :ln2;

COUNT(1)
-----
100000

```

The sixth execution at BUCKET_ID 1 (:ln2 = 10000) has made the cursor bind aware, and a new execution plan (CHILD_NUMBER 1) is compiled as follows:

[Click here to view code image](#)

```

SQL> select
      child_number
     ,bucket_id
     ,count
   from
     v$sql_cs_histogram
  where  sql_id = 'f2pmwazylrnfd' ;

```

CHILD_NUMBER	BUCKET_ID	COUNT
1	0	0
1	1	1
1	2	0
0	0	5 → COUNT of BUCKET_ID 0 = 5
0	1	5 → COUNT of BUCKET_ID 1 = 5
0	2	0

This is confirmed via the following select against v\$sql:

[Click here to view code image](#)

```
SQL> select
      sql_id
    ,child_number
  ,is_bind_sensitive
  ,is_bind_aware
  ,is_shareable
  from
    v$sql
  where
    sql_id = 'f2pmwazy1rnfd';
```

SQL_ID	CHILD_NUMBER	IS_BIND_SENSITIVE	IS_BIND_AWARE	IS_SHAREABLE
f2pmwazy1rnfd	0	Y	N	N
f2pmwazy1rnfd	1	Y	Y	Y

The same observation was found when a similar experiment was conducted with adjacent buckets 1 and 2 having a COUNT greater than 0, while the COUNT of bucket 0 was equal to 0.

Therefore, the secret sauce for the first case considered is: When the COUNT of a BUCKET_ID reaches the COUNT of its neighboring BUCKET_ID, the next execution marks the original cursor bind aware and a new child cursor is compiled. This is true provided the COUNT of the third BUCKET_ID equals 0.

Case 2: Distant Buckets 0 and 2 with a COUNT Greater Than Zero

We start this second experiment by executing the same query nine times (nine is a random number, which we can replace with any other number without altering the final result) using the bind variable value 100 (favoring bucket 0). Then we change the bind variable to 1000000 (favoring bucket 2) and requery, checking after each execution whether the cursor has been marked bind aware:

[Click here to view code image](#)

```
SQL> alter system flush shared_pool;
```

```

SQL> exec :ln2 := 100

SQL> select count(1) from t_acs where n2 = :ln2;

COUNT(1)
-----
100
SQL> repeat this 8 times

SQL> select
      child_number
    , bucket_id
    , count
  from
    v$sql_cs_histogram
  where  sql_id = 'f2pmwazylrnfd' ;

CHILD_NUMBER  BUCKET_ID      COUNT
-----  -----  -----
          0          0      9 → incremented 9 times = 9
executions
          0          1          0
          0          2          0

-- change the bind variable value
SQL> exec :ln2 := 1000000

SQL> select count(1) from t_acs where n2 = :ln2;

COUNT(1)
-----
1099049
SQL> repeat this query 2 times

```

After three executions using a bind variable value of 1000000, which processes more than a million rows and causes an increase in the COUNT of the distant BUCKET_ID 2), we are still sharing the same child cursor 0:

[Click here to view code image](#)

```

SQL> select
      child_number
    , bucket_id
    , count
  from
    v$sql_cs_histogram
  where  sql_id = 'f2pmwazylrnfd' ;

CHILD_NUMBER  BUCKET_ID      COUNT
-----  -----  -----

```

```

-----
          0           0           9 → ceil(9/3) = 3 = COUNT of
BUCKET_ID 2
          0           1           0
          0           2           3 → incremented 3 times = 3
executions

```

Let's try a fourth execution using this current bind variable value of 1000000:

[Click here to view code image](#)

```

SQL> select count(1) from t_acs where n2 = :ln2;

COUNT(1)
-----
1099049

SQL> select
      child_number
    , bucket_id
    , count
  from
    v$sql_cs_histogram
  where sql_id = 'f2pmwazylrnfd' ;

CHILD_NUMBER  BUCKET_ID  COUNT
-----
          1          0          0
          1          1          0 → bucket 1 not involved
1          2          1
          0          0          9 → bucket 0 incremented 9 times:
ceil(9/3)= 3
          0          1          0
          0          2          3 → bucket 2 incremented 3 times:
3 = ceil (9/3)

```

That's it! The fourth execution at BUCKET_ID 2 has marked the cursor bind aware, and a new execution plan (CHILD_NUMBER 1) has been compiled.

The preceding experiment uses several different numbers of initial executions using bind variables favoring BUCKET_IDS 0 and 2. We started by incrementing the BUCKET_ID 0 and then BUCKET_ID 2 (and vice versa). All those experiments explain that the secret sauce used by Oracle to mark a cursor bind aware in this case is: when the COUNT of BUCKET_ID 2 reaches ceil(COUNT of BUCKET_ID 0/3), then the next execution at BUCKET_ID 2 will mark the cursor bind aware and compile a new execution plan. This is true provided the COUNT of BUCKET_ID 1 remains at its initial value of 0.

Case 3: All Buckets with a COUNT Greater Than Zero

In this last series of tests, we'll try to figure out how Oracle decides that it is time to compile a new execution plan when the COUNTs of all buckets are greater than 0.

Despite the huge number of tests carried out in this case, we won't be able to reliably derive the secret sauce being used by Oracle to mark a cursor bind aware where all the buckets have a COUNT exceeding 0. Nevertheless, let's see what we can learn from our observations.

Let's start with the simple example where the COUNTs of all buckets are greater than 0:

[Click here to view code image](#)

```
SQL> select
      child_number
      ,bucket_id
      ,count
  from
    v$sql_cs_histogram
 where  sql_id = 'f2pmwazylrnfd';
```

CHILD_NUMBER	BUCKET_ID	COUNT
0	0	3
0	1	1
0	2	1

It goes without saying that to get the preceding results, we flushed the shared pool, executed the query at BUCKET_ID 0 ($:ln2 = 100$) three times, once at BUCKET_ID 1 ($:ln2 = 10000$), and finally once at BUCKET_ID 2 ($:ln2 = 1000000$).

You can see from this situation that any execution at any BUCKET_ID (i.e., using any value among the three bind variables' values) will mark the cursor bind aware:

[Click here to view code image](#)

```
SQL> exec :ln2 := 100
SQL> select count(1) from t_acs where n2 = :ln2;
```

CHILD_NUMBER	BUCKET_ID	COUNT
1	0	1 → when the 6th execution is for :ln2=100
1	1	0
1	2	0
0	0	3
0	1	1
0	2	1

```
SQL> exec :ln2 := 10000
```

```
SQL> select count(1) from t_acs where n2 = :ln2;
```

CHILD_NUMBER	BUCKET_ID	COUNT
1	0	0
1	1	1 → when the 6th execution is for :ln2 = 100000
1	2	0
0	0	3
0	1	1
0	2	1

SQL> exec :ln2 := 1000000

SQL> select count(1) from t_acs where n2 = :ln2;

CHILD_NUMBER	BUCKET_ID	COUNT
1	0	0
1	1	0
1	2	1 → when the 6th execution is for :ln2 = 1000000
0	0	3
0	1	1
0	2	1

Unfortunately, we can't derive the bind-aware secret sauce in this case.

In the following case, we manage to have 11 executions at BUCKET_ID 0 (:ln2 = 100), 4 executions at BUCKET_ID 2 (:ln2 = 10000), and 3 executions at BUCKET_ID 3 (:ln2 = 1000000) in that order of execution:

[Click here to view code image](#)

CHILD_NUMBER	BUCKET_ID	COUNT
0	0	11
0	1	4
0	2	3

Before reaching this bucket COUNT distribution, we would need to take great care to avoid the cases mentioned earlier. For example, had we started with 11 executions at BUCKET_ID 0 and then executed the query five ($> \text{ceil}(11/3)$) times at the distant BUCKET_ID 2, then the cursor would have been marked bind aware even before we could have attempted an execution at BUCKET_ID 1.

This next execution at BUCKET_ID 2 makes the cursor bind aware, and a new execution plan (CHILD_NUMBER 1) is generated as follows:

[Click here to view code image](#)

```
SQL> exec :ln2 := 1000000
SQL> select count(1) from t_acs where n2 = :ln2;
SQL> select
      child_number
```

```

,bucket_id
, count
from
  v$sql_cs_histogram
where   sql_id = 'f2pmwazylrnfd' ;


```

CHILD_NUMBER	BUCKET_ID	COUNT
1	0	0
1	1	0
1	2	1 → execution done at this bucket
0	0	11
0	1	4
0	2	3

Again, we are unable to derive the bind-aware secret sauce in this case. However, in the blog article “Bind aware secret sauce (again)” (<https://hourim.wordpress.com/2015/09/05/bind-aware-secret-sauce-again>), you can find a Procedural Language/Structured Query Language (PL/SQL) function, `fv_will_cs_be_bind_aware`, which might give you an indication of when a cursor is marked bind when the COUNTs of all buckets are greater than 0. For example, when this function is applied to the preceding two situations, it gives the following:

[Click here to view code image](#)

```
SQL> select fv_will_cs_be_bind_aware(3,1,1) IS_BIND_AWARE from dual;
```

```
IS_BIND_AWARE
-----
Y
```

```
SQL> select fv_will_cs_be_bind_aware(11,4,3) IS_BIND_AWARE from dual;
```

```
IS_BIND_AWARE
-----
Y
```

The Bind-Aware Cursor

As we have seen, before a cursor becomes bind aware and optimal execution plans are generated on the basis of the selectivity of the bind variable values, it has to go wrong (share the initial plan whatever the bind variable value may be). This warm-up period can dramatically vary depending on the number of executions done at the three different buckets. We can't avoid this warm-up period. Once the cursor becomes bind aware, the optimal plans are generated following a new algorithm that has nothing to do with the (COUNT, BUCKET_ID) tandem exposed extensively previously. Of course, this is true provided that the parent-child cursors are still in the shared pool.

Once a cursor is bind aware, a record is inserted into the V\$SQL_CS_SELECTIVITY dynamic view for the first time, as shown in the following, using the simplest case of two adjacent BUCKET_IDS:

[Click here to view code image](#)

```
SQL> alter system flush shared_pool;
SQL> exec :ln2 := 100
SQL> select count(1) from t_acs where n2 = :ln2;

SQL> exec :ln2 := 10000
SQL> select count(1) from t_acs where n2 = :ln2;

SQL> select
      child_number
     ,predicate
     ,low
     ,high
   from
     v$sql_cs_selectivity
  where
    sql_id = 'f2pmwazy1rnfd';

no rows selected
```

The no rows selected line indicates that the cursor is still not bind aware. Let's execute the query once more and check the contents of the V\$SQL_CS_SELECTIVITY view:

[Click here to view code image](#)

```
SQL> print :ln2
LN2
-----
10000
SQL> select count(1) from t_acs where n2 = :ln2;

SQL> select
      child_number
     ,predicate
     ,low
     ,high
   from
     v$sql_cs_selectivity
  where
    sql_id = 'f2pmwazy1rnfd';

CHILD_NUMBER
PREDICATE          LOW          HIGH
-----  -----  -----
-----  -----  -----
```

1

=LN2

0.074579 0.091152

When this view gets populated, it indicates that the cursor CHILD_NUMBER 1 has been marked bind aware and from now on, for every execution, using the extended cursor sharing layer code, Oracle Optimizer will peek at the bind variable value and check its selectivity. If the selectivity of this bind variable value is found to be between the LOW and HIGH columns values, it will share the execution plan represented by the existing CHILD_NUMBER 1. Otherwise, a new plan with a new range of selectivity (low and high) is generated as follows:

[Click here to view code image](#)

```
SQL> exec :ln2 := 100
SQL> select count(1) from t_acs where n2 = :ln2;

COUNT(1)
-----
100

SQL> select
      child_number
    , predicate
    , low
    , high
  from
    v$sql_cs_selectivity
 where
   sql_id = 'f2pmwazy1rnfd';

CHILD_NUMBER
PREDICATE          LOW          HIGH
-----  -----
2
=LN2          0.000361  0.091152
1
=LN2          0.074579  0.091152
```

Notice that a new plan (CHILD_NUMBER 2) has been generated, and its LOW and HIGH values have been updated to reflect the selectivity of the new peeked bind variable (:ln2 = 100).

Let's now use the extreme bind variable value, which favors a full table scan and increments the COUNT of BUCKET_ID 2 (remember that at this stage the [BUCKET_ID, COUNT] tandem does not play any role):

[Click here to view code image](#)

```
SQL> exec :ln2 := 1000000
SQL> select count(1) from t_acs where n2 = :ln2;
```

```

COUNT(1)
-----
1099049

SQL> select
      child_number
    , predicate
    , low
    , high
  from
    v$sql_cs_selectivity
 where
   sql_id = 'f2pmwazy1rnfd';

CHILD_NUMBER
PREDICATE          LOW           HIGH
-----  -----
3
=LN2          0.823886  1.006972
2             0.000361  0.091152
=LN2          0.074579  0.091152
1
=LN2

```

Note again that a new plan (CHILD_NUMBER 3) has been generated and its LOW and HIGH values have been updated accordingly to reflect the selectivity of the new peeked bind variable (:ln2 = 1000000).

The Oracle documentation says that when the selectivity for the peeked bind variable value is not found within an existing low and high value range, a new plan is generated. If this new plan is found to be equivalent to an existing one, the LOW and HIGH values of that existing plan will be updated without creating a new bind-aware plan. We won't test this case in this chapter. It is, nevertheless, important to emphasize here that there are two stages in the ACS feature:

- Stage 1: The cursor is not marked bind aware.
- Stage 2: The cursor is marked bind aware.

It is during the first stage that ACS layer code is in action using the (BUCKET_ID, COUNT) relationship to mark a cursor bind aware. The extended cursor sharing layer code kicks in during the second stage, which peeks at each bind variable, gets its selectivity, and checks whether a child cursor exists within the selectivity range of the specified bind value. If it exists, the child cursor is shared. Otherwise, a new child cursor is generated with a new range of selectivity. If the new child cursor (in fact, the new execution plan) is found to be equivalent to an existing one, then the new selectivity will be merged with that of the existing equivalent child cursor.

Finally, [Figure 4.2](#) summarizes the ACS–ECS triggering algorithm.

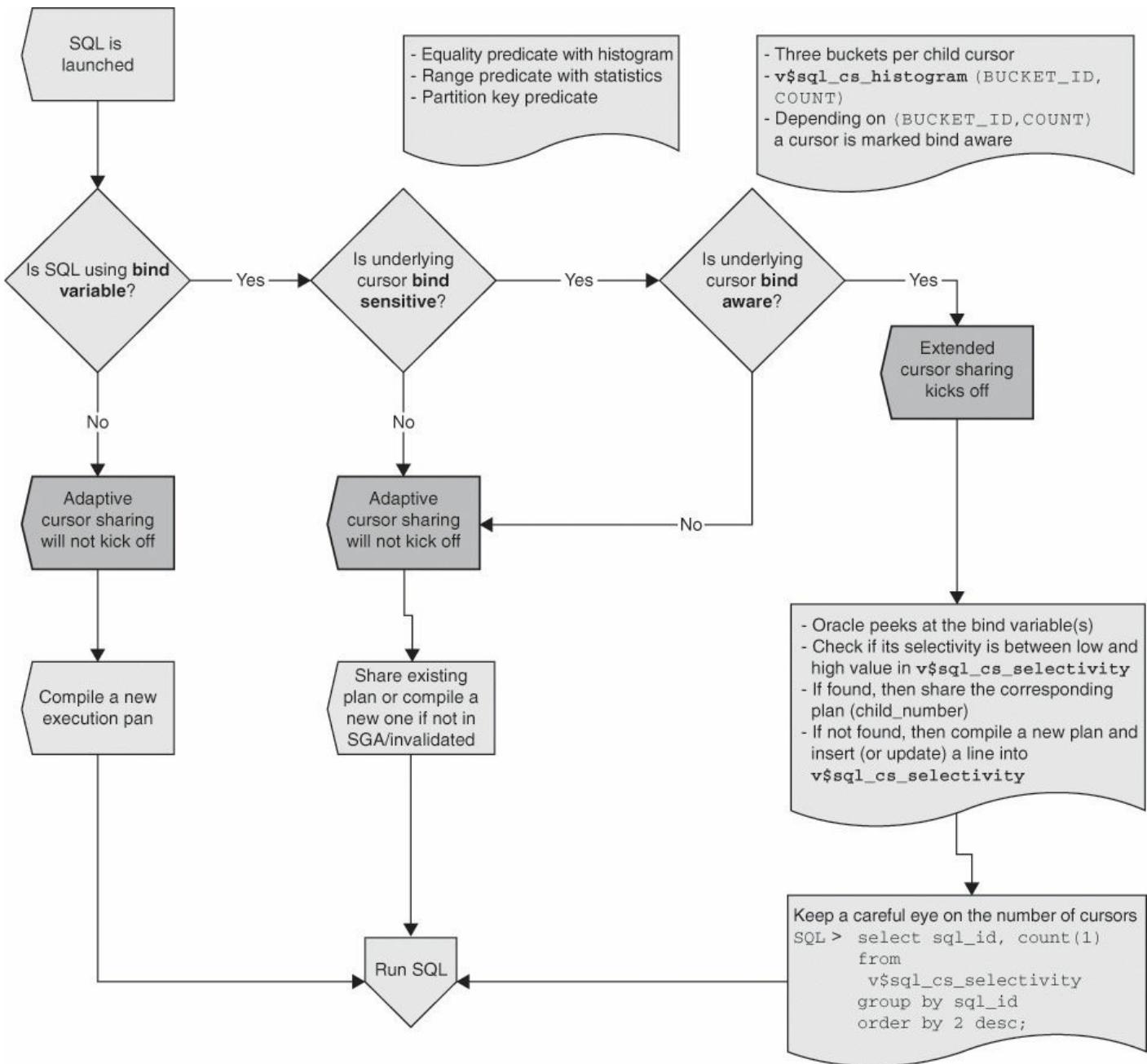


Figure 4.2 ACS–ECS triggering algorithm

A Practical Case

Earlier, this chapter proclaimed that ACS is an answer to sharing cursors and optimizing SQL. The ACS principles we demonstrated, when applied against gentle models like the one used throughout this chapter, seem perfect—that is, until we start troubleshooting the following performance issue in a real-life running system. We realize, surprisingly, that the root cause of this performance issue is the ACS feature.

[Click here to view code image](#)

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

```
BANNER
```

```
Oracle Database 11g Enterprise Edition Release 11.2.0.3.0 - 64bit
```

```

Production
PL/SQL Release 11.2.0.3.0 - Production
CORE    11.2.0.3.0      Production
TNS for Linux: Version 11.2.0.3.0 - Production
NLSRTL Version 11.2.0.3.0 - Production

```

```

SQL> select
      sql_id
     ,count(1)
  from
    v$sql
 where executions < 2
 group by sql_id
 having count(1) > 10
 order by 2 desc;

```

SQL_ID	COUNT(1)
7zwq7z1nj7vga	44217
39ax31acw29z6	75
0v3dvmc22qnam	29
412j04p609svj	25
5s34t44u10q4g	17
c8gnrhxma4tas	16
g8m7zdgak6pmm	16
gjm43un5cy843	16
6wdu577suw74s	15
23nad9x295gkf	14
848dyu9288c3h	14
6wm3n4d7bnddg	14
2am60vd2kw8ux	14
0ctk7jpx5chm	14
3x13xht9dh5c3	14
gdn3ysuyssf82	13
gg17hgzmmttbu	13
53ps5ua5ms44q	13
1z2rnd9bubah9	13

What caused the first SQL_ID (7zwq7z1nj7vga) to have 44,217 versions in V\$SQL? An odd phenomenon is happening in this system, considering that the application is using bind variables and therefore should normally reuse cursors and share resources. The first and obvious thing to do in this case is figure out why sharing of child cursors is preempted. Tanel Poder script, *nonshared*, when applied to this particular SQL_ID, gives us the following result:

[Click here to view code image](#)

```
SQL> @nonshared 7zwq7z1nj7vga
```

```
Show why existing SQL child cursors were not reused
(V$SQL_SHARED_CURSOR) ...
```

```

-----
SQL_ID          : 7zwq7z1nj7vga
ADDRESS         : 000000406DBB30F8
CHILD_ADDRESS   : 00000042CE36F7E8
CHILD_NUMBER    : 0
BIND_EQUIV_FAILURE : Y
REASON          : <ChildNode><ChildNumber>0</ChildNumber>
<ID>40</ID>
<size>2x4</size>
<reason>Bindmismatch (33)</reason>
<init_ranges_in_first_pass>0</init_ranges_in_>
<selectivity>1097868685</selectivity>
</ChildNode>

-----
SQL_ID          : 7zwq7z1nj7vga
ADDRESS         : 000000406DBB30F8
CHILD_ADDRESS   : 00000045B5C5E478
CHILD_NUMBER    : 1
BIND_EQUIV_FAILURE : Y
REASON          : <ChildNode><ChildNumber>1</ChildNumber>
<ID>40</ID>
<size>2x4</size>
<reason>Bindmismatch (33)</reason>
<init_ranges_in_first_pass>0</init_ranges_in_>
<selectivity>915662630</selectivity>
</ChildNode>

-----
SQL_ID          : 7zwq7z1nj7vga
ADDRESS         : 000000406DBB30F8
CHILD_ADDRESS   : 00000038841E2868
CHILD_NUMBER    : 2
BIND_EQUIV_FAILURE : Y
REASON          : <ChildNode><ChildNumber>2</ChildNumber>
<ID>40</ID>
<size>2x4</size>
<reason>Bindmismatch (33)</reason>
<init_ranges_in_first_pass>0</init_ranges_in_>
<selectivity>163647208</selectivity>
</ChildNode>

-----
SQL_ID          : 7zwq7z1nj7vga
ADDRESS         : 000000406DBB30F8
CHILD_ADDRESS   : 00000038841E2708
CHILD_NUMBER    : 3
BIND_EQUIV_FAILURE : Y
REASON          : <ChildNode><ChildNumber>3</ChildNumber>
```

```

<ID>40</ID>
<size>2x4</size>
<reason>Bindmismatch (33)</reason>
<init_ranges_in_first_pass>0</init_ranges_in_>
<selectivity>4075662961</selectivity>
<ChildNode>

.../...

-----
SQL_ID : 7zwq7z1nj7vga
ADDRESS : 000000406DBB30F8
CHILD_ADDRESS : 00000042DD3D7208
CHILD_NUMBER : 97
BIND_EQUIV_FAILURE : Y
REASON : <ChildNode><ChildNumber>97</ChildNumber>

<ID>40</ID>
<size>2x4</size>
<reason>Bindmismatch (33)</reason>
<init_ranges_in_first_pass>0</init_ranges_in_>
<selectivity>3246589452</selectivity>
<ChildNode>

-----
SQL_ID : 7zwq7z1nj7vga
ADDRESS : 000000406DBB30F8
CHILD_ADDRESS : 00000042DD3D70A8
CHILD_NUMBER : 98
BIND_EQUIV_FAILURE : Y
REASON : <ChildNode><ChildNumber>98</ChildNumber>

<size>2x4</size>
<reason>Bindmismatch (33)</reason>
<init_ranges_in_first_pass>0</init_ranges_in_>
<selectivity>3246589452</selectivity>
<ChildNode>

-----
SQL_ID : 7zwq7z1nj7vga
ADDRESS : 000000406DBB30F8
CHILD_ADDRESS : 00000045B5C5E5D8
CHILD_NUMBER : 99
BIND_EQUIV_FAILURE : Y
REASON : <ChildNode><ChildNumber>99</ChildNumber>

<size>2x4</size>
<reason>Bindmismatch (33)</reason>
<init_ranges_in_first_pass>0</init_ranges_in_>
<selectivity>3246589452</selectivity>
<ChildNode>

```

There are 100 (0–99) nonshared child cursors due to BIND_EQUIV_FAILURE:

[Click here to view code image](#)

```
SQL> select
      count(1)
    from
      v$sql_shared_cursor
  where SQL_ID = '7zwq7z1nj7vga';
```

```
COUNT (1)
-----
45125
```

```
SQL> select
      count(1)
    from
      v$sql_shared_cursor
  where sql_id = '7zwq7z1nj7vga'
    and BIND_EQUIV_FAILURE = 'Y';
```

```
COUNT (1)
-----
45121
```

In almost all (99%) of the cases, the reason sharing is preempted for SQL_ID 7zwq7z1nj7vga is BIND_EQUIV_FAILURE. The official Oracle documentation defines this reason: “The bind value’s selectivity does not match that used to optimize the existing child cursor.” This definition gives a clue to the reason Oracle is failing to share an existing cursor. As explained earlier in this chapter, there are two layers of code implemented to manage sharing of cursors: the ACS layer code is responsible for marking a cursor bind aware and the extended cursor sharing (ECS) layer code is responsible for attaching (or compiling) a child cursor corresponding to the bind variable selectivity. We observed that when the cursor becomes bind aware, the ECS layer code kicks in for each execution of the underlying query. It peeks at the bind variables (there might be several) and matches their selectivity with the one in the V\$SQL_CS_SELECTIVITY dynamic view.

Let’s see what we can find in this selectivity table for the SQL_ID 7zwq7z1nj7vga:

[Click here to view code image](#)

```
SQL> select
      count(1)
    from
      v$sql_cs_selectivity
  where
    sql_id = '7zwq7z1nj7vga';
```

```
COUNT (1)
-----
16,847,320
```

This table contains a huge number of records, which might dramatically affect the execution of the initial query. Each time we run the query, Oracle has to issue a select, behind the scenes, against this huge V\$SQL_CS_SELECTIVITY table in order to check if any existing child cursor already covers the selectivity of the bind variables.

What makes the situation even more dramatic is that, among those 100 child cursors, there are only four distinct execution plans, as the following shows:

[Click here to view code image](#)

```
SQL> select * from table(dbms_xplan.display_awr('7zwq7z1nj7vga'));
```

Plan hash value: **587060143**

Id	Operation	Name	Rows
0	SELECT STATEMENT		
1	TABLE ACCESS BY INDEX ROWID	MHO_TABLES_ACS	159K
2	INDEX RANGE SCAN	TABL_ACS_INDX1	159K

Note

- dynamic sampling used for this statement (level=4)

Plan hash value: **1114469665**

Id	Operation	Name	Rows
0	SELECT STATEMENT		
1	TABLE ACCESS BY INDEX ROWID	MHO_TABLES_ACS	1
2	INDEX RANGE SCAN	TABL_ACS_INDX_PK	1

Note

- dynamic sampling used for this statement (level=4)

Plan hash value: **2117864734**

Id	Operation	Name	Rows
0	SELECT STATEMENT		
1	TABLE ACCESS BY INDEX ROWID	MHO_TABLES_ACS	1

```
| 2 | INDEX RANGE SCAN           | TABL_ACS_INDX2 | 1 |
```

Note

- dynamic sampling used for this statement (level=4)

Plan hash value: **3054136074**

Id	Operation	Name
Rows		

0	SELECT STATEMENT	
1	TABLE ACCESS BY INDEX ROWID	
MHO_TABLES_ACS		159K
2	INDEX RANGE SCAN	
TABL_ACS_INDX3		159K

Note

- dynamic sampling used for this statement (level=4)

Finally, let's look at the SQL code of this query, which might give us a clue why we have been in such a situation:

```
SELECT
  {list_of_columns}
FROM
  mho_tables_acs
WHERE
  col_1    =:1
AND col_2    =:2
AND col_3    =:3
AND col_4    =:4
AND col_5    =:5
AND col_6    =:6
AND col_7    =:7
AND col_8    =:8
AND col_9    >=:9;
```

This query shows us two important points we have not investigated. First, will a cursor using more than 14 bind variables be bind aware and therefore not be affected by ACS? Second, will a query like this be handled with a bind-sensitive cursor if all nine

columns do not possess a histogram but the unique range predicate applied on COL_9 suffices to mark a cursor bind sensitive?

Summary

Beginning with Oracle Database 11g, ACS makes it possible for a SQL statement using bind variables to have multiple optimal execution plans. Certain prerequisites need to be fulfilled for ACS to kick in. More important, Oracle monitors the cursor to decide when it is to be marked bind aware and subsequently generates a new plan. Finally, in this “sharing and optimizing” conflict, there are two layers of code playing different roles: while the ACS layer is responsible for marking a cursor bind aware, the ECS layer peeks at the bind variables and, based on their selectivity, decides whether to share an existing plan or generate a new one.

There is, however, a limitation of ACS within PL/SQL that was intentionally not discussed in this chapter in order to keep it concise and reserved only for SQL queries.

5. Stabilizing Query Response Time Using SQL Plan Management

Every database administrator (DBA) has experienced the frustrating occasion when a query, which performed well earlier, suddenly starts deviating from its normal response time and becomes alarmingly lethargic. This performance degradation might be attributable to several direct or indirect causes, but it is fairly likely that the cause is a change in the execution plan. If you do not have enough in-house tuning experience to identify its root cause, but the problem requires an immediate solution and you want to set the critical query back to its original accepted response time, then using a SQL plan baseline to fix an acceptable execution plan and stabilize the query execution time might be a strategic solution.

This chapter discusses the SQL plan baseline, a mechanism of SQL plan management (SPM), and outlines briefly the different methods available to capture a plan in the SQL Management Base and attach it to the culprit query. It also demonstrates how to transfer an accepted SQL plan between two SQL statements. The chapter then explores how the cost-based optimizer interacts with a SQL plan baseline, and looks at the reasons a valid SPM plan might be rendered unusable. The concluding section investigates the SPM plan behavior in the presence of the adaptive cursor sharing (ACS) feature that was explained in [Chapter 4, “Adaptive Cursor Sharing.”](#)

Getting Started

You can find a great deal of helpful information about the SQL plan baseline in the official Oracle documentation, in many articles published by the Oracle Optimizer team, and in several blog articles. Rather than describe the concept, then, we jump right into a demonstration. Let's start by issuing a query for which two access paths are available: full table scan and index range scan. We will presume that the index range scan access path is the preferable one, and hence we ensure that the table will always be accessed through an index range scan.

Let's create the test table `t1` and the index `i1` with these presumptions in mind:

[Click here to view code image](#)

```
SQL> select * from v$version where rownum=1;
BANNER
-----
-----
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit
Production      0

SQL> create table t1
      (col1 number
```

```

, col2 varchar2(50)
, flag varchar2(2));

SQL> insert into t1
  select
    rownum
   , lpad('X', 50, 'X')
   , case when rownum = 1
          then 'Y1'
        when rownum = 2
          then 'Y2'
        when mod(rownum, 2) = 0
          then 'N1'
        else 'N2'
      end
  from dual
 connect by rownum <= 100000;

SQL> create index i1 on t1(flag);

-- gather statistics without histogram
SQL> exec dbms_stats.gather_table_stats(user, 't1', method_opt =>
 'for all columns size 1');

```

For the sake of simplicity, we set the CURSOR_SHARING parameter to FORCE so that the bind variable will occur under the hood (it is not advisable to do this in a production system):

[Click here to view code image](#)

```
SQL> alter session set cursor_sharing=force;
```

The flag column of the t1 table has a nonuniform data distribution, as shown in the following:

```
SQL> select
  flag, count(1)
from t1
group by flag;
```

FL	COUNT(1)
N1	49999
N2	49999
Y1	1
Y2	1

If you want to make the optimizer aware of this uneven distribution, you should collect a histogram on the flag column:

[Click here to view code image](#)

```
SQL> begin
```

```

exec dbms_stats.gather_table_stats(user
                                     , 't1'
                                     , method_opt => 'for columns flag
size skewonly');
end;

```

The following query, which searches for 49999 rows with flag = 'N1', when executed for the first time, will be hard parsed, and a new execution plan will be compiled by peeking at the transmitted bind variable:

[Click here to view code image](#)

```

SQL> select count(*), max(col2) from t1 where flag = 'N1';

COUNT(*)  MAX(COL2)
-----
49999 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

```
SQL_ID  6fbvysnhkvugw, child number 0
-----
select count(*), max(col2) from t1 where flag = :"SYS_B_0"
```

```
Plan hash value: 3724264953
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)
0	SELECT STATEMENT				273
(100)					
1	SORT AGGREGATE		1	54	
2	TABLE ACCESS FULL	T1	48640	2565K	273 (1)
0:00:01					

```
Predicate Information (identified by operation id):
```

```
2 - filter("FLAG"=:SYS_B_0)
```

If we re-execute the same query but this time use a flag value of 'Y1', which will return just one row, we will share the same execution plan (remember that we are using a bind variable behind the scenes thanks to the FORCE value of the CURSOR_SHARING parameter):

[Click here to view code image](#)

```
SQL> select count(*), max(col2) from t1 where flag = 'Y1';
```

```

COUNT(*) MAX(COL2)
-----
1 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

SQL> select * from table(dbms_xplan.display_cursor);

SQL_ID  6fbvysnhkvugw, child number 0
-----
select count(*), max(col2) from t1 where flag = :"SYS_B_0"

Plan hash value: 3724264953
-----
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU) |
Time   |
-----
|   0 | SELECT STATEMENT |      |       |        |    273
(100) |
|   1 |   SORT AGGREGATE |      |       |      1 |     54
|       |                 |
|*  2 | TABLE ACCESS FULL| T1   | 48640 | 2565K|    273  (1)
00:00:01 |

Predicate Information (identified by operation id):
-----
2 - filter("FLAG"=:SYS_B_0)

```

Accessing a single row via a full table scan is obviously not optimal. This is the dark side of using bind variables: when we share everything, we share execution plans as well!

In order to solve this forced plan-sharing issue, Oracle Database 11g introduced adaptive cursor sharing (see [Chapter 4](#) for more details). This feature requires, however, that the SQL statement employ bind variables, and histograms should exist on the columns containing the bind values.

Moreover, a skewed column, even with a histogram, when used in a WHERE clause can cause severe performance problems. First, histograms (at least before Oracle Database 12c) are very expensive to collect, and you need to collect them at the right moment when the data distribution is nonuniform. Moreover, uneven data distribution might not be precisely captured by the DBMS_STATS package, as in the case of height-balanced histograms. Additionally, when the cursor becomes bind aware and therefore subject to extended cursor sharing (see [Chapter 4](#)), for every parse, the query optimizer performs an estimation of the selectivity of the predicates. Based on that estimation, an already existing child cursor is used. Or, if no cursor for that selectivity range exists, a new child cursor is created. This approach generates multiple execution plans for the

same query, and the database engine groups together bind variable values that have about the same selectivity and, consequently, should lead to the same execution plan. In cases where the SQL query uses multiple bind variables, drastic performance issues can arise due to incorrect selectivity estimates and extra overhead of selectivity estimation during every parse. For all of these reasons, using a SQL plan baseline might sometimes prove to be the perfect solution.

The next section explains how we can fix a plan using SQL plan baseline so that the query mentioned earlier always employs an index range scan access path.

Creating a SQL Plan Baseline

There are several ways of creating a SQL plan baseline for a given query (generally identified by a `sql_id`). Let's begin with the obvious (yet maybe the less used) option, capturing a plan automatically.

Capturing Plans Automatically

Two important initialization parameters go hand in hand with SPM:

[Click here to view code image](#)

```
SQL> show parameter baseline
```

NAME	TYPE	VALUE
optimizer_capture_sql_plan_baselines	boolean	FALSE
optimizer_use_sql_plan_baselines	boolean	TRUE

When the `optimizer_use_sql_plan_baselines` parameter value is set to TRUE (the default value), Oracle Optimizer uses the execution plan in the SQL Management Base even when it generates a different plan. This, of course, is true provided the plan stored in the SQL Management Base is still reproducible (we will see this in detail later in this chapter).

When the `optimizer_capture_sql_plan_baselines` parameter value is set to TRUE (the default value is FALSE), Oracle automatically captures a SQL plan baseline for every repeatable SQL statement. Let's make this clear via a reproducible example:

[Click here to view code image](#)

```
SQL> alter session set optimizer_capture_sql_plan_baselines=TRUE;

SQL> Select count(*), max(col2) From t1 where flag = 'Y1';

COUNT(*)  MAX(COL2)
-----
1 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
SQL> /
```

```

COUNT (*)  MAX (COL2)
-----
1 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
SQL> alter session set optimizer_capture_sql_plan_baselines=FALSE;

```

Notice the use of the capital letter *S* in the word *Select*, which forces a hard parse. This code example also uses the condition *flag = 'Y1'*, which returns only one row and hence results in an index range scan path, which we want to capture in the SQL plan baseline. Notice also that the query was executed twice in order to be captured into a SQL plan baseline, as only repeatable SQL statements are captured.

Let's see now if Oracle has captured the index range scan plan:

[Click here to view code image](#)

```

SQL> select
      to_char(signature) signature
    ,plan_name
    ,origin
    ,enabled
    ,accepted
  from
    dba_sql_plan_baselines
 where
    sql_text like '%Select count(*)%';

```

SIGNATURE	PLAN_NAME	ORIGIN
ACC		
---	---	---
15340826253708983785	SQL_PLAN_d9tch6banyzg97823646b	AUTO-
CAPTURE	YES	YES

We can see that, indeed, Oracle has inserted a new execution plan into the SQL Management Base. This stored plan has been immediately marked as accepted so that any SQL statement in the library cache (*V\$SQL*) that has the property *exact_matching_signature*, which matches the SQL plan baseline signature, will be honored by this captured plan.

The following *SQL_IDs* represent the two semantically equivalent queries we've tried so far:

[Click here to view code image](#)

```

SQL> select
      sql_id
    ,child_number
    ,sql_text
  from
    v$sql a

```

```
where
  a.is_shareable = 'Y'
and exists (select
              null
            from
              dba_sql_plan_baselines b
            where
              b.signature = a.exact_matching_signature
            );
-----
```

SQL_ID	CHILD_N	SQL_TEXT
98ub2xj0nt01g	0	Select count(*), max(col2) From t1 where flag = :"SYS_B_0"
6fbvysnhkvugw	0	select count(*), max(col2) from t1 where flag = :"SYS_B_0"

As you can see, it is not strictly necessary for a SQL statement to be identical to the SQL text stored in the SQL plan baseline for it to be executed using SPM. All it needs is to have the following:

[Click here to view code image](#)

```
v$sql.exact matching signature = dba_sql_plan baselines.signature
```

To get the content of this captured SQL plan baseline, we can use the `dbms_xplan` package:

[Click here to view code image](#)

```
SQL> select * from
table(dbms_xplan.display_sql_plan_baseline(plan_name =>
'SQL_PLAN_d9tch6banyzg97823646b'));
```

SQL handle: SQL_d4e59032d54f7de9

SQL text: Select count(*), max(col2) From t1 where flag =
:"SYS_B_0"

Plan name: SQL_PLAN_d9tch6banyzg97823646b Plan id:
2015585387

Enabled: YES Fixed: NO Accepted: YES Origin: AUTO-
CAPTURE

Plan rows: From dictionary

```
Plan hash value: 497086120
```

Id	Operation	Name	Rows	Bytes
Cost	(%CPU)			
0	SELECT STATEMENT		1	54
2	(0)			
1	SORT AGGREGATE		1	54
2	TABLE ACCESS BY INDEX ROWID BATCHED	T1	1	54
2	(0)			
* 3	INDEX RANGE SCAN	I1	1	
	1 (0)			

```
Predicate Information (identified by operation id):
```

```
3 - access("FLAG"=:SYS_B_0)
```

From now on, any query that has the same `exact_matching_signature` as the signature of this SQL plan baseline will inherit the SPM execution plan, as shown in the following (notice the new flag value and the differences in the SQL text of the query):

[Click here to view code image](#)

```
SQL> SELECT count(*), max(col2) FROM t1 where flag = 'N2';
```

```
COUNT(*) MAX(COL2)
```

```
49999 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

Id	Operation	Name	Rows	Bytes
Cost	(%CPU)			
0	SELECT STATEMENT		1002 (100)	
1	SORT AGGREGATE		1	54
2	TABLE ACCESS BY INDEX ROWID BATCHED	T1	50887	
2683K	1002 (1)			
* 3	INDEX RANGE SCAN	I1	50887	
	100 (0)			

Predicate Information (identified by operation id):

3 - access("FLAG"=:SYS_B_0)

Note

- SQL plan baseline SQL_PLAN_d9tch6banyzg97823646b used for this statement

SQL> SeLeCt count(*), max(col2) FROM t1 where flag = 'Y2';

COUNT(*) MAX(COL2)

1 XXX

SQL> select * from table(dbms_xplan.display_cursor);

Id	Operation	Name	Rows	Bytes
Cost (%CPU)				
0	SELECT STATEMENT		2 (100)	
1	SORT AGGREGATE		1	54
2	TABLE ACCESS BY INDEX ROWID BATCHED	T1	1	54
2	(0)			
* 3	INDEX RANGE SCAN	I1	1	
	1 (0)			

Predicate Information (identified by operation id):

3 - access("FLAG"=:SYS_B_0)

Note

- SQL plan baseline SQL_PLAN_d9tch6banyzg97823646b used for this statement

The Note at the bottom of the plan is there to remind you that your SQL query has been executed via an execution plan coming from a SQL plan baseline.

Loading Plans from the Cursor Cache

If you don't want your SQL query to be constrained by the index range scan plan that you previously captured automatically into a SQL plan baseline, you can disable it using either of the following two options, setting its property ENABLED to NO or dropping it from the SQL plan baseline using the DBMS_SPM package:

[Click here to view code image](#)

```
SQL> declare
      rs pls_integer;
begin
  -- disabling a SQLplan from a SQL Management Base
  rs := dbms_spm.alter_sql_plan_baseline
    (plan_name      => 'SQL_PLAN_d9tch6banyzg97823646b'
     ,attribute_name => 'ENABLED'
     ,attribute_value => 'NO'
    );
  --
  -- dropping a SQL plan from a SQL Management Base
  rs := dbms_spm.drop_sql_plan_baseline
    (plan_name      => 'SQL_PLAN_d9tch6banyzg97823646b'
     );
end;
/
PL/SQL procedure successfully completed.
```

Now, if you execute your query with a value favoring a full table scan, you will not be constrained by the previously captured index range scan execution plan, as shown in the following:

[Click here to view code image](#)

```
SQL> SELECT count(*), max(col2) FROM t1 where flag = 'N2';

COUNT(*)  MAX(COL2)
-----
49999 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

SQL> select * from table(dbms_xplan.display_cursor);

SQL_ID  9cmb2wv5gkq2x, child number 0
-----
-----| Id  | Operation          | Name | Rows  | Bytes | Cost  (%CPU) |
-----|   0 | SELECT STATEMENT   |       |        |        | 273  (100) |
|   1 |   SORT AGGREGATE   |       |        1 |      54 |           |
|*  2 | TABLE ACCESS FULL | T1   | 50887 | 2683K| 273  (1)

-----| Predicate Information (identified by operation id): |
-----
```

```
2 - filter("FLAG"=:SYS_B_0)
```

If you want to fix the above full table scan execution plan, then instead of enabling the automatic capture (which involves the risk of forgetting to reset it to its default value), you can capture this execution plan into a SQL plan baseline from the cursor cache using the DBMS_SPM package and the corresponding SQL_ID as follows:

[Click here to view code image](#)

```
SQL> declare
      rs pls_integer;
begin
  rs := dbms_spm.load_plans_from_cursor_cache('9cmb2wv5gkq2x');
end;
/
PL/SQL procedure successfully completed.
```

We can check the effectiveness of fixing the full table scan plan by trying one of the earlier semantically equivalent queries and by observing the Note at the bottom of the corresponding execution plan:

[Click here to view code image](#)

```
SQL> SELECT count(*), MAX(col2) FROM t1 where flag = 'Y2';

  COUNT(*)  MAX(COL2)
-----
 1 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

SQL> select * from table(dbms_xplan.display_cursor);

-----
| Id  | Operation          | Name | Rows  | Bytes | Cost  (%CPU) |
Time   |
-----
|   0 | SELECT STATEMENT |       |       |       |       | 273
(100) |
|   1 | SORT AGGREGATE   |       |       |       |       | 54
|       |                   |
|*  2 | TABLE ACCESS FULL| T1   |       |       | 54 | 273  (1)
00:00:01 |

-----
Predicate Information (identified by operation id):
-----
 2 - filter("FLAG"=:SYS_B_0)
```

Note

```
-----  
- SQL plan baseline SQL_PLAN_d9tch6banyzg9616acf47 used for this  
statement
```

There are a couple other techniques for capturing a SQL plan baseline, such as loading plans from a SQL tuning set, which you can use while upgrading, for example, from an Oracle 10g database. You can also move baselines between databases by exporting and importing baseline plans from a staging table. More details about these two techniques can be found in the Oracle official documentation.

Faking Baselines

Possibly the most used technique to stabilize an execution plan is transferring SQL plan baselines between two semantically equivalent SQL statements having two distinct SQL_IDs. For example, consider the following non-hinted query:

[Click here to view code image](#)

```
SQL> SELECT count(*), max(col2) FROM t1 where flag = 'N2';
```

You want this query to be run with the execution plan of the following hinted version of the same query:

[Click here to view code image](#)

```
SQL> SELECT /*+ index (t1) */ count(*), max(col2) FROM t1 where  
flag = 'N2';
```

Since you are not allowed to change the SQL code directly in the application because it is third-party software, you need to use a SQL plan baseline to accomplish this task. This section explains the steps for reaching this goal.

You first need to load the execution plan of the hinted query into a SQL plan baseline by executing it once and loading the corresponding execution plan from cursor cache, as follows:

[Click here to view code image](#)

```
SQL> SELECT /*+ index (t1) */ count(*), max(col2) FROM t1 where  
flag = 'N2';
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

```
-----  
SQL_ID  fzk36bs5436us, child number 0 Plan hash value: 497086120  
-----
```

```
-----  
| Id  | Operation          | Name | Rows  | Bytes  
| Cost (%CPU) |  
-----  
|   0  | SELECT STATEMENT    |      |       | 1002 (100)|  
|   1  |   SORT AGGREGATE    |      |       | 1      | 54  
-----
```

```

|           |
|   2 | TABLE ACCESS BY INDEX ROWID BATCHED| T1      | 50887
| 2683K| 1002    (1) |
|* 3 | INDEX RANGE SCAN                  | I1      | 50887
|     | 100     (0) |
-----
```

Predicate Information (identified by operation id):

```
3 - access("FLAG"=:SYS_B_0)
```

```
SQL> declare
      rs pls_integer;
begin
  rs := dbms_spm.load_plans_from_cursor_cache('fxk36bs5436us');
end;
/
```

Let's now find out how many enabled plans we have in the SQL Management Base:

[Click here to view code image](#)

```
SQL> select
      sql_handle
      ,plan_name
      ,substr(sql_text,1,30) sql_text
      ,accepted
  from
    dba_sql_plan_baselines
 where
   enabled = 'YES'
 ;
```

SQL_HANDLE	PLAN_NAME	SQL_TEXT
SQL_d422e56d9adb0b66	SQL_PLAN_d88r5dqddq2v67823646b	SELECT /*+ index (t1) */ count YES
SQL_d4e59032d54f7de9	SQL_PLAN_d9tch6banyzg9616acf47	Select count(*), max(col2) Fro YES
SQL_d4e59032d54f7de9	SQL_PLAN_d9tch6banyzg97823646b	Select count(*), max(col2) Fro NO

As a result of the loading plan from the cursor cache discussed earlier, the following query with a hinted SQL plan baseline is already protected against execution plan changes:

[Click here to view code image](#)

```
SQL> SELECT count(*), max(col2) FROM t1 where flag = 'N2';
```

```

-----
SQL_ID  9cmb2wv5gkq2x, child number 0
-----
-----
| Id  | Operation          | Name | Rows | Bytes | Cost  (%CPU) |
Time   |
-----
|  0  | SELECT STATEMENT   |       |       |       | 273
(100) |
|  1  |   SORT AGGREGATE   |       |       | 1    | 54
|      |                   |
|* 2  | TABLE ACCESS FULL | T1   | 50887 | 2683K| 273  (1)
00:00:01 |
-----
-----
Predicate Information (identified by operation id):
-----
 2 - filter("FLAG"=:SYS_B_0)

```

Note

- SQL plan baseline SQL_PLAN_d9tch6banyzg9616acf47 used for this statement

Therefore, disabling the SQL plan baseline

SQL_PLAN_d9tch6banyzg9616acf47 mentioned in the Note in the previous code becomes necessary in order to preempt it from constraining the cost-based optimizer (CBO) plan:

[Click here to view code image](#)

```

SQL> declare
  ln_ps number;
begin
  ln_ps := dbms_spm.alter_sql_plan_baseline
            (sql_handle      => 'SQL_d4e59032d54f7de9'
            ,plan_name       => 'SQL_PLAN_d9tch6banyzg9616acf47'
            ,attribute_name  => 'enabled'
            ,attribute_value => 'NO');
end;
/

```

PL/SQL procedure successfully completed.

Finally, the last step is to link the SQL plan baseline (index range scan) of the hinted SQL_ID (fxk36bs5436us) to the non-hinted SQL_ID (9cmb2wv5gkq2x), as follows:

[Click here to view code image](#)

```

SQL> declare
      ln_ps pls_integer;
begin
  ln_ps := dbms_spm.load_plans_from_cursor_cache
  (sql_id  => 'fxk36bs5436us' -- sql_id of the index range
scan plan
  ,plan_hash_value => 497086120 -- plan hash value of the index
range scan plan
  ,sql_handle=> 'SQL_d4e59032d54f7de9' -- sql handle of the full
table scan plan
);
end;
/
PL/SQL procedure successfully completed.

```

Let's now check if we've successfully attached the SQL plan baseline of the hinted query to the semantically equivalent non-hinted query:

[Click here to view code image](#)

```

SQL> SELECT count(*), max(col2) FROM t1 where flag = 'N2';

  COUNT(*)  MAX(COL2)
-----
        49999 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

SQL> select * from table(dbms_xplan.display_cursor);

-----
SQL_ID  9cmb2wv5gkq2x, child number 0
-----
-----
| Id  | Operation          | Name | Rows  | Bytes
| Cost (%CPU) |
-----
|   0 | SELECT             |      |       | 1002 (100) |
STATEMENT
|   1 | SORT AGGREGATE    |      |       | 1 | 54
|       |
|   2 | TABLE ACCESS BY INDEX ROWID BATCHED| T1  | 50887
| 2683K| 1002 (1) |
|*  3 | INDEX RANGE SCAN   | I1  | 50887
|       | 100 (0) |
-----
-----
Predicate Information (identified by operation id):
-----
 3 - access("FLAG"=:SYS_B_0)

```

Note

- SQL plan baseline SQL_PLAN_d9tch6banyzg97823646b used for this statement

As you can see, we have successfully moved an index range scan SQL plan baseline from a hinted version of a SQL query to a non-hinted version of the same SQL query.

Carlos Sierra's script, which you can download from his blog (<http://carlos-sierra.net/2014/06/24/creating-a-sql-plan-baseline-out-of-a-modified-sql/>), is another way of reaching the same goal easily and more elegantly. Therefore, you should try the next method.

First, clean up the whole situation by dropping all existing SQL plan baselines (you should not do this in a running system without specifying a where clause in the for loop statement):

[Click here to view code image](#)

```
SQL> declare
      ln_ps pls_integer;
begin
  for x in (select plan_name from dba_sql_plan_baselines)
  loop
    ln_ps := dbms_spm.drop_sql_plan_baseline(plan_name =>
x.plan_name);
    end loop;
exception
  when others then
    null; -- don't use this in production
end;
/
PL/SQL procedure successfully completed.
```

```
SQL> select count(1) from dba_sql_plan_baselines;

  COUNT(1)
-----
          0
```

Simply put, suppose that you have a SQL_ID (9cmb2wv5gkq2x) representing a query honored via a full table scan plan and another SQL_ID (fxk36bs5436us) representing a query honored via an index range scan plan (PLAN_HASH_VALUE = 497086120). Now suppose you want to create a SQL plan baseline for the second tandem (fxk36bs5436us, 497086120) with which you want to constrain the plan of the first SQL_ID (9cmb2wv5gkq2x). Use Carlos Sierra's script (coe_load_sql_baseline.sql) to achieve this goal:

[Click here to view code image](#)

```
SQL> start coe_load_sql_baseline.sql 9cmb2wv5gkq2x fxk36bs5436us
```

497086120

coe_load_sql_baseline completed.

If we rerun the full table scan query, we see that it is now protected by an index range scan plan:

[Click here to view code image](#)

```
SQL> SELECT count(*), max(col2) FROM t1 where flag = 'N2';

SQL> select * from table(dbms_xplan.display_cursor);

SQL_ID  9cmb2wv5gkq2x, child number 0
-----
SELECT count(*), max(col2) FROM t1 where flag = :"SYS_B_0"

Plan hash value: 497086120
-----
| Id  | Operation          | Name | Rows  | Bytes |
| Cost (%CPU) |                               |
-----
|   0 | SELECT STATEMENT |       |   1002 (100) | |
|   1 |   SORT AGGREGATE |       |      1 |     54 |
|       |                   |
|   2 | TABLE ACCESS BY INDEX ROWID BATCHED | T1    | 50887  |
| 2683K|   1002 (1) |
|*  3 |   INDEX RANGE SCAN | I1    | 50887  |
|       |   100  (0) |
-----
| Predicate Information (identified by operation id): |
|-----|
| 3 - access("FLAG"=:SYS_B_0)                         |
| Note
|-----|
| - SQL plan baseline SQL_PLAN_d9tch6banyzg97823646b used for this statement
```

Having discussed what a SQL plan baseline is, how to capture it, and how to transfer it between two different SQL_IDs, in the next section we investigate the behavior of the Oracle CBO in the presence of one or more SQL plan baselines.

Oracle Optimizer and SPM Interaction

Since the goal of the CBO is to optimize an execution plan, how will it react if the plan

it is going to compile is constrained by a plan from a SQL plan baseline?

The best way to investigate this interaction is to walk through a CBO 10053 trace file event. In the previous section, we worked with two SQL plan baselines:

- SQL_PLAN_d9tch6banyzg97823646b for the index range scan path
- SQL_PLAN_d9tch6banyzg9616acf47 for the full table scan path

Let's suppose that they are both enabled and accepted:

[Click here to view code image](#)

```
SQL> select
      a.plan_name
    , b.plan_id
    , a.enabled
    , a.accepted
  from
    dba_sql_plan_baselines a
  , sys.sqlobj$ b
 where
    a.signature = b.signature
  and a.plan_name = b.name;
```

PLAN_NAME	PLAN_ID	ENA	ACC
SQL_PLAN_d9tch6banyzg9616acf47	1634389831	YES	YES → full table scan plan
SQL_PLAN_d9tch6banyzg97823646b	2015585387	YES	YES → index range scan plan

PLAN_ID is highlighted in bold because it is crucial information retrieved from SYS.SQLOBJ\$, an internal SPM table. This is the fundamental information with which Oracle uniquely identifies the execution plan stored in the SPM Management Base. We will see later in this section that any execution plan compiled by the CBO has to have a PLAN_HASH_2 value matching this PLAN_ID value in order for Oracle to use it.

When the CBO Plan Matches the SQL Plan Baseline

Investigating this case consists of executing a SQL statement with the same signature and deciphering the content of the corresponding CBO trace file:

[Click here to view code image](#)

```
SQL> alter session set events '10053 trace name context forever,
level 1';

SQL> SELECT count(*), max(col2) FROM t1 where flag = 'Y1';

COUNT(*)  MAX(COL2)
-----
1 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
1 row selected.
```

```
SQL> alter session set events '10053 trace name context off';
```

Unsurprisingly, at the very beginning of the trace file, the CBO starts by signaling that the SQL it is going to compile exists in a SQL Management Base:

```
SPM: statement found in SMB
```

It then identifies the current SQL statement:

[Click here to view code image](#)

```
*****
----- Current SQL Statement for this session (sql_id=4sdth4kka4ykw)
-----
SELECT count(*), max(col2) FROM t1 where flag = :"SYS_B_0"
*****
```

It proceeds to the usual execution plan compilation as if there were no constraining plans in the SQL Management Base. After the CBO plan has been compiled we can read the following lines in the same trace file:

[Click here to view code image](#)

```
SPM: cost-based plan found in the plan baseline, planId =
```

2015585387

```
SPM: cost-based plan successfully matched, planId = 2015585387
```

Thus, the CBO is signaling that the plan it came up with has a `PLAN_HASH_2` value that matches the `planId` of one of the two existing SQL plan baselines and prints this plan as follows:

[Click here to view code image](#)

```
=====
Plan Table
=====
-----+-----+
| Id | Operation | Name | Rows | Bytes
| Cost | Time | |
-----+-----+
| 0 | SELECT STATEMENT | | | 2
|   |   | |
| 1 |   SORT AGGREGATE | | | 1 | 54
|   |   | |
| 2 |   TABLE ACCESS BY INDEX ROWID BATCHED | T1 | 1 | 54
| 2 |   00:00:01 |
| 3 |   INDEX RANGE SCAN | I1 | 1
|   |   1 | 00:00:01 |
```

```

-----
-----+
Predicate Information:
-----
3 - access ("FLAG"=:SYS_B_0)

Content of other_xml column
=====
db_version      : 12.1.0.1
parse_schema    : C##MHOURI
plan_hash      : 497086120
plan_hash_2   : 2015585387

```

In fact, we can easily verify that the PLAN_HASH_2 value the CBO has just compiled for the current SQL_ID equals the planId of the SPM index range scan plan via the following SELECT:

[Click here to view code image](#)

```

SQL> SELECT
      p.sql_id
     ,p.plan_hash_value
     ,p.child_number
     ,t.phv2
  FROM
    v$sql_plan p
   ,xmltable('for $i in /other_xml/info
              where $i/@type eq "plan_hash_2"
              return $i'
             passing xmltype(p.other_xml)
             columns phv2 number path '/') t
 WHERE p.sql_id = '4sdth4kka4ykw'
   AND p.other_xml is not null;

SQL_ID      PLAN_HASH_VALUE CHILD_NUMBER    PHV2
-----      -----
4sdth4kka4ykw      497086120      0          2015585387

```

Since the CBO plan (PLAN_HASH_2) matches an SPM PLAN_ID, Oracle has decided to use the CBO plan.

If you repeat the same experiment using a bind variable favoring a full table scan path, you will find that the CBO comes up with a plan having a PLAN_HASH_2 value matching the planId of the full table scan SQL plan baseline as the following briefly shows:

[Click here to view code image](#)

```

*****
----- Current SQL Statement for this session (sql_id=gvrgd85zjjdps)
-----
select count(*), max(col2) FROM t1 where flag = :"SYS_B_0"

```

```
*****
SPM: cost-based plan found in the plan baseline, planId =
1634389831
SPM: cost-based plan successfully matched, planId = 1634389831

=====
Plan Table
=====

-----+-----+
| Id | Operation           | Name   | Rows | Bytes | Cost |
-----+-----+
| 0  | SELECT STATEMENT    |        |       |       | 273  |
|    |                   |         |       |       |      |
| 1  |   SORT AGGREGATE    |        |       | 1    | 54   |
|    |                   |         |       |      |
| 2  |     TABLE ACCESS FULL | T1    | 50K | 2683K | 273  |
| 00:00:04 |
-----+-----+
-----+
Predicate Information:
-----
2 - filter("FLAG"=:SYS_B_0)

Content of other_xml column
=====
db_version      : 12.1.0.1
parse_schema    : C##MHOURI
plan_hash       : 3724264953
plan_hash_2     : 1634389831
```

The first conclusion we can make at this stage of the investigation is very simple: when the CBO comes up with a plan (PLAN_HASH_2) matching an existing planId, then the CBO plan will be used to execute the current SQL query.

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

A second experiment which is worth investigation is when the plan compiled by the CBO does not match any planId in the SQL Management Base. A simple way to provoke such a situation is to set the optimizer back to one of its previous Oracle Database 11g releases:

[Click here to view code image](#)

```
SQL> alter session set optimizer_features_enable='11.2.0.3';
```

As a result, while accessing t1 table via index rowid, the optimizer will not use the Oracle 12c TABLE ACCESS BY INDEX ROWID BATCHED feature, and hence it will produce a different execution plan. Let's see this in action and remember that, until

now, we have only enabled and accepted two SQL plan baselines:

[Click here to view code image](#)

```
SQL> select plan_name from dba_sql_plan_baselines where accepted =  
'YES';  
PLAN_NAME  
-----  
SQL_PLAN_d9tch6banyzg9616acf47  
SQL_PLAN_d9tch6banyzg97823646b
```

And there is no non-accepted plan yet:

[Click here to view code image](#)

```
SQL> select plan_name from dba_sql_plan_baselines where accepted =  
'NO';  
no rows selected  
  
SQL> SELECT count(*), max(col2) FROM t1 where flag = 'Y1';  
  
COUNT(*) MAX(COL2)  
-----  
1 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

We immediately find that a new execution plan has been added into the SQL Management Base with an accepted property set to NO, as shown in the following:

[Click here to view code image](#)

```
SQL> select plan_name from dba_sql_plan_baselines where accepted =  
'NO';  
PLAN_NAME  
-----  
SQL_PLAN_d9tch6banyzg98576eb1f
```

This is a clear indication that the CBO has come up with a different execution plan that has a PLAN_HASH_2 value that does not match a planId of the two enabled and accepted SQL plan baselines. We can easily prove this via the following SELECT showing the execution plan of the previous new PLAN_NAME (notice the absence of the Oracle 12c TABLE ACCESS BY INDEX ROWID BATCHED feature):

[Click here to view code image](#)

```
SQL>select * from  
table(dbms_xplan.display_sql_plan_baseline(plan_name =>  
'SQL_PLAN_d9tch6banyzg98576eb1f'));  
-----  
-----  
SQL handle: SQL_d4e59032d54f7de9  
SQL text: SELECT count(*), max(col2) FROM t1 where flag =
```

```
:"SYS_B_0"
```

```
-----  
-----  
-----  
-----  
Plan name: SQL_PLAN_d9tch6banyzg98576eb1f          Plan id:  
2239163167
```

```
Enabled: YES      Fixed: NO      Accepted: NO      Origin: AUTO-  
CAPTURE  
Plan rows: From dictionary
```

```
-----  
-----  
-----  
-----  
Plan hash value: 3625400295
```

```
| Id  | Operation           | Name | Rows | Bytes | Cost  
(%CPU) | Time     |  
-----  
| 0  | SELECT STATEMENT    |       |       | 1   | 54  
| 2  | (0) | 00:00:01 |  
| 1  | SORT AGGREGATE     |       |       | 1   | 54  
|    |               |  
| 2  | TABLE ACCESS BY INDEX ROWID | T1   |       | 1   | 54  
| 2  | (0) | 00:00:01 |  
|* 3  | INDEX RANGE SCAN     | I1   |       | 1  
|    | 1 (0) | 00:00:01 |
```

```
-----  
-----  
-----  
-----  
Predicate Information (identified by operation id):
```

```
3 - access("FLAG"=:SYS_B_0)
```

Bear in mind as well that if you try to get the plan_hash_2 value (2239163167) of the constrained CBO plan from the V\$SQL_PLAN view, you will not be able to find it, as shown in the following:

[Click here to view code image](#)

```
SQL> SELECT  
      p.sql_id  
      ,p.plan_hash_value  
      ,p.child_number  
      ,t.phv2  
  FROM  
    v$sql_plan p  
  ,xmltable('for $i in /other_xml/info
```

```

        where $i/@type eq "plan_hash_2"
            return $i'
            passing xmltype(p.other_xml)
            columns phv2 number path '/') t
    WHERE p.sql_id = '4sdth4kka4ykw'
    AND   p.other_xml is not null;

```

SQL_ID	PLAN_HASH_VALUE	CHILD_NUMBER	PHV2
4sdth4kka4ykw	497086120	0	2015585387 → 12c ofe execution plan

This limitation occurs because only executed (used) CBO plans are stored in V\$SQL_PLAN. And, as this generated CBO plan has been constrained by an existing SQL plan baseline, it has not been used and therefore has not been inserted into V\$SQL_PLAN dynamic view. If we disable the use of SQL plan baselines, the CBO plan will be used and inserted into V\$SQL_PLAN, as follows:

[Click here to view code image](#)

```
SQL> alter session set optimizer_use_sql_plan_baselines = FALSE;
```

```
SQL>SELECT count(*), max(col2) FROM t1 where flag = 'Y1';
```

COUNT(*)	MAX(COL2)
1	XX

```

SQL> SELECT
      p.sql_id
    ,p.plan_hash_value
    ,p.child_number
    ,t.phv2
  FROM
    v$sql_plan p
  ,xmltable('for $i in /other_xml/info
    where $i/@type eq "plan_hash_2"
        return $i'
        passing xmltype(p.other_xml)
        columns phv2 number path '/') t
  WHERE p.sql_id = '4sdth4kka4ykw'
  AND   p.other_xml is not null;

```

SQL_ID	PLAN_HASH_VALUE	CHILD_NUMBER	PHV2
4sdth4kka4ykw	3625400295	0	2239163167 → new 11g ofe execution plan
4sdth4kka4ykw	497086120	0	2015585387 → 12c ofe execution plan

Now that we have succeeded in producing a CBO plan that doesn't match any of the

existing SQL plan baselines, let's see how the CBO has internally managed this situation via the corresponding 10053 trace file. This file starts by signaling the presence of a plan in the SQL Management Base via the usual warning:

```
SPM: statement found in SMB
```

It then prints the current SQL statement text:

[Click here to view code image](#)

```
*****  
---- Current SQL Statement for this session (sql_id=4sdth4kka4ykw)  
----  
SELECT count(*), max(col2) FROM t1 where flaG = :"SYS_B_0"  
*****
```

The interesting stuff about the nonmatching execution plan generated by the CBO comes from a few lines down in the same trace file:

[Click here to view code image](#)

```
SPM: setup to add new plan to existing plan baseline  
, sig = 15340826253708983785, planId = 2239163167  
SPM: sql stmt=SELECT count(*), max(col2) FROM t1 where flaG =  
:"SYS_B_0"  
SPM: planId's of plan baseline are: 1634389831 2015585387
```

The optimizer is indirectly saying that it came up with a plan having a PLAN_HASH_2 value not matching any of the existing SQL plan baselines; and therefore it has added it to the existing SQL plan baseline for future evaluation and evolution.

The next step the optimizer carries out in this case is reproducing and costing the two accepted and enabled plans. If both plans are reproducible, then the plan with the lower cost will be selected and used to honor the current SQL statement. In this particular case, the optimizer has started by trying to reproduce the full table scan SQL plan baseline with a planId = 1634389831, as follows:

[Click here to view code image](#)

```
SPM: using qksan to reproduce, cost and select accepted plan, sig =  
15340826253708983785  
SPM: plan reproducibility round 1 (plan outline + session OFE)  
  
SPM: using qksan to reproduce accepted plan, planId = 1634389831  
SPM: plan reproducibility - session OFE = 11020003, hinted OFE =  
12010001
```

Notice how the optimizer is emphasizing that the current Optimizer Feature Enabled (OFE) is 11.2.0.3, while the stored SQL plan baseline OFE is 12.1.0.1. Further down the trace file, we read this:

[Click here to view code image](#)

```
Final query after transformations:***** UNPARSED QUERY IS *****
```

```
SELECT /*+ FULL ("T1") */ COUNT(*) "COUNT(*)", MAX("T1"."COL2")
"MAX(COL2)" FROM "C##MHOURI"."T1" "T1" WHERE "T1"."FLAG"=:B1
```

Remember that the current SQL statement doesn't contain any hint. The FULL(t1) hint indicates that the optimizer is trying to reproduce the SPM full table scan using outline hints taken from the underlying SYS.SQLOBJ\$DATA (plus the session OFE 11.2.0.3), which we can see in the same trace file:

[Click here to view code image](#)

```
Outline Data:
/*+
BEGIN_OUTLINE_DATA
IGNORE_OPTIM_EMBEDDED_HINTS
OPTIMIZER_FEATURES_ENABLE('11.2.0.3') --> this is the altered
session OFE
DB_VERSION('12.1.0.1')
ALL_ROWS
OUTLINE_LEAF(@"SEL$1")
FULL(@"SEL$1" "T1@"SEL$1") --> this is the SPM full scan
hint
END_OUTLINE_DATA
*/
```

The optimizer ends this reproducibility step by signaling that it has succeeded in reproducing the SPM full table scan plan and has evaluated its cost to 272.81, as follows:

[Click here to view code image](#)

```
SPM: planId in plan baseline = 1634389831, planId of reproduced
plan = 1634389831
SPM: best cost so far = 272.81, current accepted plan cost = 272.81
```

Since there are two plans in the SQL Management Base the next step for the optimizer is to do the same reproducibility and costing operations for the index range scan plan using outline hints and current session OFE (11.2.0.3) as follows:

[Click here to view code image](#)

```
SPM: plan reproducibility round 1 (plan outline + session OFE)
SPM: using qksan to reproduce accepted plan, planId = 2015585387
```

```
Stmt: ***** UNPARSED QUERY IS *****
SELECT /*+ BATCH_TABLE_ACCESS_BY_ROWID ("T1")
INDEX_RS_ASC ("T1" "I1") */ COUNT(*) "COUNT(*)", MAX("T1"."COL2")
"MAX(COL2)"
FROM "C##MHOURI"."T1" "T1" WHERE "T1"."FLAG"=:B1
```

```
Outline Data:
```

```
/*+
BEGIN_OUTLINE_DATA
IGNORE_OPTIM_EMBEDDED_HINTS
```

```

    OPTIMIZER_FEATURES_ENABLE('11.2.0.3') --> this is the altered
session OFE
    DB_VERSION('12.1.0.1')
    ALL_ROWS
    OUTLINE_LEAF(@"SEL$1")
    INDEX_RS_ASC(@"SEL$1" "T1"@"SEL$1" ("T1"."FLAG")) --> this
the SPM hint
    BATCH_TABLE_ACCESS_BY_ROWID(@"SEL$1" "T1"@"SEL$1") --> this
the SPM hint
    END_OUTLINE_DATA
*/

```

SPM: planId in plan baseline = 2015585387, planId of reproduced
plan = **2015585387**

SPM: best cost so far = 2.00, current accepted plan cost = 2.00

Remember that the current SQL statement doesn't contain any hint. The INDEX_RS_ASC hint is the optimizer trying to reproduce the SPM index range scan using outline hints taken from the underlying SYS.SQLOBJ\$DATA (plus the session OFE 11.2.0.3 taken from the current session).

This is confirmed by looking at the corresponding outline hints stored in the SQL plan baseline underlying table:

[Click here to view code image](#)

```

SQL> select
      extractValue(value(i),'.') outline_hints
  from
      sys.sqlobj$data sqd,
      table(xmlsequence(extract(xmltype(sqd.comp_data),'/outline_'))
  where
      signature = '15285764617405881585';

OUTLINE_HINTS
-----
/*+
   BEGIN_OUTLINE_DATA
   IGNORE_OPTIM_EMBEDDED_HINTS
   OPTIMIZER_FEATURES_ENABLE('12.1.0.1') --> this is OFE at the
SPM capture time
   DB_VERSION('12.1.0.1')
   ALL_ROWS
   OUTLINE_LEAF(@"SEL$1")
   INDEX_RS_ASC(@"SEL$1" "T1"@"SEL$1" ("T1"."FLAG"))
   BATCH_TABLE_ACCESS_BY_ROWID(@"SEL$1" "T1"@"SEL$1")
   END_OUTLINE_DATA
*/

```

Since the optimizer has succeeded in reproducing and costing both of the SQL plan baselines it will decide to use the plan with the lowest cost, as shown in the

corresponding trace file:

[Click here to view code image](#)

```
SPM: statement found in SMB
SPM: re-parsing to generate selected accepted plan, planId =
2015585387
SPM: re-parsed to use selected accepted plan, planId = 2015585387
```

Outline Data:

```
/*+
BEGIN_OUTLINE_DATA
IGNORE_OPTIM_EMBEDDED_HINTS
OPTIMIZER_FEATURES_ENABLE('12.1.0.1')
DB_VERSION('12.1.0.1')
ALL_ROWS
OUTLINE_LEAF(@"SEL$1")
INDEX_RS_ASC(@"SEL$1" "T1"@"SEL$1" ("T1"."FLAG"))
BATCH_TABLE_ACCESS_BY_ROWID(@"SEL$1" "T1"@"SEL$1")
END_OUTLINE_DATA
*/
```

Notice again that when reparsing the selected accepted plan, the CBO used the OFE stored at the SPM plan capture time ('12.1.0.1') and not the current altered session OFE value ('11.2.0.3').

The trace file ends with the following lines:

[Click here to view code image](#)

```
SPM: PHV2 on user parse = 2015585387, PHV2 on recursive parse =
2239163167
SPM: recursive parse succeeded, sig = 15340826253708983785, new
planId = 2239163167
SPM: add new plan: sig = 15340826253708983785, planId = 2239163167
SPM: new plan added to existing plan baseline
, sig = 15340826253708983785, planId = 2239163167
```

At this stage of the investigation, we can say that if the CBO comes up with a plan that has a PLAN_HASH_2 value that does not match an existing SPM planId, it will try to reproduce and cost all existing SQL plan baselines and select the plan having the best cost to run the current SQL statement.

When SQL Plan Baseline Is Not Reproducible

The last point to investigate in order to have a complete understanding of the CBO–SPM interaction is when the optimizer comes up with a plan that doesn't match any of the existing SQL plan baselines, and at the same time, those SQL plan baselines are no longer reproducible. A simple way to trigger this situation is to disable the SPM full table scan plan and rename the index used by the SPM index range scan plan:

[Click here to view code image](#)

```

SQL> alter session set optimizer_use_sql_plan_baselines = TRUE;
-- back to normal OFE
SQL> alter session set optimizer_features_enable='12.1.0.1.1';

SQL> declare
      rs pls_integer;
begin
  -- disabling the full table SPM plan
  rs := dbms_spm.alter_sql_plan_baseline
        (plan_name      =>
 'SQL_PLAN_d9tch6banyzg9616acf47'
          ,attribute_name  => 'ENABLED'
          ,attribute_value  => 'NO'
        );
end;
/

```

```
SQL> alter index i1 rename to rn_i1;
```

If we execute the same query, the CBO will be unable to reproduce the former `i1` index range scan SQL plan baseline because this one has been renamed. Let's verify this:

[Click here to view code image](#)

```
SQL> SELECT count(*), max(col2) FROM t1 where flag = 'Y1';
```

```
-----  
SQL_ID 4sdth4kka4ykw, child number 0  
-----
```

```
Plan hash value: 2491235600  
-----
```

Id	Operation	Name	Rows
Bytes	Cost (%CPU)		
0	SELECT STATEMENT (100)		2
1	SORT AGGREGATE		1
2	TABLE ACCESS BY INDEX ROWID BATCHED	T1	1
3	INDEX RANGE SCAN	RN_I1	1
	1 (0)		

```
Predicate Information (identified by operation id):  
-----
```

```
3 - access ("FLAG"=:SYS_B_0)
```

As expected, the Optimizer has come up with a new execution plan that is not constrained by the two existing SQL plan baselines because one of them (full table scan) has been disabled and the other one has not been reproduced because the index used during the SPM plan capture (`i1`) has been renamed (see the next section for more details). Following is the corresponding 10053 trace file:

[Click here to view code image](#)

```
SPM: setup to add new plan to existing plan baseline  
, sig = 15340826253708983785, planId = 842046253  
SPM: sql stmt=SELECT count(*), max(col2) FROM t1 where flag =  
:"SYS_B_0"
```

The first phrase confirms that the optimizer has come up with a new execution plan having a `PLAN_HASH_2` value (842046253) that does not match any SQL plan baseline `planId`. The new CBO plan is added into the same SQL plan baseline for future evolution. We can immediately check for the presence of this new plan by looking at the `DBA_SQL_PLAN_BASELINES` view:

[Click here to view code image](#)

```
SQL> select  
      a.plan_name  
    , b.plan_id  
    , a.enabled  
    , a.accepted  
  from  
    dba_sql_plan_baselines a  
  , sys.sqlobj$ b  
 where  
    a.signature = b.signature  
  and a.plan_name = b.name  
  and b.plan_id = 842046253;
```

PLAN_NAME	PLAN_ID	ENA	ACC
SQL_PLAN_d9tch6banyzg932309b2d	842046253	YES	NO

The trace file continues by indicating that it found only one accepted and enabled SQL plan baseline that might constrain the plan it has generated, and it is going to reproduce this plan and cost it using both outline hints and the current session OFE:

[Click here to view code image](#)

```
SPM: planId's of plan baseline are: 2015585387  
SPM: using qksan to reproduce, cost and select accepted plan, sig =  
15340826253708983785  
SPM: plan reproducibility round 1(plan outline + session OFE)  
SPM: using qksan to reproduce accepted plan, planId = 2015585387
```

```
SPM: plan reproducibility - session OFE = 99999999, hinted OFE =
12010001
```

Notice the bizarre value of the session OFE (99999999); it might simply indicate that the session OFE is the same as the SPM outline OFE.

Unfortunately, the optimizer failed to reproduce the SPM index range scan (i1) plan and printed the following lines to indicate this failure:

[Click here to view code image](#)

```
SPM: planId in plan baseline = 2015585387, planId of reproduced
plan = 842046253
----- START SPM Plan Dump -----
SPM: failed to reproduce the plan using the following info:
parse_schema name      : C##MHOURI
plan_baseline signature : 15340826253708983785
plan_baseline plan_id   : 2015585387
plan_baseline hintset   :
  hint num 1 len 27 text: IGNORE_OPTIM_EMBEDDED_HINTS
  hint num 2 len 37 text: OPTIMIZER_FEATURES_ENABLE('12.1.0.1')
  hint num 3 len 22 text: DB_VERSION('12.1.0.1')
  hint num 4 len 8 text: ALL_ROWS
  hint num 5 len 22 text: OUTLINE_LEAF(@"SEL$1")
  hint num 6 len 49 text: INDEX_RS_ASC(@"SEL$1" "T1"@"SEL$1"
("T1"."FLAG"))
  hint num 7 len 50 text: BATCH_TABLE_ACCESS_BY_ROWID(@"SEL$1"
"T1"@"SEL$1")
SPM: generated non-matching plan:
```

Having failed to reproduce the SQL plan baseline using both outline hints and session OFE, the CBO will next try reproducing the same plan using the plan outline only, as indicated here:

[Click here to view code image](#)

```
SPM: plan reproducibility round 1 (plan outline only)
SPM: using qksan to reproduce accepted plan, planId = 2015585387
```

Unfortunately, the CBO fails again to reproduce the SQL plan baseline and prints the following corresponding lines to indicate this failure:

[Click here to view code image](#)

```
SPM: planId in plan baseline = 2015585387, planId of reproduced
plan = 842046253
----- START SPM Plan Dump -----
SPM: failed to reproduce the plan using the following info:
parse_schema name      : C##MHOURI
plan_baseline signature : 15340826253708983785
plan_baseline plan_id   : 2015585387
plan_baseline hintset   :
  hint num 1 len 27 text: IGNORE_OPTIM_EMBEDDED_HINTS
  hint num 2 len 37 text: OPTIMIZER_FEATURES_ENABLE('12.1.0.1')
```

```

hint num 3 len 22 text: DB_VERSION('12.1.0.1')
hint num 4 len 8 text: ALL_ROWS
hint num 5 len 22 text: OUTLINE_LEAF(@"SEL$1")
hint num 6 len 49 text: INDEX_RS_ASC(@"SEL$1" "T1"@"SEL$1"
("T1"."FLAG"))
hint num 7 len 50 text: BATCH_TABLE_ACCESS_BY_ROWID(@"SEL$1"
"T1"@"SEL$1")

```

The optimizer tries a second time to reproduce the plan, this time using only a hint for the OFE:

[Click here to view code image](#)

```

SPM: plan reproducibility round 2 (hinted OFE only)
SPM: using qksan to reproduce accepted plan, planId = 2015585387

```

And again, Oracle fails to reproduce the SQL plan baseline in this second round, as the following shows:

[Click here to view code image](#)

```

SPM: planId in plan baseline = 2015585387, planId of reproduced
plan = 3153386212
----- START SPM Plan Dump -----
SPM: failed to reproduce the plan using the following info:
parse_schema name      : C##MHOURI
plan_baseline signature : 15340826253708983785
plan_baseline plan_id   : 2015585387
plan_baseline hintset   :
    hint num 1 len 37 text: OPTIMIZER_FEATURES_ENABLE('12.1.0.1')

```

The 10053 trace file ends by printing the following elegant and clear conclusion which can be easily understood by everyone:

[Click here to view code image](#)

```

SPM: change REPRODUCED status to NO, planName =
SQL_PLAN_d9tch6banyzg97823646b
SPM: REPRODUCED status changes: cntRepro = 1, bitvecRepro = 000
SPM: couldn't reproduce any enabled+accepted plan so using
the cost-based plan, planId = 842046253

```

With these failures, the third conclusion imposes itself: when the plan (PLAN_HASH_2) compiled by the optimizer doesn't match any SQL plan baseline (planId), the CBO will try to reproduce the enabled and accepted plan using two rounds if necessary. During the first round, both outline hints stored in the SYS.SQLOBJ\$DATA and the session OFE are used. If this first round is unsuccessful, the optimizer starts a second round using the session OFE only. If this second round is also unsuccessful, then the CBO uses the cost-based plan it initially came up with.

[Figure 5.1](#) summarizes the CBO–SPM baseline interaction.

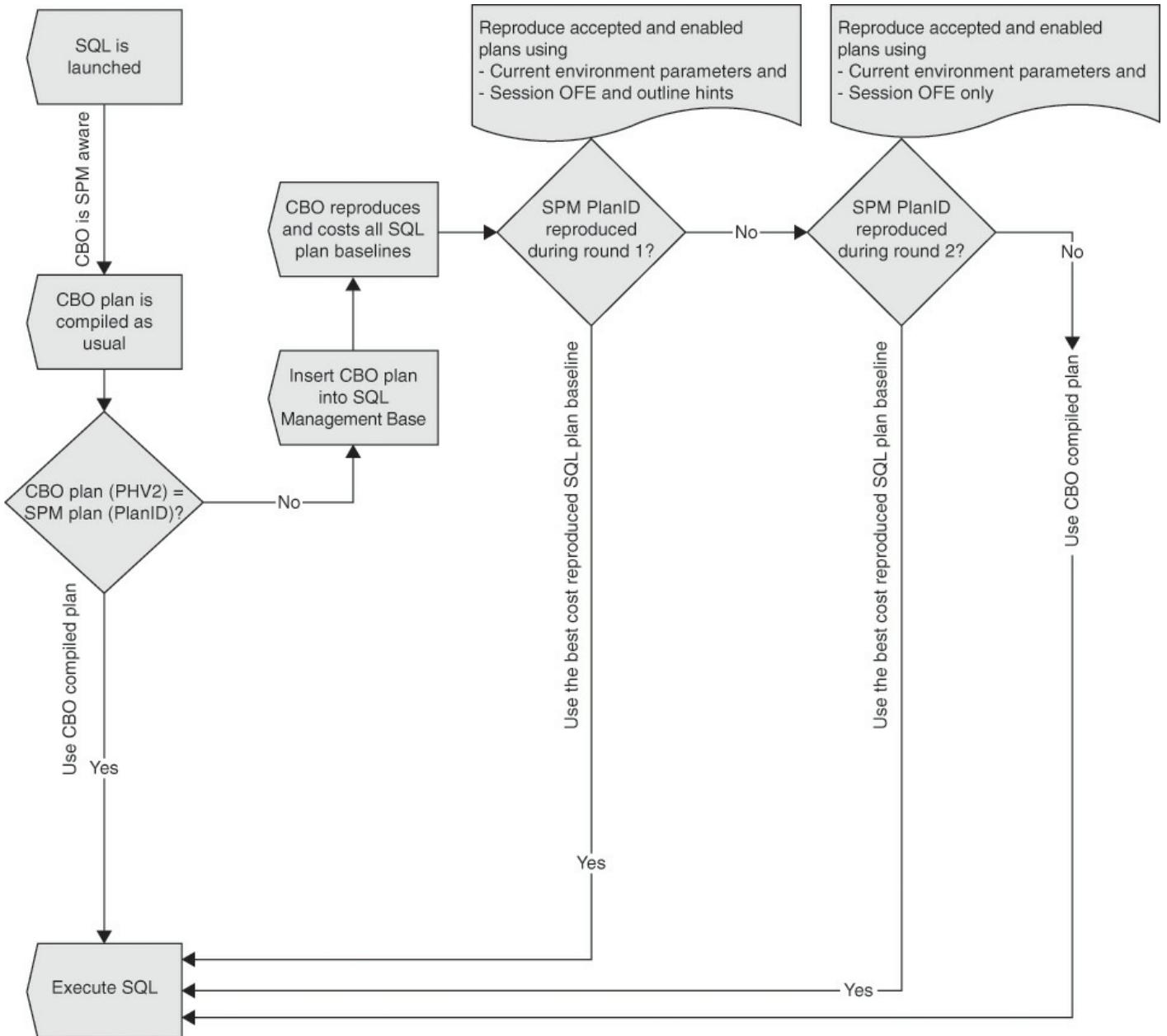


Figure 5.1 CBO and SPM plan selection

In contrast to SQL profiles, SQL plan baselines are designed not only for *stability* (only known and accepted plans are tolerated) but also for plan *flexibility*. Plan flexibility is useful, but it comes nevertheless with an extra time overhead when compared to SQL profile. Because we always want to see if the plan the CBO comes up with is better than the accepted one, Oracle decided to let the CBO carry out its normal optimization task even in the presence of a SQL plan baseline. The actual CBO plan is used only if it matches the SQL plan baseline. When it doesn't match, the CBO plan is nevertheless stored, awaiting evaluation and evolution (Oracle 12c comes up with an automatic evolution). And you must have noticed that to guarantee this plan stability and flexibility, you might incur a parse time overhead. This is particularly true when the CBO comes up with a different plan and there are *multiple* accepted and enabled baselines that compete in terms of cost and might even be unreproducible.

SQL Plan Baseline Reproducibility

We saw in the previous section that when the CBO comes up with a plan that doesn't exist in the SQL Management Base, this plan will be constrained provided the baselined plans are still reproducible. This section explains some of those reasons that might lead to the non-reproducibility of a baselined plan. We'll skip the obvious reason of an absence of the object (e.g., dropping an index) that has been used during the baseline capture and concentrate on cases that are likely to happen in a real-world production application.

Renaming the Index

In the previous section, we saw that renaming the initial index used during the SPM plan capture from `i1` to `rn_i1` made the corresponding SQL plan baseline irreproducible. This situation seems strange because rather than using the index name, Oracle uses the index columns from the outline hints stored in the `SYS.SQLOBJ$DATA` to reproduce the plan. This outline contains the following hint in which the index name is not used:

[Click here to view code image](#)

```
INDEX_RS_ASC(@"SEL$1" "T1"@@"SEL$1" ("T1"."FLAG"))
```

So why, then, is the plan not reproducible in this case? The answer is straightforward if you followed the explanation given in the previous section: a SQL plan baseline is used provided the `PLAN_HASH_2` value of the reproduced plan is equal to the corresponding `planId` stored in the SQL Management Base. And renaming the index (the index name is used in the `PHV2` value generation) breaks this equality, as you can see in the following simple example:

[Click here to view code image](#)

```
SQL> create table t_range
  (
    id          number           not null,
    X           varchar2(30 char) not null,
    D           date,
    C1          number
  )
  partition by range (id)
  (
    partition P_10000 values less than (10000) ,
    partition P_20000 values less than (20000) ,
    partition P_30000 values less than (30000) ,
    partition P_40000 values less than (40000) ,
    partition P_50000 values less than (50000) ,
    partition P_60000 values less than (60000)
  );
```

```
SQL> create index t_r_i1 on t_range(id, c1);
```

```
SQL> select * from t_range where id = 150 and c1 = 42;
```

```

SQL> select * from table(dbms_xplan.display_cursor);

SQL_ID  by2nsk6r62312, child number 0
-----
-----
-----
| Id  | Operation                                | Name      |
Rows  |
-----
|   0 | SELECT                                     |          |
STATEMENT
|   1 |  TABLE ACCESS BY GLOBAL INDEX ROWID BATCHED| T_RANGE
|   1 |
|* 2 |  INDEX RANGE SCAN                         |
T_R_I1 |    1 |
-----
-----

```

Predicate Information (identified by operation id):

```

2 - access("ID"=:SYS_B_0 AND "C1"=:SYS_B_1)

```

Note

- dynamic statistics used: dynamic sampling (level=2)

Notice that the index has been used, and the PHV2 of the corresponding plan follows:

[Click here to view code image](#)

```

SQL> SELECT
      p.sql_id
     ,p.plan_hash_value
     ,p.child_number
     ,t.phv2
  FROM
    v$sql_plan p
   ,xmltable('for $i in /other_xml/info
             where $i/@type eq "plan_hash_2"
             return $i'
            passing xmldocument(p.other_xml)
            columns phv2 number path '/') t
 WHERE p.sql_id = 'by2nsk6r62312'
   AND p.other_xml is not null;

```

SQL_ID	PLAN_HASH_VALUE	CHILD_NUMBER	PHV2
by2nsk6r62312	2479844656	0	2995765617

Suppose that this plan has been captured into a SQL plan baseline and that its PHV2 (2995765617) is the key point for its reproducibility. Let's then change the index

name, re-execute the same query, and check what PHV2 value Oracle will come up with:

[Click here to view code image](#)

Predicate Information (identified by operation id):

2 - access ("ID"=:SYS_B_0 AND "C1"=:SYS_B_1)

Note

- dynamic statistics used: dynamic sampling (level=2)

SQL> SELECT

```
p.sql_id
,p.plan_hash_value
,p.child_number
,t.phv2
FROM
  v$sql_plan p
  ,xmltable('for $i in /other_xml/info
              where $i/@type eq "plan_hash_2"
              return $i'
            passing xmltype(p.other_xml)
            columns phv2 number path '/') t
WHERE p.sql_id = 'by2nsk6r62312'
AND   p.other_xml is not null;
```

SQL_ID PLAN_HASH_VALUE CHILD_NUMBER PHV2

Renaming the index name leads to a new PHV2 value (2863680406), which is not equal to the original one (2995765617). Therefore, renaming an index used in a stored SQL plan baseline (PHV2) will result in the corresponding plan no longer being reproducible.

Changing the Index Type

Changing the type of the index that served during an SPM plan capture might influence the reproducibility of the plan as well. For example, if we change the original index type from a simple B-tree index to a locally prefixed index (an index that does include the partition key) and re-execute the original query, a new PHV2 plan will be generated and hence any SQL plan baseline using this index will cease to be used:

[Click here to view code image](#)

```
SQL> drop index t_r_i2;

SQL> create index t_r_i1 on t_range(id,c1) local;

SQL> select * from t_range where id =150 and c1 = 42;

SQL> select * from table(dbms_xplan.display_cursor);

SQL_ID  by2nsk6r62312, child number 0
-----
-----
--  

| Id  | Operation          | Name   |
Rows  |
-----  

--  

| 0  | SELECT             |        |
STATEMENT  

| 1  | PARTITION RANGE   |        |
SINGLE  

| 2  | TABLE ACCESS BY LOCAL INDEX ROWID BATCHED| T_RANGE
| 1  |
|* 3  | INDEX RANGE SCAN  |        |
T_R_I1  |    1  |
-----  

--  

Predicate Information (identified by operation id):  

-----  

3 - access("ID"=:SYS_B_0 AND "C1"=:SYS_B_1)
```

Note

- dynamic statistics used: dynamic sampling (level=2)

```
SQL> SELECT
      p.sql_id
    ,p.plan_hash_value
    ,p.child_number
    ,t.phv2
  FROM
    v$sql_plan p
   ,xmltable('for $i in /other_xml/info
              where $i/@type eq "plan_hash_2"
              return $i'
             passing xmltype(p.other_xml)
             columns phv2 number path '/') t
 WHERE p.sql_id = 'by2nsk6r62312'
 AND   p.other_xml is not null;
```

SQL_ID	PLAN_HASH_VALUE	CHILD_NUMBER	PHV2
by2nsk6r62312	4134837294	0	794144393

Again, changing the index type from B-tree to local leads to a new PHV2 value (794144393) which is not equal to the original one (2995765617) and makes the SPM plan irreproducible.

Adding Trailing Columns to the Index

If you modify the index used during an SPM plan capture but keep its leading columns unchanged, it is fairly likely that you will not change the PHV2 of the original plan and hence you will still be using the SQL plan baseline, as shown:

[Click here to view code image](#)

```
SQL> drop index t_r_il;

SQL> create index t_r_il on t_range(id,c1, d desc);

SQL> select * from t_range where id =150 and c1 = 42;

SQL> select * from table(dbms_xplan.display_cursor);

SQL_ID  by2nsk6r62312, child number 0
-----
-----|-----|-----|-----|
| Id  | Operation          | Name   |
-----|-----|-----|-----|
| 0   | SELECT STATEMENT   |        |
|     |                   |        |
-----|-----|-----|-----|
```

```

|   1 | TABLE ACCESS BY GLOBAL INDEX ROWID BATCHED| T_RANGE
|   1 |
|* 2 | INDEX RANGE SCAN
T_R_I1 |    1 |
-----
-----
Predicate Information (identified by operation id):
-----
2 - access("ID"=:SYS_B_0 AND "C1"=:SYS_B_1)

Note
-----
- dynamic statistics used: dynamic sampling (level=2)

SQL> SELECT
      p.sql_id
     ,p.plan_hash_value
     ,p.child_number
     ,t.phv2
  FROM
      v$sql_plan p
     ,xmltable('for $i in /other_xml/info
               where $i/@type eq "plan_hash_2"
               return $i'
            passing xmldocument(p.other_xml)
            columns phv2 number path '/') t
 WHERE p.sql_id = 'by2nsk6r62312'
   AND p.other_xml is not null;

SQL_ID          PLAN_HASH_VALUE CHILD_NUMBER        PHV2
-----          -----          -----          -----
by2nsk6r62312      2479844656          0  2995765617

```

As expected, adding a trailing column to an index used by a SQL plan baseline will not preempt this plan from being used by subsequent executions of the corresponding SQL query.

Nevertheless, if you break the order of leading columns in the initial index by adding, for example, a new column at the leading edge of the index, you will likely end up generating a plan with a new PHV2, and therefore the original SQL plan baseline will no longer be reproducible.

Reversing the Index

Very often, we are wrongly advised to reverse a unique index on which we are experiencing severe buffer busy wait events, but hash partitioning is the best solution from both a `SELECT` and `INSERT` performance point of view. If, however, you are using a SQL plan baseline with such an index and you decide to reverse it, you will likely not make your plan irreproducible because reversing the index will not change the

initial PHV2 value:

[Click here to view code image](#)

```
SQL> drop index t_r_i1;

SQL> create index t_r_i1 on t_range(id,c1) reverse;

SQL> select * from t_range where id =150 and c1 = 42;

SQL> select * from table(dbms_xplan.display_cursor);

SQL_ID  by2nsk6r62312, child number 0
-----
select * from t_range where id =:"SYS_B_0" and c1 = :"SYS_B_1"
-----
-----
| Id   | Operation           | Name      |
Rows  |
-----
|    0 | SELECT              |           |
STATEMENT          |           |           |
|    1 | TABLE ACCESS BY GLOBAL INDEX ROWID BATCHED | T_RANGE |
|    1 |           |
|*   2 | INDEX RANGE SCAN   |           |
T_R_I1  |    1 |           |
-----
-----
Predicate Information (identified by operation id):
-----
2 - access("ID"=:SYS_B_0 AND "C1"=:SYS_B_1)

Note
-----
- dynamic statistics used: dynamic sampling (level=2)

SQL> SELECT
      p.sql_id
     ,p.plan_hash_value
     ,p.child_number
     ,t.phv2
  FROM
    v$sql_plan p
   ,xmltable('for $i in /other_xml/info
             where $i/@type eq "plan_hash_2"
             return $i'
            passing xmltype(p.other_xml)
            columns phv2 number path '/') t
 WHERE p.sql_id = 'by2nsk6r62312'
```

```

        AND    p.other_xml is not null;

SQL_ID          PLAN_HASH_VALUE CHILD_NUMBER      PHV2
-----
by2nsk6r62312    2479844656           0  2995765617

```

Although we haven't exhausted all the possible situations that might affect whether a SQL plan is reproducible, this example shows information you need to take into account before changing an object (an index or table) used during the SPM plan capture. You shouldn't be surprised to see your critical report performing badly even when you think you constrained it with a stable and accurate plan. Next we'll check what optimizer parameters are used by the Oracle Optimizer when reproducing a SQL plan baseline.

NLS_SORT and SQL Plan Baseline Reproducibility

When the CBO is due to reproduce an enabled and accepted SQL plan baseline constraining it, there are two possible values of the NLS_SORT parameter that it can use during the compilation phase: the value of NLS_SORT in the compilation session or the value of NLS_SORT stored in the SQL plan baseline at capture time. This section walks through the following reproducible example to investigate this issue:

[Click here to view code image](#)

```

SQL> create table t
      (c1 varchar2(64), c2 char(15), d1 date);

SQL> insert into t
      select
        mod(abs(dbms_random.random),3)+1||chr(ascii('Y'))
      ,dbms_random.string('L',dbms_random.value(1,5))||rownum
      ,to_date(to_char(to_date('01/01/1980','dd/mm/yyyy'),'J') +
        trunc(dbms_random.value(1,11280)),'J')
      from dual
      connect by level <= 1e3;

SQL> alter table t add constraint t_pk primary key (c1,c2);

```

With this setup, we execute a query with the value of the NLS_SORT parameter as BINARY and load its execution plan into a SQL plan baseline automatically:

[Click here to view code image](#)

```

SQL> show parameter nls_sort
NAME                      TYPE        VALUE
-----
nls_sort                  string     BINARY

SQL> alter session set optimizer_capture_sql_plan_baselines=TRUE;
SQL> select
      c1
      from t

```

```

        group by c1
        order by c1 asc nulls last;

SQL> /

SQL> alter session set optimizer_capture_sql_plan_baselines=FALSE;

SQL> select
      c1
    from t
   group by c1
  order by c1 asc nulls last;

SQL> select * from table(dbms_xplan.display_cursor);

-----
SQL_ID  16nxkhb12va0b, child number 1
-----
Plan hash value: 1476560607
-----
| Id  | Operation          | Name | Rows  | Bytes |
-----+
|   0 | SELECT STATEMENT   |       |        |        |
|   1 |  SORT GROUP BY     |       | 1000  | 34000 |
|   2 |   TABLE ACCESS FULL| T    | 1000  | 34000 |
-----+
Note
-----
  - dynamic sampling used for this statement (level=2)
  - SQL plan baseline SQL_PLAN_90sg67694zwyj4b85249e used for this statement
```

We have successfully captured an SPM plan so that when we re-execute the same query under the BINARY nls_sort parameter, the Note of the corresponding execution plan indicates that the CBO plan has been constrained by the captured SQL plan baseline SQL_PLAN_90sg67694zwyj4b85249e.

Let's now change the current session NLS_SORT parameter value (FRENCH) so that it no longer matches the NLS_SORT parameter value (BINARY) used during the SPM plan capture. The corresponding set of hints stored in the underlying sys objects contains the following:

[Click here to view code image](#)

```

SQL> -- if running 11g
SQL> select
      extractValue(value(i),'.') outline_hints
    from
      sys.sqlobj$data sqd,
      table(xmlsequence(extract(xmltype(sqd.comp_data), '/outline_d
where
```

```

signature in (select signature from
              dba_sql_plan_baselines
              where plan_name =
'SQL_PLAN_90sg67694zwyj4b85249e');

OUTLINE_HINTS
-----
FULL(@"SEL$1" "T@"SEL$1")
OUTLINE_LEAF(@"SEL$1")
ALL_ROWS
DB_VERSION('11.2.0.3')
OPTIMIZER_FEATURES_ENABLE('11.2.0.3')
IGNORE_OPTIM_EMBEDDED_HINTS

SQL> -- if running 12c
SQL> select
       substr(extractvalue(value(d), '/hint'), 1, 100) as
outline_hints
  from
    xmltable('/*/outline_data/hint'
  passing (
    select
      xmltype(other_xml) as xmlval
    from
      sys.sqlobj$plan
   where
     signature in (select
                     signature
                   from dba_sql_plan_baselines
                   where plan_name =
'SQL_PLAN_90sg67694zwyj4b85249e'
                 )
    and other_xml is not null
  )
  ) d;

OUTLINE_HINTS
-----
FULL(@"SEL$1" "T@"SEL$1")
OUTLINE_LEAF(@"SEL$1")
ALL_ROWS
DB_VERSION('12.1.0.1')
OPTIMIZER_FEATURES_ENABLE('12.1.0.1.1')
IGNORE_OPTIM_EMBEDDED_HINTS

```

There is no mention of the `NLS_SORT` parameter in this set of hints, which suggests that changing its value will have an effect on the reproducibility of the stored SQL plan baseline. Let's now change the current session `NLS_SORT` parameter value (`FRENCH`) so that it no longer matches the `NLS_SORT` parameter value (`BINARY`) used during the SPM plan capture:

[Click here to view code image](#)

```
SQL> alter session set nls_sort = FRENCH;
SQL> select
      c1
    from t
   group by c1
  order by c1 asc nulls last;

SQL> select * from table(dbms_xplan.display_cursor);

-----
SQL_ID  16nxkhb12va0b, child number 2

An uncaught error happened in prepare_sql_statement: ORA-01403: no
data found

NOTE: cannot fetch plan for SQL_ID: 16nxkhb12va0b, CHILD_NUMBER: 2
      Please verify value of SQL_ID and CHILD_NUMBER;
      It could also be that the plan is no longer in cursor cache
      (check v$sql_plan)
```

This error is an indication that the CBO has come up with a new cost-based execution plan, which it has added to the SQL Management Base, consequently invalidating the cursor we just executed. Running the same query under the same condition a second time shows that the CBO has not been able to reproduce the SQL plan baseline and has decided to use the plan it comes up with:

[Click here to view code image](#)

```
SQL> select
      c1
    from t
   group by c1
  order by c1 asc nulls last;

SQL> select * from
table(dbms_xplan.display_cursor(format=>'advanced')) ;
-----
SQL_ID  16nxkhb12va0b, child number 1
-----
-----| Id  | Operation           | Name | Rows  | Bytes | Cost  (%CPU) |
Time   |           |       |       |       |       |           |
-----| 0  | SELECT STATEMENT    |       |       |       |       |      5
(100) |           |           |       |       |       |           |
| 1  | SORT ORDER BY      |       | 1000 | 34000 |      5  (40) |
00:00:01 |           |           |       |       |       |           |
```

```

| 2 | HASH GROUP BY      |           | 1000 | 34000 |      5 (40) |
00:00:01 |
| 3 | TABLE ACCESS FULL| T      | 1000 | 34000 |      3 (0) |
00:00:01 |
-----
```

Column Projection Information (identified by operation id):

```

1 - (#keys=1) NLSSORT("C1", 'nls_sort='FRENCH') [522],
    "C1" [VARCHAR2, 64]
2 - "C1" [VARCHAR2, 64]
3 - (rowset=200) "C1" [VARCHAR2, 64]
```

Note

- dynamic sampling used for this statement (level=2)

The bottom line: be very careful when changing environment parameters such as NLS_SORT and NLS_LANG that are not part of the outline hints stored in the originally captured plan. With different values of such parameters, you might end up not using your SQL plan baseline.

ALL_ROWS versus FIRST_ROWS

When investigating issues of non-reproducibility of SQL plan baselines, an interesting question might come up: What optimizer mode will be used while reproducing a SQL plan baseline? Is it the optimizer mode stored in the SYS.SQLOBJ\$DATA during SQL baseline capture or the optimizer mode of the current compilation session? If you want to know the answer, look at the following demonstration:

[Click here to view code image](#)

```

SQL> create table t1
  as
    with generator as (
      select --+ materialize
             rownum id
        from dual
       connect by
         level <= 10000)
  select
        rownum                  id,
        trunc(dbms_random.value(1,1000))      n1,
        lpad(rownum,10,'0') small_vc,
        rpad('x',100)      padding
   from
        generator    v1,
        generator    v2
  where
```

```

rownum <= 1000000;

SQL> create index t1_n1 on t1(id, n1);

SQL> create table t2
  as
  with generator as (
    select --+ materialize
      rownum id
    from dual
    connect by
      level <= 10000)
  select
    rownum           id,
    trunc(dbms_random.value(10001,20001))   x1,
    lpad(rownum,10,'0') small_vc,
    rpad('x',100)    padding
  from
    generator v1,
    generator v2
  where
    rownum <= 1000000;

```

```
SQL> create index t2_i1 on t2(x1);
```

```
SQL> create index ind_t1_extra on t1(id);
```

```

SQL> exec dbms_stats.gather_table_stats(user, 't2', method_opt =>
  'for all columns size 1');
SQL> exec dbms_stats.gather_table_stats(user, 't1', method_opt =>
  'for all columns size 1');

```

Having engineered the model (based on Jonathan Lewis's table script), let's run a query and load its corresponding execution plan into a SQL plan baseline using the ALL_ROWS optimizer mode parameter value:

[Click here to view code image](#)

```
SQL> show parameter optimizer_mode
```

NAME	TYPE	VALUE
optimizer_mode	string	ALL_ROWS

```
SQL> alter session set optimizer_capture_sql_plan_baselines=TRUE;
```

```
SQL> select * from t1 where id in (select id from t2 where x1 =
  17335) order by id;
```

```
SQL> /
```

```
SQL> alter session set optimizer_capture_sql_plan_baselines=FALSE;
```

From this point on, this query will be protected against any change in execution plan by a dedicated SQL plan baseline:

[Click here to view code image](#)

```
SQL> select * from t1 where id in (select id from t2 where x1 = 17335) order by id;
```

```
88 rows selected.
```

```
Elapsed: 00:00:00.18
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

Id	Operation	Name	Rows
Bytes			
0	SELECT STATEMENT		
1	NESTED LOOPS		
2	NESTED LOOPS		100
13100			
3	SORT UNIQUE		100
1000			
4	TABLE ACCESS BY INDEX ROWID	T2	100
1000			
5	INDEX RANGE SCAN	T2_I1	100
* 6	INDEX RANGE SCAN	IND_T1_EXTRA	1
7	TABLE ACCESS BY INDEX ROWID	T1	1
121			

```
Predicate Information (identified by operation id):
```

```
-----  
5 - access("X1"=17335)  
6 - access("ID"="ID")
```

Note

```
-----  
- SQL plan baseline SQL_PLAN_bjazgx0cv6xtwff8eddc2 used for this statement
```

The outline hints and session OFE stored in the underlying SPM tables for this

particular plan are as follows:

[Click here to view code image](#)

```
SQL> -- if running 11g
SQL> select
      extractValue(value(i), '.') outline_hints
  from
    sys.sqlobj$data sqd,
    table(xmlsequence(extract(xmltype(sqd.comp_data), '/outline_da
  where
    exists (select null
            from
              dba_sql_plan_baselines dsb
            where dsb.signature = sqd.signature
            and dsb.plan_name =
'SQL_PLAN_bjazgx0cv6xtwff8eddc2'
          );
-----  
OUTLINE_HINTS
-----  
NLJ_BATCHING(@"SEL$5DA710D3" "T1"@"SEL$1")
USE_NL(@"SEL$5DA710D3" "T1"@"SEL$1")
LEADING(@"SEL$5DA710D3" "T2"@"SEL$2" "T1"@"SEL$1")
INDEX(@"SEL$5DA710D3" "T1"@"SEL$1" ("T1"."ID"))
INDEX_RS_ASC(@"SEL$5DA710D3" "T2"@"SEL$2" ("T2"."X1"))
OUTLINE(@"SEL$2")
OUTLINE(@"SEL$1")
UNNEST(@"SEL$2")
OUTLINE_LEAF(@"SEL$5DA710D3")
ALL_ROWS
DB_VERSION('11.2.0.3')
OPTIMIZER_FEATURES_ENABLE('11.2.0.3')
IGNORE_OPTIM_EMBEDDED_HINTS

SQL> -- if running 12c
SQL> select
      substr(extractvalue(value(d), '/hint'), 1, 100) as
outline_hints
  from
    xmltable('/outline_data/hint'
  passing (select
            xmltype(other_xml) as xmlval
          from
            sys.sqlobj$plan
          where
            signature in (select
                            signature
                          from
                            dba_sql_plan_baselines
                          where plan_name =
```

```

'SQL_PLAN_aw0ntx07w1vggff8eddc2'
)
      and other_xml is not null
)
) d;

OUTLINE_HINTS
-----
SEMI_TO_INNER(@"SEL$5DA710D3" "T2"@"SEL$2")
NLJ_BATCHING(@"SEL$5DA710D3" "T1"@"SEL$1")
USE_NL(@"SEL$5DA710D3" "T1"@"SEL$1")
LEADING(@"SEL$5DA710D3" "T2"@"SEL$2" "T1"@"SEL$1")
INDEX(@"SEL$5DA710D3" "T1"@"SEL$1" ("T1"."ID"))
INDEX_RS_ASC(@"SEL$5DA710D3" "T2"@"SEL$2" ("T2"."X1"))
OUTLINE(@"SEL$2")
OUTLINE(@"SEL$1")
UNNEST(@"SEL$2")
OUTLINE_LEAF(@"SEL$5DA710D3")
ALL_ROWS
DB_VERSION('12.1.0.1')
OPTIMIZER_FEATURES_ENABLE('12.1.0.1.1')
IGNORE_OPTIM_EMBEDDED_HINTS

```

Notice particularly the optimizer mode (**ALL_ROWS**) this outline has stored and which the CBO will be normally using when checking the SQL plan baseline reproducibility. We saw in [Figure 5.1](#) that when the CBO compiles a plan that has a PHV2 that doesn't match an existing SPM planId, it will then try to reproduce, cost, and select the best costed SQL plan baseline. Let's then change the optimizer mode and run the same query again:

[Click here to view code image](#)

```
SQL> alter session set optimizer_mode = first_rows;
```

For the sake of simplicity, let's briefly suspend the baseline usage in order to check what execution plan the CBO will come up with under this new altered optimizer mode:

[Click here to view code image](#)

```
SQL> alter session set optimizer_use_sql_plan_baselines = FALSE;
SQL> select * from t1 where id in (select id from t2 where x1 =
17335) order by id;
88 rows selected.
Elapsed: 00:01:54.61
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

```
-----
-- 
| Id  | Operation          | Name   | Rows  | Bytes
|---|---|---|---|---|
```

```

|
-----
---  

|   0 | SELECT  

STATEMENT          |           |           |           |
|   1 | NESTED LOOPS SEMI          |           | 100 | 13100
|
|   2 | TABLE ACCESS BY INDEX ROWID|
T1              | 1000K| 115M|
|   3 | INDEX FULL SCAN          | IND_T1_EXTRA
| 1000K|           |
|*  4 | TABLE ACCESS BY INDEX ROWID| T2          | 1 | 10
|
|*  5 | INDEX RANGE SCAN          | T2_I1       | 100
|           |
-----
```

Predicate Information (identified by operation id):

```

4 - filter("ID"="ID")
5 - access("X1"=17335)
```

Notice that the CBO, under the FIRST_ROWS mode, has preferred to full scan the IND_T1_EXTRA index in order to avoid the order by operation, irrespective of the higher cost of the index full scan, thereby causing a serious performance penalty. This is one of the several bugs the deprecated FIRST_ROWS mode can introduce. It is used here mainly for a pedagogic reason, and you should not use it anymore.

We see now that the CBO will come up with a different plan and, if constrained by an SPM plan, it will try to reproduce this plan. Let's check this situation by setting the OPTIMIZER_USE_SQL_PLAN_BASELINES parameter back to its default value and re-executing the same query:

[Click here to view code image](#)

```

SQL> alter session set optimizer_use_sql_plan_baselines=TRUE;

SQL> select * from t1 where id in (select id from t2 where x1 =
17335) order by id;

SQL> select * from table(dbms_xplan.display_cursor);
```

```

-----
---  

| Id  | Operation          | Name      | Rows  |
Bytes |
-----
|   0 | SELECT             |          |        |
STATEMENT          |           |           |
|   1 | NESTED
```

```

LOOPS
| 2 | NESTED LOOPS | | | 100 |
13100 |
| 3 | SORT UNIQUE | | | 100
| 1000 |
| 4 | TABLE ACCESS BY INDEX ROWID| T2 | | 100
| 1000 |
|* 5 | INDEX RANGE SCAN | T2_I1 | | 100
| |
|* 6 | INDEX RANGE SCAN | IND_T1_EXTRA | | 1
| |
| 7 | TABLE ACCESS BY INDEX ROWID | T1 | | 1
| 121 |
-----
-----

```

Predicate Information (identified by operation id):

```

5 - access("X1"=17335)
6 - access("ID"="ID")

```

Note

- SQL plan baseline SQL_PLAN_bjazgx0cv6xtwff8eddc2 used for this statement

This example clearly demonstrates that, when reproducing a SQL plan baseline, Oracle will use the optimizer mode of the plan outline stored in the underlying SYS.SQLOBJ\$DATA and not the optimizer mode of the current compilation session.

However, bear in mind that, while reproducing SQL plan baselines, the CBO will not always use parameters stored during the SPM plan capture rather than those available at compilation time.

Adaptive Cursor Sharing and SPM

[Chapter 4](#) paved the way to this next section where we investigate how adaptive cursor sharing and SPM collaborate. Assume that we already have two bind-aware cursors present in the cursor cache, allowing the following query to switch from an index range (plan_hash_value 3625400295 flag values 'Y1' and 'Y2') to a full table scan (plan_hash_value 3724264953 flag values 'N1' and 'N2') depending on the selectivity of the bind variable (remember that we are using CURSOR_SHARING = FORCE):

[Click here to view code image](#)

```

SQL> alter session set optimizer_mode=all_rows;

SQL> select count(*), max(col2) from t1 where flag = 'Y1';

SQL> select

```

```

        child_number
, is_bind_aware
, is_bind_sensitive
, to_char(exact_matching_signature) sig
        , plan_hash_value
from v$sql
where sql_id = '6fbvysnhkvugw'
and is_shareable = 'Y';

```

CHILD_NUMBER	IS_BIND_AWARE	IS_BIND_SENSITIVE	SIG	PLAN_HASH_VALUE
1		Y	Y	1534082625370898378!
2		Y	Y	1534082625370898378!

The SQL_ID of the preceding query is 6fbvysnhkvugw. Let's now load both its execution plans into a SQL plan baseline directly from the cursor cache:

[Click here to view code image](#)

```

SQL> declare
      rs pls_integer;
begin
  rs := dbms_spm.load_plans_from_cursor_cache('6fbvysnhkvugw');
end;
/

```

```

SQL> select
      to_char(signature) sig
    , plan_name
    , enabled
    , accepted
  from
    dba_sql_plan_baselines;

```

SIG	PLAN_NAME	ENABLED	ACCEPTED
15340826253708983785	SQL_PLAN_d9tch6banyzg9616acf47	YES	YES ->
15340826253708983785	SQL_PLAN_d9tch6banyzg97823646b	YES	YES ->

ACS and SPM in Oracle 11g Release 11.2.0.3.0

There are two bind-aware child cursors and two SQL plan baselines: one forcing an INDEX RANGE SCAN and the other guaranteeing the FULL TABLE SCAN access path. Executing the same query but switching from a bind variable favoring an index

range scan to a bind variable favoring a full table scan shows a perfect harmony between these two features (ACS and SPM), as shown here:

[Click here to view code image](#)

```
SQL> select * from v$version where rownum =1;

BANNER
-----
-----
Oracle Database 11g Enterprise Edition Release 11.2.0.3.0 - 64bit
Production

SQL> select count(*), max(col2) from t1 where flag = 'Y1';

-----| Id   | Operation           | Name | Rows | Bytes |
-----| 0   | SELECT STATEMENT    |       |       |       |
| 1   |   SORT AGGREGATE    |       |       | 1     | 54   |
| 2   | TABLE ACCESS BY INDEX ROWID| T1   | 18   | 972  |
|* 3   |   INDEX RANGE SCAN   | I1   | 18   |       |
-----
```

Predicate Information (identified by operation id):

```
-----3 - access("FLAG"=:SYS_B_0)
```

Note

```
----- - SQL plan baseline SQL_PLAN_d9tch6banyzg98576eb1f used for this statement
```

```
SQL> select count(*), max(col2) from t1 where flag = 'N1';

-----| Id   | Operation           | Name | Rows | Bytes |
-----| 0   | SELECT STATEMENT    |       |       |       |
| 1   |   SORT AGGREGATE    |       |       | 1     | 54   |
|* 2   |   TABLE ACCESS FULL | T1   | 50505 | 2663K|
-----
```

Predicate Information (identified by operation id):

```
-----2 - filter("FLAG"=:SYS_B_0)
```

Note

```
----- - SQL plan baseline SQL_PLAN_d9tch6banyzg9616acf47 used for this statement
```

If we were to try several executions of the same query alternating between ('Y1', 'Y2') and ('N1', 'N2') bind variable values, each time we would get the desired execution plan from the SQL Management Base—that is, index range scan for the former pair of values and full table scan for the later tandem of values. Here is the content of v\$sql after a warming up period:

[Click here to view code image](#)

```
SQL> select
      sql_id
    , child_number
    , is_bind_aware
    , is_bind_sensitive
    , to_char(exact_matching_signature) sig
    , executions
    , plan_hash_value
  from v$sql
 where sql_id = '6fbvysnhkvugw'
   and is_shareable = 'Y'
 ;
SQL_ID          CHILD_NUMBER I I SIG          EXECUTIONS
PLAN_HASH_VALUE
-----
-----
6fbvysnhkvugw          1 Y Y
15340826253708983785      1        3724264953
6fbvysnhkvugw          2 Y Y
15340826253708983785      1        3625400295
6fbvysnhkvugw          5 Y Y
15340826253708983785      5        3625400295
6fbvysnhkvugw          6 Y Y
15340826253708983785      4        3724264953
```

An interesting thing happens when we disturb this scenario a little bit by creating a new index. This forces the CBO to compile a new execution plan not currently present in the SQL Management Base:

[Click here to view code image](#)

```
SQL> create index i2 on t1(flag,col2);

SQL> select count(*), max(col2) from t1 where flag = 'N2';

SQL> select * from table(dbms_xplan.display_cursor);

-----
SQL_ID  6fbvysnhkvugw, child number 7

An uncaught error happened in prepare_sql_statement: ORA-01403: no
data found
NOTE: cannot fetch plan for SQL_ID: 6fbvysnhkvugw, CHILD_NUMBER: 7
```

Please verify value of SQL_ID and CHILD_NUMBER;
It could also be that the plan is no longer in cursor cache
(check v\$sql_plan)

This error message pops up when a new cost-based plan is added to the SQL plan baseline.

Running the same query again will show that the SQL plan baseline loaded earlier is finally used:

[Click here to view code image](#)

```
SQL_ID  6fbvysnhkvugw, child number 7
-----
-----
| Id  | Operation          | Name | Rows  | Bytes | Cost  (%CPU) |
Time   |
-----
|   0 | SELECT STATEMENT   |       |       |       |       | 273
(100) |
|   1 |   SORT AGGREGATE    |       |       |       |       | 54
|       |                   |
|*  2 |   TABLE ACCESS FULL| T1   | 49749 | 2623K| 273   (1) |
00:00:04 |
-----
```

Predicate Information (identified by operation id):

```
2 - filter("FLAG"=:SYS_B_0)
```

Note

- SQL plan baseline SQL_PLAN_d9tch6banyzg9616acf47 used for this statement

Let's now execute the same query with a bind variable favoring an index range scan (remember that just before creating the I2 index, we were in a stable situation where switching between bind variables produced the desired plan accordingly):

[Click here to view code image](#)

```
SQL> select count(*), max(col2) from t1 where flag = 'Y2';
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

```
SQL_ID  6fbvysnhkvugw, child number 7
-----
-----
| Id  | Operation          | Name | Rows  | Bytes | Cost  (%CPU) |
Time   |
-----
```

```
|   0 | SELECT STATEMENT   |       |       |       |       | 273
(100) |
|   1 |   SORT AGGREGATE    |       |       |       |       | 54
|       |                   |
|*  2 |   TABLE ACCESS FULL| T1   | 49749 | 2623K| 273   (1) |
00:00:04 |
-----
```

Time							
0	SELECT STATEMENT						273
(100)							
1	SORT AGGREGATE			1		54	
*	TABLE ACCESS FULL	T1	49749	2623K	273	(1)	
00:00:04							

Predicate Information (identified by operation id):

2 - filter("FLAG"=:SYS_B_0)

Note

- SQL plan baseline SQL_PLAN_d9tch6banyzg9616acf47 used for this statement

This time the switch to an index range scan plan didn't occur. Maybe after a warming-up period, executing the same query again might produce the desired switch to index range scan plan. Let's try:

[Click here to view code image](#)

```
SQL> select count(*), max(col2) from t1 where flag = 'Y2';
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

```
SQL_ID  6fbvysnhkvugw, child number 7
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	
0	SELECT STATEMENT					273
(100)						
1	SORT AGGREGATE			1	54	
*	TABLE ACCESS FULL	T1	49749	2623K	273	(1)
00:00:04						

Predicate Information (identified by operation id):

```
2 - filter("FLAG"=:SYS_B_0)
```

Note

- SQL plan baseline SQL_PLAN_d9tch6banyzg9616acf47 used for this statement

Unfortunately, even after executing the same query 14 times, the switch to the index range scan plan didn't take place, as the following shows:

[Click here to view code image](#)

```
SQL> select
      sql_id
      ,child_number
      ,is_bind_aware
      ,is_bind_sensitive
      ,to_char(exact_matching_signature) sig
      ,executions
      ,plan_hash_value
  from v$sql
 where sql_id = '6fbvysnhkvugw'
   and is_shareable = 'Y'
 ;
```

SQL_ID	CHILD_NUMBER	I	I	SIG	EXECUTIONS
PLAN_HASH_VALUE					
6fbvysnhkvugw	1	Y	Y		
15340826253708983785	1			3724264953	
6fbvysnhkvugw	2	Y	Y		
15340826253708983785	1			3625400295	
6fbvysnhkvugw	5	N	N		
15340826253708983785	14			3724264953	

Bizarrely, if we disable the use of SQL plan baselines, the switch to the index range scan will immediately occur, as follows:

[Click here to view code image](#)

```
SQL> alter session set optimizer_use_sql_plan_baselines = FALSE;
SQL> select count(*), max(col2) from t1 where flag = 'Y2';
SQL> select * from table(dbms_xplan.display_cursor);
-----  
-----  
| Id | Operation          | Name | Rows | Bytes | Cost  
(%CPU) | Time       |  
-----
```

```

-----
|   0 | SELECT STATEMENT          |       |       |       |       |       2
(100)|       |                               |
|   1 |   SORT AGGREGATE          |       |       1 |      54
|       |       |
|   2 |     TABLE ACCESS BY INDEX ROWID| T1    |      19 |  1026
|   2 |       (0) | 00:00:01 |
|*  3 |     INDEX RANGE SCAN        | I1    |      19
|       |       1   (0) | 00:00:01 |
-----
-----
```

Predicate Information (identified by operation id):

```
3 - access("FLAG"=:SYS_B_0)
```

And bizarrely again, on allowing the use of baselines, you will be back to the earlier situation of not using the desirable index range scan plan:

[Click here to view code image](#)

```

SQL> alter session set optimizer_use_sql_plan_baselines = TRUE;
SQL> select count(*), max(col2) from t1 where flag = 'Y2';
SQL> select * from table(dbms_xplan.display_cursor);
```

```

-----
| Id  | Operation           | Name | Rows  | Bytes | Cost (%CPU) |
Time |                               |
-----
|   0 | SELECT STATEMENT    |       |       |       |      273
(100)|       |                               |
|   1 |   SORT AGGREGATE    |       |       1 |      54
|       |       |
|*  2 |   TABLE ACCESS FULL| T1    | 49749 | 2623K|      273  (1) |
00:00:04 |
-----
```

Predicate Information (identified by operation id):

```
2 - filter("FLAG"=:SYS_B_0)
```

Note

```
- SQL plan baseline SQL_PLAN_d9tch6banyzg9616acf47 used for this
statement
```

```

SQL> select
      sql_id
    ,child_number
  ,is_bind_aware
  ,is_bind_sensitive
  ,to_char(exact_matching_signature) sig
  ,executions
  ,plan_hash_value
from v$sql
where sql_id = '6fbvysnhkvugw'
and is_shareable = 'Y'
;

SQL_ID          CHILD_NUMBER I I SIG          EXECUTIONS
PLAN_HASH_VALUE
-----
-----
6fbvysnhkvugw           1 N Y
15340826253708983785     1        3625400295
6fbvysnhkvugw           5 N N
15340826253708983785     15       3724264953

```

The bottom line is that when mixing an SPM plan with ACS, you might end up with the ACS feature not working as expected. This is particularly true when the CBO comes up with a new plan and adds it to the SQL Management Base. If you want ACS to work correctly in this case, you need to disable the SQL baselines or load and accept the new cost-based plan into the SQL plan baseline. The next section shows that this unstable collaboration has been fixed in the new Oracle Database 12c release.

ACS and SPM in Oracle Database 12c Release 12.1.0.1.0

In order to extend the same investigation to the new Oracle Database 12c release, let's re-establish the scenario. Again, suppose we have two bind-aware child cursors with two SQL plan baselines for the same signature. Executing the same query with two different bind variables will not only switch plans accordingly but also will collaborate very well with the presence of the earlier SQL plan baselines, as shown here:

[Click here to view code image](#)

```

SQL> select * from v$version where rownum =1;

BANNER
-----
-----
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit
Production      0

SQL> select count(*), max(col2) from t1 where flag = 'Y1';

```

Id	Operation				Name	Rows	Bytes
0	SELECT STATEMENT						
1	SORT AGGREGATE					1	54
2	TABLE ACCESS BY INDEX ROWID BATCHED	T1				1	54
* 3	INDEX RANGE SCAN		I1			1	

Predicate Information (identified by operation id):

3 - access("FLAG"=:SYS_B_0)

Note

- SQL plan baseline SQL_PLAN_d9tch6banyzg97823646b used for this statement

SQL> select count(*), max(col2) from t1 where flag = 'N1';

SQL> select * from table(dbms_xplan.display_cursor);

Id	Operation		Name	Rows	Bytes	Cost (%CPU)	
Time							
0	SELECT STATEMENT					273	
(100)							
1	SORT AGGREGATE			1	54		
* 2	TABLE ACCESS FULL	T1		49874	2630K	273	(1)
00:00:01							

Predicate Information (identified by operation id):

2 - filter("FLAG"=:SYS_B_0)

Note

- SQL plan baseline SQL_PLAN_d9tch6banyzg9616acf47 used for this statement

Let's now disturb this situation by creating an extra index, as we did earlier, and check whether the Oracle Database 11g symptoms appear in Oracle Database 12c as well:

[Click here to view code image](#)

```
SQL> create index i2 on t1(flag,col2);

SQL> select count(*), max(col2) from t1 where flag = 'N2';

SQL> select * from table(dbms_xplan.display_cursor);
-----
-----  
| Id  | Operation           | Name | Rows | Bytes | Cost (%CPU) |
Time   |  
-----  
|    0 | SELECT STATEMENT    |      |      |       | 273  
(100)|           |  
|    1 | SORT AGGREGATE     |      |      |       1 | 54  
|*   2 | TABLE ACCESS FULL | T1   | 49874 | 2630K| 273  (1) |
00:00:01 |  
-----  
-----
```

Predicate Information (identified by operation id):

```
2 - filter("FLAG"=:SYS_B_0)
```

Note

- SQL plan baseline SQL_PLAN_d9tch6banyzg9616acf47 used for this statement

As expected, the full table scan plan has been used thanks to the corresponding stored SQL plan baseline. Let's now use the bind variable favoring an index range scan and check whether or not a switch to the index range scan plan will occur:

[Click here to view code image](#)

```
SQL> select count(*), max(col2) from t1 where flag = 'Y2';
-----
-----  
| Id  | Operation           | Name | Rows | Bytes | Cost (%CPU) |
Time   |  
-----  
|    0 | SELECT STATEMENT    |      |      |       | 273  
-----  
|
```

```

(100) |
|   1 | SORT AGGREGATE      |           |   1 |      54
|       |                   |
|* 2 | TABLE ACCESS FULL| T1    | 49874 | 2630K| 273  (1)
00:00:01 |
-----
-----
```

Predicate Information (identified by operation id):

```

2 - filter("FLAG"=:SYS_B_0)
```

Note

```

- SQL plan baseline SQL_PLAN_d9tch6banyzg9616acf47 used for this
statement
```

The switch to the index range scan plan didn't occur, which is reasonable because ACS needs a warming-up period or a certain number of executions at the adjacent BUCKET_ID (see [Chapter 4](#)). Let's re-execute the query and check whether the plan switches after the warming-up period:

[Click here to view code image](#)

```

SQL> select count(*), max(col2) from t1 where flag = 'Y2';

-----
-----
```

Id	Operation	Name	Rows	Bytes
Cost (%CPU)				
0	SELECT STATEMENT		2 (100)	
1	SORT AGGREGATE		1 54	
2	TABLE ACCESS BY INDEX ROWID BATCHED	T1	1 54	
2	(0)			
* 3	INDEX RANGE SCAN	I1	1	
	1 (0)			

```

-----
```

Predicate Information (identified by operation id):

```

3 - access("FLAG"=:SYS_B_0)
```

Note

```

- SQL plan baseline SQL_PLAN_d9tch6banyzg97823646b used for this
statement
```

And the plan switch happens! This is because Oracle 12c has dramatically enhanced the collaboration between ACS and SPM. If we rerun the same query using a bind variable favoring a table full scan, the switch (in contrast to the same experiment done in Oracle 11g earlier) to the full table scan will happen immediately, as the following shows:

[Click here to view code image](#)

```
SQL> select count(*), max(col2) from t1 where flag = 'N2';

-----
| Id  | Operation          | Name | Rows | Bytes | Cost (%CPU) |
Time   |
-----
| 0  | SELECT STATEMENT |      |      |       | 273
(100) |
| 1  |   SORT AGGREGATE |      |      |       1 |      54
|      |                   |
|* 2 | TABLE ACCESS FULL| T1  | 49874 | 2630K| 273 (1) |
00:00:01 |

-----
Predicate Information (identified by operation id):
-----
2 - filter("FLAG"=:SYS_B_0)

Note
-----
- SQL plan baseline SQL_PLAN_d9tch6banyzg9616acf47 used for this
statement
```

When trying this in the Oracle 11g investigation, we could have executed the same query multiple times without being able to invoke a plan switch. In Oracle 12c, we executed the same query with the index range scan bind variable the same number of times as the number of executions done with the full table scan bind variable (two executions) to see the ACS kick in:

[Click here to view code image](#)

```
SQL> select
      sql_id
    , child_number
    , is_bind_aware
    , is_bind_sensitive
    , to_char(exact_matching_signature) sig
    , executions
    , plan_hash_value
  from v$sql
```

```
where sql_id = '6fbvysnhkvugw'
and is_shareable = 'Y';
```

SQL_ID	CHILD_NUMBER	I	I	SIG	EXECUTIONS
PLAN_HASH_VALUE					
6fbvysnhkvugw	1	Y	Y		
15340826253708983785	1			3724264953	
6fbvysnhkvugw	2	Y	Y		
15340826253708983785	2			497086120	
6fbvysnhkvugw	4	Y	Y		
15340826253708983785	1			497086120	
6fbvysnhkvugw	6	Y	Y		
15340826253708983785	1			3724264953	

Summary

You learned from this chapter that starting with Oracle Database 11g, you have a feature called SQL Plan Management (SPM) at your disposal, which allows you to freeze the “good” plans to stabilize your report execution time. This feature not only prevents your critical SQL report from flip-flopping on the plan and execution time but it also has flexibility and evolution. This is possible because the Oracle Optimizer can still generate the best execution plans and store them in the SQL Management Base. These plans can be evolved and accepted later at an appropriate time. As we saw, this is only possible provided the plan stored in the SQL Management Base is still reproducible at the report execution time.

We also spent considerable time exploring some of the reasons that a SQL plan baseline can become irreproducible. This chapter did not present all of the possible scenarios that might lead to an irreproducible SQL plan baseline, but it certainly opened a door for further possible investigations of how the CBO manages to reproduce a SQL plan baseline using the PLAN_HASH_2 value and planId stored in the underlying SYS.SQLOBJ\$ table. In particular, we looked at the parse time penalty a constrained CBO might introduce when it is due to reproduce and cost all the enabled and acceptable plans, particularly when these plans have been made irreproducible, thereby driving the CBO through two fruitless reproducibility rounds.

The last part of this chapter demonstrated how SPM behavior deviates in the presence of a cursor subject to the extended cursor sharing layer; DBAs should always be cautious when mixing the two features in a running system. Finally, we learned that the new Oracle Database 12c release provides an enhancement that makes ACS and SPM work in a perfect harmony.

6. DDL Optimization Tips, Techniques, and Tricks

Prior to Oracle Database 11g, adding a column with a default value to a real-life production table was painful, particularly for a huge table where a substantial number of records would be enriched with the new column and physically updated with its default value. Such a table modification was far from being a simple operation. Fortunately, starting with Oracle Database 11g, a new optimization technique for improving the performance of data definition language (DDL) operations has been implemented. Adding a non-null column with a default value to an existing table can be done almost instantaneously. This chapter explains how Oracle has internally transformed such a painful table modification into a simple metadata operation. It also shows how to check the optimizer's cardinality estimate for such added columns and the ability of such columns to serve as virtual columns and to be used in a column group extension. The chapter investigates the performance penalty this new column might introduce when it is selected from a table. It also explores the effect of this optimizing technique against the creation of an index and, finally, examines the effect of the Oracle Database 12c extension of this technique to nullable columns as well.

DLL Optimization Concept

Consider the following table having 3 million rows:

[Click here to view code image](#)

```
SQL> create table t1
      as select
          rownum n1
        , trunc ((rownum-1)/3) n2
        , trunc(dbms_random.value(rownum, rounum*10)) n3
        , dbms_random.string('U', 10) c1
     from dual
connect by level <= 3e6;
```

```
SQL> desc t1
      Name           Null? Type
----- -----
 1   N1             NUMBER
 2   N2             NUMBER
 3   N3             NUMBER
 4   C1             VARCHAR2(4000 CHAR)
```

The following SQL statement adds an extra non-null column with a default value of 42 to this table:

[Click here to view code image](#)

```
SQL> alter table t1 add C_DDL number DEFAULT 42 NOT NULL;
```

The two crucial keywords **DEFAULT** and **NOT NULL** represent the keys driving this new feature.

To appreciate the difference in the execution time of the preceding `alter table` command, let's execute it in two different Oracle Database releases: 10.2.0.4.0 and 11.2.0.3.0:

[Click here to view code image](#)

```
10.2.0.4.0> alter table t1 add C_DDL number default 42 not null;
```

Table altered.

Elapsed: 00:00:48.53

```
11.2.0.3.0> alter table t1 add C_DDL number default 42 not null;
```

Table altered.

Elapsed: 00:00:00.04

Notice the difference in the execution times. The `C_DDL` column was added instantaneously in Oracle Database 11g Release 2, whereas it took almost 49 seconds in Oracle Database 10g Release 2. What is this new mechanism that allows such a rapid execution time when adding a `NOT NULL` column with `DEFAULT` value to an existing table? How could 3 million rows be updated in just 4 milliseconds?

Let's verify that the update has really been carried out (from here on, this chapter will refer to Oracle Database version 11.0.2.3 unless specified otherwise):

[Click here to view code image](#)

```
SQL> select count(1) from t1;
```

```
-----  
COUNT (1)
```

```
3000000
```

```
SQL> select count(1) from t1 where c_ddl = 42;
```

```
-----  
COUNT (1)
```

```
3000000
```

Although Oracle altered the table `T1` instantaneously, the query shows that all the `C_DDL` column values have been updated with a default value set to 42. How is this possible? Will the following execution plan be of any help here?

[Click here to view code image](#)

```
SQL> select * from table(dbms_xplan.display_cursor);
```

Id	Operation	Name	Rows	Bytes	Cost	(%CPU)
Time						
<hr/>						
0	SELECT STATEMENT					3736
(100)						
1	SORT AGGREGATE			1	13	
* 2	TABLE ACCESS FULL	T1	2712K	33M	3736	(1)
00:00:45						

Predicate Information (identified by operation id):

2 - **filter(NVL("C_DDL",42)=42)**

Note

- dynamic sampling used for this statement (level=2)

Notice the presence of the NVL function in the predicate part of this execution plan. This optimization technique does not tell the whole story. Behind the scenes, Oracle is still considering the C_DDL column as NULLABLE (which means it has not been updated), and consequently, Oracle is protecting it from the null threat by replacing it with its DEFAULT value 42.

Let's have a look at the execution plan in the earlier release and locate the difference:

[Click here to view code image](#)

10.2.0.4.0 > select count(1) from t1 where c_ddl = 42;

COUNT(1)

3000000

10.2.0.4.0> select * from table(dbms_xplan.display_cursor);

Id	Operation	Name	Rows	Bytes	Cost	(%CPU)
Time						
<hr/>						
0	SELECT STATEMENT					4001
(100)						
1	SORT AGGREGATE		1	3		

```
|* 2 | TABLE ACCESS FULL| T1 | 3000K| 8789K| 4001 (8) |
00:00:09 |
```

Predicate Information (identified by operation id):

```
2 - filter("C_DDL"=42)
```

The absence of the NVL function in the predicate part together with the long time (00:00:48.53 sec) it took to add the column in Oracle Database 10g Release 2 explain the working concept introduced in Oracle Database 11g Release 1 to optimize the addition of a non-null column with a default value to an existing table.

Another clue to what is happening in this technique in Oracle Database 11.2.0.3 can be found by comparing the table size before and after the ALTER TABLE statement:

[Click here to view code image](#)

```
SQL> SELECT
      segment_name
    , bytes/1024/1024 mb
  FROM
    dba_segments
 WHERE
    segment_type = 'TABLE'
  AND segment_name = 'T1';
```

SEGMENT_NAME	MB
T1	112

```
SQL> alter table t1 add C_DDL number default 42 not null;
```

```
SQL> SELECT
      segment_name
    , bytes/1024/1024 mb
  FROM
    dba_segments
 WHERE
    segment_type = 'TABLE'
  AND segment_name = 'T1';
```

SEGMENT_NAME	MB
T1	112

Even after the addition of a column, the table size remains unchanged, indicating that the table has not been physically updated.

Simply put, in Oracle Database 11g Release 1 and later, when you add a non-null column with a default value, Oracle will not update all existing rows with this default

value. Instead, Oracle will store metadata for this new column (NOT NULL constraint and DEFAULT value) and allow the table to be modified almost instantaneously irrespective of the size of the altered table. Of course, this is possible at the cost of adding an NVL function when retrieving the added column from a table.

The DDL Optimization Mechanism

Now that we understand the wonderful concept of DDL optimization, let's explore further how Oracle manages this feature to ensure rapidity of DDL while still guaranteeing the correctness of results during data retrieval. This section investigates whether the Oracle Optimizer can accurately estimate the cardinality of such columns and whether this altered column can be used in a column group extension and/or serve as a virtual column.

Table Cardinality Estimation

It's fairly well known that if the Oracle Optimizer can estimate cardinality accurately, it will almost always generate an optimal execution plan. Hence, it is worth investigating whether the selectivity of a column added via this new technique can be accurately estimated using the following instructions:

[Click here to view code image](#)

```
SQL> exec dbms_stats.gather_table_stats(user , 't1', method_opt =>
  'for all columns size 1' , no_
  invalidate => false);

SQL> select /*+ gather_plan_statistics */ count(1) from t1 where
C_DDL = 42;

COUNT(1)
-----
3000000

SQL> select * from
table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));

-----
| Id  | Operation          | Name   | Starts | E-Rows | A-Rows | A-
Time   |
-----
|    0 | SELECT STATEMENT  |        |        |       1 |        |       1
|00:00:00.60 |
|    1 | SORT AGGREGATE    |        |        |       1 |       1 |       1
|00:00:00.60 |
|*   2 | TABLE ACCESS FULL | T1    |        |      1 |
| 3000K| 3000K|00:00:00.45 |
-----
```

Predicate Information (identified by operation id):

2 - filter(NVL("C_DDL",42)=42)

The Oracle optimizer does a perfect single table cardinality estimation when using the C_DDL column as Starts*E-Rows = A-Rows. This was, by the way, expected given the available statistics, as shown in the following:

[Click here to view code image](#)

```
SQL> select
      ut.num_rows*us.density as card
    from
      user_tables ut
    ,user_tab_col_statistics us
   where
      ut.table_name  = us.table_name
    and ut.table_name  = 'T1'
    and us.column_name = 'C_DDL';
```

CARD

3000000

Notice that the cost-based optimizer (CBO) is clever enough to distinguish between a function applied to a column behind the scenes and a function applied by the end user directly into the query. In other words, we all know that the selectivity calculated by the CBO on a predicate of the form function (column)= constant is 1 percent of the number of rows in the table (30,000 in this case). However, the CBO got a perfect estimation (3,000,000) despite the hidden NVL function applied by Oracle while retrieving values in the NOT NULL C_DDL column.

Let's now disturb the uniformly distributed data in C_DDL column a little, gather statistics without a histogram, display the new C_DDL distribution, and check whether the Oracle Optimizer is still able to get an accurate estimate:

[Click here to view code image](#)

```
SQL> update t1
  set c_ddl = 25
  where
    mod(n1,100) = 0;

30000 rows updated.

SQL> commit;

SQL> begin
  dbms_stats.gather_table_stats(user , 't1'
                                , method_opt => 'for all columns size 1'
```

```
        , no_invalidate => false);  
end;  
/  
/
```

```
SQL> select  
      c_ddl  
      ,count(1)  
    from t1  
   group by  
      c_ddl;  
  
C_DDL  COUNT(1)  
-----  
42      2970000  
25      30000
```

The update skews the C_DDL column to check the cardinality estimate of the optimizer when the column C_DDL is used in the WHERE clause predicate as follows:

[Click here to view code image](#)

```
SQL> select /*+ gather_plan_statistics */ count(1) from t1 where  
C_DDL = 42;  
  
COUNT(1)  
-----  
2970000  
  
SQL> select * from  
table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));  
  
-----  
| Id  | Operation          | Name | Starts | E-Rows | A-Rows | A-  
Time  |  
-----  
| 0  | SELECT STATEMENT  |       |       | 1     |       | 1  
|00:00:00.40 |  
| 1  | SORT AGGREGATE   |       |       | 1     |       | 1  
|00:00:00.40 |  
|* 2  | TABLE ACCESS FULL| T1   |       | 1  
| 1500K| 2970K|00:00:00.50 |  
-----  
-----  
Predicate Information (identified by operation id):  
-----  
 2 - filter(NVL("C_DDL",42)=42)
```

The optimizer is no longer doing a perfect estimation ($E\text{-}Rows * Starts \neq A\text{-}Rows$), as shown via the operation in line 2. However, this incorrect estimation is not because

of the added column. It is common behavior for all columns with an uneven distribution resulting in a large skew.

C_DDL Column in a Virtual Column

The issue of inaccurate cardinality estimates can be resolved by collecting a histogram for the C_DDL column so that the Oracle CBO is aware of nonuniform distribution. However, in such a situation, it's often better to create a virtual column instead of collecting a histogram. The C_DDL column can be used as a virtual column and save us from the wrong cardinality estimation:

[Click here to view code image](#)

```
SQL> alter table t1 add c_ddl_virt number
      generated always as (case when c_ddl = 42 then c_ddl else null
end)
      virtual;

SQL> exec dbms_stats.gather_table_stats (user, 't1', method_opt =>
'for columns c_ddl_virt size 1', no_invalidate => false);
SQL> select /*+ gather_plan_statistics */ count(1) from t1 where
c_ddl_virt = 42;

          COUNT(1)
-----
2970000

SQL> select * from
table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
-----
-----| Id   | Operation           | Name | Starts | E-Rows | A-Rows | A-
Time   |
-----| 0   | SELECT STATEMENT    |       |        | 1     |        | 1
|00:00:00.85 |
| 1   | SORT AGGREGATE     |       |        | 1     |        | 1
|00:00:00.85 |
|*  2   | TABLE ACCESS FULL | T1   |        | 1
| 2970K| 2970K|00:00:00.70 |

-----| Predicate Information (identified by operation id): |
-----| 2 - filter(CASE NVL("C_DDL",42) WHEN 42 THEN NVL("C_DDL",42)
ELSE NULL
END =42)
-----
```

Thanks to the virtual column on the top of the C_DDL column, the CBO now gets a

perfect estimate and hence an optimal execution plan. Notice in the predicate part that the NVL function is wrapped around the C_DDL column. Refer to the definition of the virtual column, and you will find that it doesn't use the NVL function in its definition. It is Oracle that recognizes the C_DDL column through its internal metadata that protects it against null values.

You might also have noticed that to take advantage of the virtual column, the original query was modified and the C_DDL column was replaced by the C_DDL_VIRT column. Several Oracle experts have written that the virtual column has an advantage over a function-based index because it doesn't require a rewrite of the original query in order for the CBO to be able to use it at least for its estimation. However, this is true only for virtual columns defined using the SQL TRUNC function. Had the original query gone unchanged, the Oracle optimizer would have made a wrong cardinality estimate, as shown in the following corresponding execution plan:

[Click here to view code image](#)

```
SQL> select /*+ gather_plan_statistics */ count(1) from t1 where
C_DDL = 42;

      COUNT (1)
-----
      2970000
-----

| Id  | Operation          | Name | Starts | E-Rows | A-Rows | A-
Time |
-----
|   0 | SELECT STATEMENT  |       |        |    1   |        |    1
|00:00:00.41 |
|   1 | SORT AGGREGATE    |       |        |    1   |        |    1
|00:00:00.41 |
|*  2 | TABLE ACCESS FULL | T1   |        |    1   |
| 1500K| 2970K|00:00:00.51 |

-----
```

Predicate Information (identified by operation id):

```
2 - filter(NVL("C_DDL",42)=42)
```

You can see that the CBO could not detect the presence of the virtual column and hence could not use it in its cardinality estimation.

C_DDL Column in a Column Group Extension

One very important and grossly underused technique for enhancing the optimizer estimation is to use a column group extension. Although the CBO can accurately estimate the cardinality of a single column predicate, it cannot spot a correlation between two

columns used in an equality predicate. Let's investigate whether the C_DDL column, when used in a conjunction predicate with another normal column, presents any anomalies:

[Click here to view code image](#)

```
SQL> select /*+ gather_plan_statistics */ count(1)
      from t1
     where C_DDL = 42
       and n2      = 22;

          COUNT(1)
-----
          3

SQL> select * from
table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));

-----  
| Id  | Operation           | Name | Starts | E-Rows | A-Rows | A-
Time   |
-----  
|    0 | SELECT STATEMENT   |       |        1 |        |        1
|00:00:00.15 |
|    1 | SORT AGGREGATE     |       |        1 |        1 |        1
|00:00:00.15 |
|*   2 | TABLE ACCESS FULL | T1   |       1 |        1 |        3
|00:00:00.15 |

-----  
Predicate Information (identified by operation id):
-----  
2 - filter(("N2"=22 AND NVL("C_DDL",42)=42))
```

Notice that the cardinality estimation at operation 2 is wrong. A cardinality that equals 1 is always a symptom of an incorrect CBO estimation.

Let's then create a column group extension to help the CBO in its cardinality estimation task:

[Click here to view code image](#)

```
SQL> SELECT
      dbms_stats.create_extended_stats
      (ownname=>user
      ,tabname=>'t1'
      ,extension=>'(c_ddl,n2)')
  FROM dual;

DBMS_STATS.CREATE_EXTENDED_STATS(OWNNAME)
```

```
-----  
SYS_STU5FUD4#KCC03#_TZNTSTN5AV
```

As you might have already guessed, creating a column group extension results in the creation of the virtual column SYS_STU5FUD4#KCC03#_TZNTSTN5AV, on which we can gather statistics without a histogram and issue again the same query with the two initial conjunction predicates:

[Click here to view code image](#)

```
SQL> begin  
      dbms_stats.gather_table_stats  
        (user  
         , 't1'  
         , method_opt      => 'for columns  
SYS_STU5FUD4#KCC03#_TZNTSTN5AV size 1'  
          , cascade        => true  
          , no_invalidate  => false  
        );  
    end;  
/  
  
SQL> select /*+ gather_plan_statistics */ count(1)  
  from t1  
 where C_DDL = 42  
   and n2     = 22;  
  
COUNT(1)  
-----  
3  
  
SQL> select * from  
table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));  
  
-----  
-----  
| Id  | Operation           | Name | Starts | E-Rows | A-Rows | A-  
Time  |  
-----  
| 0  | SELECT STATEMENT    |       |       | 1 |       | 1  
|00:00:00.14 |  
| 1  | SORT AGGREGATE     |       |       | 1 |       | 1  
|00:00:00.14 |  
|* 2  | TABLE ACCESS FULL| T1    |       | 1 |       | 3  
|00:00:00.14 |  
-----  
-----  
  
Predicate Information (identified by operation id):  
-----
```

```
2 - filter(("N2"=22 AND NVL("C_DDL", 42)=42))
```

The optimizer, with the help of the new extended column group, now can do a perfect estimation. You might be wondering how this perfect estimation results from the presence of the column group extension. There are two ways to back this assumption. The first way is taken directly from the available statistics, and the second way is taken from the corresponding 10053 trace file, as the following shows:

[Click here to view code image](#)

```
SQL> select
      round (ut.num_rows*us.density) as card
    from
      user_tables ut
    ,user_tab_col_statistics us
   where
     ut.table_name = us.table_name
   and ut.table_name = 'T1'
   and us.column_name = 'SYS_STU5FUD4#KCC03#_TZNTSTN5AV';

CARD
-----
3

Access path analysis for T1
*****
SINGLE TABLE ACCESS PATH
  Single Table Cardinality Estimation for T1[T1]
  SPD: Return code in qosdDSDirSetup: NODIR, estType = TABLE
  Column (#5): C_DDL(NUMBER)
    AvgLen: 3 NDV: 2 Nulls: 0 Density: 0.000000 Min: 0.000000 Max:
  25.000000
  Column (#2): N2(NUMBER)
    AvgLen: 5 NDV: 1000000 Nulls: 0 Density: 0.000000 Min: 0.000000
  Max: 17.000000
  Column (#6): SYS_STU5FUD4#KCC03#_TZNTSTN5AV(NUMBER)
    AvgLen: 12 NDV: 1030000 Nulls: 0 Density: 0.000000
  ColGroup (#1, VC) SYS_STU5FUD4#KCC03#_TZNTSTN5AV
    Col#: 2 5 CorStregth: 1.94
  ColGroup Usage:: PredCnt: 2 Matches Full: #1 Partial: Sel:
  0.0000
  Table: T1 Alias: T1
  Card: Original: 3000000.000000 Rounded: 3 Computed: 2.91 Non
  Adjusted: 2.91
```

It is clear from this trace file that the optimizer uses the column group extension to get the cardinality estimation using the following formula:

[Click here to view code image](#)

```
card = t1(num_rows) / NDV(SYS_STU5FUD4#KCC03#_TZNTSTN5AV)
card = 3000000.000000 / 1030000 = 2,912621 rounded to 3
```

When the Default Value of C_DDL Changes

Remember that we have defined the C_DDL column to default to 42 when it is not supplied a specific value. Also remember that this has been recorded in the corresponding metadata column without updating the entire table, resulting in the instantaneous change to the structure of the table. But what if during the life cycle of this table, we decide to change the C_DDL column default value to 0, for example? Will this change the existing metadata?

Before executing this second alter table, and for the sake of clarity, take a look at the actual data distribution of the C_DDL column:

```
SQL> select
      c_ddl
      ,count(1)
    from t1
   group by
      c_ddl;
```

C_DDL	COUNT (1)
42	2970000
25	30000

Remember also that the only manual update concerns the C_DDL value 25 because the value 42 comes from Oracle using its internal recorded metadata against the C_DDL column. Now let's modify the default value:

[Click here to view code image](#)

```
SQL> alter table t1 modify c_ddl default 0;
Elapsed: 00:00:00.03
```

Again, this second alter table operation was almost instantaneous. What happens now to the initial query involving the first default value 42? Let's select rows from table t1 that have the initial default value of C_DDL column:

[Click here to view code image](#)

```
SQL> select /*+ gather_plan_statistics */ count(1) from t1 where
      C_DDL = 42;
-----  
2970000  
  
SQL> select * from
      table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));  
-----  
-----
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time
<hr/>						
0	SELECT STATEMENT			1		1
00:00:00.63						
1	SORT AGGREGATE			1	1	1
00:00:00.63						
*	TABLE ACCESS FULL	T1		1		
1500K	2970K	00:00:00.48				

Predicate Information (identified by operation id):

2 - filter(NVL("C_DDL", 42)=42)

The predicate part still uses the NVL function, which was, by the way, expected. But what wasn't foreseen is that the old initial default value of 42 is still referenced in the predicate part. Have we not changed the default value to 0?

In fact, Oracle is considering that the records present in table t1 at the time of the *first alter table* have been physically updated with the default value 42 (although this update is only a metadata change). And only a physical update is necessary to put them back to another default value. Altering the t1 table to modify the default value to 0 will not affect the existing records. It will affect only records inserted after the second *alter table* modification. This is perfectly correct and conforms to the earlier releases when this optimization technique was not used, as shown via the following example:

[Click here to view code image](#)

```
SQL> create table t2 as select
  rownum n1
 , trunc ((rownum -1)/3) n2
 from dual
 connect by level<=10;
SQL> alter table t2 add n3 number default 10;
```

This *alter table* was executed in Oracle Database 11.2.0.3, and the key word NOT NULL was omitted so that the DDL optimization technique will not kick in. Selecting from this table will show that the N3 column has been assigned the default value 10, as follows:

[Click here to view code image](#)

```
SQL> select * from t2;
```

N1	N2	N3
1	0	10

2	0	10
3	0	10
4	1	10
5	1	10
6	1	10
7	2	10
8	2	10
9	2	10
10	3	10

Now, if we change the default value from 10 to 20 and add a few rows without supplying an explicit value for the `C_DDL` column, this is what will happen:

[Click here to view code image](#)

```
SQL> alter table t2 modify n3 default 20;
```

```
SQL> select * from t2;
```

N1	N2	N3
1	0	10
2	0	10
3	0	10
4	1	10
5	1	10
6	1	10
7	2	10
8	2	10
9	2	10
10	3	10

```
SQL> insert into t2(n1,n2) values (10,5);
```

```
SQL> select * from t2 where n1 = 10;
```

N1	N2	N3
10	3	10
10	5	20

This example demonstrates that changing the default value of an existing column follows exactly the same algorithm irrespective of the method used to add this column—the conventional method or the Oracle Database 11g new DDL optimization feature. The default value resulting from the second `alter table` affects only newly inserted rows.

C_DDL Column and Indexes

When a function is applied to a column that figures in the predicate part, it will preclude the use of any index that might exist on this column. In this particular case, will the NVL

function that is applied to the C_DDL column impeach an index to be used by the CBO if this column is indexed? That's what we are going to see next.

Consider the following index:

[Click here to view code image](#)

```
SQL> create index i1_c_ddl on t1(c_ddl);
```

Index created.

Elapsed: 00:00:02.14

And query this again:

[Click here to view code image](#)

```
SQL> select /*+ gather_plan_statistics */ count(1) from t1 where  
C_DDL = 42;
```

```
COUNT(1)
```

```
-----  
2970000
```

```
SQL> select * from  
table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

```
-----  
| Id  | Operation          | Name      | Starts | E-Rows | A-Rows  
| A-Time   |  
-----  
| 0  | SELECT STATEMENT    |           | 1       |        | 1  
| 00:00:00.84 |  
| 1  | SORT AGGREGATE     |           | 1       | 1      | 1  
| 00:00:00.84 |  
|* 2  | INDEX FAST FULL SCAN| I1_C_DDL |         | 1  
| 1500K| 2970K|00:00:00.70 |  
-----
```

Predicate Information (identified by operation id):

```
-----  
2 - filter("C_DDL"=42)
```

There is good news to emphasize here: the index is used. The hidden NVL function is not applied on the C_DDL column when accessing it via the index, which explains why the index is used by the CBO. If the index considers the column without its underlying NVL function, then this means that, at index creation time, the index leaf blocks, in contrast to the table blocks, have been entirely filled up with the default value 42 and not with its corresponding stored metadata.

Another way of reinforcing the nonextension of the DDL optimization technique to indexes is the size of the index itself. Had only a metadata change been made, the size of the index would have been zero. This obviously is not the case, as shown here:

[Click here to view code image](#)

```
SQL>SELECT
      segment_name
     ,bytes/1024/1024 mb
  FROM
    dba_segments
 WHERE
    segment_type = 'INDEX'
  AND segment_name = 'I1_C_DDL';

SEGMENT_NAME   MB
-----  -----
I1_C_DDL        47
```

But you could argue that this is the normal behavior: an index cannot contain null column values. Let's then create a composite multicolumn index with a NOT NULL column to protect non-null values of the C_DDL column:

[Click here to view code image](#)

```
SQL> drop index i1_c_ddl;
Index dropped.

SQL> alter table t1 modify n1 not null;
Table altered.

SQL> create index i2_n1_c_ddl on t1(n1,c_ddl);
Index created.

SQL> select /*+ gather_plan_statistics */ count(1) from t1 where
n1= 101 and C_DDL = 42;

COUNT(1)
-----
1

SQL> select * from
table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));

-----  
-  
| Id  | Operation          | Name           | Starts | E-Rows | A-Rows  
|  
-----  
-
```

```

| 0 | SELECT STATEMENT   |           | 1 |           | 1
|
| 1 | SORT AGGREGATE    |           | 1 |           | 1
|
| * 2 | INDEX RANGE SCAN | I2_N1_C_DDL | 1 |           | 1
|
-----
```

Predicate Information (identified by operation id) :

```
-----  
2 - access("N1"=101 AND "C_DDL"=42)
```

Even when the added C_DDL column is protected against null values by its presence in a composite index, there is no trace of the hidden NVL function applied to the C_DDL column. This example clearly demonstrates that, in contrast to the table blocks where there is no update of the C_DDL column, an index that is created on the same column will see its leaf blocks immediately being populated by the default value of the C_DDL column.

Before finishing this section, let's look at one more interesting issue. We have seen so far that each time the CBO has decided to visit a table block, it has applied the NVL function to the C_DDL column to ensure retrieving a non-null C_DDL value. But we have seen that this filter is always applied when the table is fully scanned (TABLE ACCESS FULL). Will the CBO apply this NVL function when the t1 table is accessed via index (TABLE ACCESS BY INDEX ROWID)? Let's engineer a simple case and observe the CBO reaction in this particular situation:

[Click here to view code image](#)

```

SQL> drop index i2_n1_c_ddl;

SQL> create index i2_n1_c_ddl on t1(n1);

SQL> select /*+ gather_plan_statistics */ count(1) from t1 where
n1= 101 and C_DDL = 42;

SQL> select * from
table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

```

-----  
-----  

| Id  | Operation          | Name      | Starts | E-  

Rows | A-Rows  |  

-----  

| 0  | SELECT STATEMENT    |          | 1      |
|     |                   1 |
| 1  | SORT AGGREGATE     |          | 1      |
|     |                   1 |
```

```

|* 2 | TABLE ACCESS BY INDEX ROWID| T1          |      1
|   1 |           1 |
|* 3 | INDEX RANGE SCAN          | I2_N1_C_DDL |      1
|   1 |           1 |
-----
```

Predicate Information (identified by operation id):

```

2 - filter(NVL("C_DDL",42)=42)
3 - access("N1"=101)
```

Notice that the NVL function is also applied on the C_DDL column even when the table t1 is visited via index row ID.

We can now say that each time the CBO visits a table block, via a single block or a multiblock read, it will apply the NVL function to any DDL optimized column it has to filter from those visited table blocks. However, the CBO will not apply the NVL function to the DDL optimized column if this one is acquired from an index leaf block simply because, in contrast to table blocks, index leaf blocks are physically populated by the column default value at the index creation time.

DDL Optimization for NULL Columns

With the arrival of Oracle Database 12c, it is legitimate to ask whether the DDL optimization is still available. Let's repeat the same experiment for this release:

[Click here to view code image](#)

```
12c> alter table t1 add C_DDL number default 42 not null;
Elapsed: 00:00:00.02
```

This alter table operation happens almost instantaneously. The DDL optimization technique kicks in here, too, as shown, and is proven again via the use of the NVL function in the predicate part of the following query:

[Click here to view code image](#)

```
12c> select count(1) from t1 where c_ddl=42;
-----
```

COUNT(1)
3000000

```
12c> select * from table(dbms_xplan.display_cursor);
```

```

-----
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)
Time					

```

-----
|   0 | SELECT STATEMENT      |           |           |           | 3802
(100)|           |           |
|   1 |  SORT AGGREGATE       |           |           1 | 13
|           |           |
| * 2 |    TABLE ACCESS FULL | T1        | 3538K| 43M| 3802  (1)
00:00:01 |
-----
-----
```

Predicate Information (identified by operation id):

```

-----
2 - filter(NVL("C_DDL",42)=42)
```

Note

```

-----
- dynamic statistics used: dynamic sampling (level=2)
```

But there is a little addition to the DDL optimization in Oracle Database 12c compared to Oracle Database 11g Release 2. In release 12c, the DDL optimization has been extended to null columns having default value. Consider the following alter table done in Oracle Database 11g Release 2 and Oracle Database 12c respectively to clearly appreciate the difference:

[Click here to view code image](#)

```
11.2.0.3.0> alter table t1 add C_DDL_2 number default 84;
```

Table altered.

Elapsed: 00:00:58.25

```
12c> alter table t1 add C_DDL_2 number default 84;
```

Elapsed: 00:00:00.02

Adding the nullable C_DDL_2 column took 58 milliseconds in Oracle Database 11g Release 2, but it is instantaneously done in Oracle Database 12c.

The invariable Oracle Database 12c table size before and after the alter table operation demonstrates the DDL optimization technique for null columns having a default value:

[Click here to view code image](#)

```
12c before alter> SELECT
      segment_name
      ,bytes/1024/1024 mb
  FROM
    dba_segments
 WHERE
    segment_type = 'TABLE'
```

```
AND segment_name = 'T1';

SEGMENT_NAME MB
-----
T1           112

12c after alter> SELECT
                  segment_name
                  ,bytes/1024/1024 mb
  FROM
      dba_segments
 WHERE
      segment_type = 'TABLE'
 AND segment_name = 'T1';

SEGMENT_NAME MB
-----
T1           112
```

This example clearly demonstrates that in Oracle Database 12c, DDL optimization has been extended to include null columns having default values. When you query table t1 to get the distinct values of the newly added column (C_DDL_2), you will realize that all table rows have seen their metadata (default value 84) updated, as shown via the following query:

[Click here to view code image](#)

```
12c> select c_ddl_2, count(1) from t1 group by c_ddl_2;
```

C DDL 2 COUNT(1)

84 3000000

SQL> select cou

```
SQL> select count(1) from t1 where c ddl 2=84;
```

COUNT (1)

3000000

```
SQL> select * from table(dbms_xplan.display_cursor);
```

Time	Id	Operation	Name	Rows	Bytes	Cost (%CPU)
	0	SELECT STATEMENT				3803
(100)	1	SORT AGGREGATE		1	13	

```

|* 2 | TABLE ACCESS FULL| T1      | 3538K|    43M|   3803   (1) |
00:00:01 |
-----
-----
Predicate Information (identified by operation id):
-----
2 -
filter(DECODE(TO_CHAR(SYS_OP_VECBIT("SYS_NC00006$", 0)), NULL, NVL(
    C_DDL_2", 84), '0', NVL("C_DDL_2", 84), '1', "C_DDL_2")=84)

```

Note

- dynamic statistics used: dynamic sampling (level=2)

However, in order to ensure DDL optimization for null columns with a default value, things become more complex than it used to be for non-null columns in the preceding release. We went from a simple implicit use of the NVL function to a complex and exotic predicate part involving a SYS_OP_VECBIT Oracle undocumented function and a new internal column SYS_NC00006\$ in order to honor the default value, as this one has not been physically updated.

In contrast to what you might immediately think, the SYS_NC00006\$ column is not a virtual column. It represents instead a hidden system-generated column, as shown in the following:

[Click here to view code image](#)

```

12c> SELECT
        column_name
       ,virtual_column
       ,hidden_column
       ,user_generated
  FROM
    user_tab_cols
 WHERE
    table_name = 'T1'
  AND column_name = 'SYS_NC00006$';

COLUMN_NAME          VIR HID USE
-----  -----  ---  ---
SYS_NC00006$        NO   YES  NO

```

Even though this column is hidden, it doesn't preempt us from selecting it:

```

12c> select
        a.c_ddl_2
       ,a.SYS_NC00006$
  from t1 a
 where c_ddl_2 =84
  and rownum <=5;

```

```
C_DDL_2  SYS_NC00006$  
-----  
84  
84  
84  
84  
84
```

The `SYS_NC00006$` column will remain null until the `C_DDL_2` column is given a value that is not equal to the default value 84. Consider the following inserts:

[Click here to view code image](#)

```
12c> insert into t1(n1,n2,n3,c1,c_ddl,c_ddl_2) values  
(0,0,0,'xxxxx',110,130);  
  
12c> insert into t1(n1,n2,n3,c1,c_ddl,c_ddl_2) values  
(1,1,1,'xxxxx',140,150);  
  
12c> insert into t1(n1,n2,n3,c1,c_ddl, c_ddl_2) values  
(1,1,1,'xxxxx',200,null);
```

```
12c> select  
      a.c_ddl_2  
      ,a.SYS_NC00006$  
     from t1 a  
    where a.c_ddl_2 in (130,150);
```

```
C_DDL_2  SYS_NC00006$  
-----  
130 01  
150 01
```

```
SQL> select  
      a.c_ddl_2  
      ,a.SYS_NC00006$  
     from t1 a  
    where a.c_ddl_2 is null;
```

```
C_DDL_2  SYS_NC00006$  
-----  
01
```

Notice that the `SYS_NC00006$` column value is no longer `NULL` when we insert a non-default value into the `C_DDL_2` column (including the explicit `NULL` value).

Putting together the different pieces of the puzzle, we can easily understand exactly what that exotic but simple predicate part reproduced here is doing:

[Click here to view code image](#)

```
Predicate Information (identified by operation id):  
-----
```

```

2 -
filter(DECODE(TO_CHAR(SYS_OP_VECBIT("SYS_NC00006$",0)),NULL,NVL(
    C_DLL_2",84),'0',NVL("C_DLL_2",84),'1',"C_DLL_2")=84)

```

Oracle is simply checking through its system-generated column and determining via the SYS_OP_VECBIT function the default value of the C_DLL_2 column or the real value introduced by an end user via an explicit insert statement.

Let's mimic what Oracle is doing with the SYS_NC00006\$ column values, which are 01 and NULL:

[Click here to view code image](#)

```

12c> SELECT
      a.c_dll_2
      ,TO_CHAR(sys_op_vecbit(a.sys_nc00006$,0)) cbo_ddl
   FROM t1 a
  WHERE a.c_dll_2 IN (130,150) -- supplied values
UNION ALL
  SELECT
      a.c_dll_2
      ,TO_CHAR(sys_op_vecbit(a.sys_nc00006$,0)) cbo_ddl
   FROM t1 a
  WHERE a.c_dll_2 IS NULL          -- supplied value
UNION ALL
  SELECT
      a.c_dll_2
      ,TO_CHAR(sys_op_vecbit(a.sys_nc00006$,0)) cbo_ddl
   FROM t1 a
  WHERE c_dll_2 =84                -- default value
  AND rownum <=1
 order by c_dll_2 nulls last;

```

C_DLL_2	CBO_DLL
84	{null}
130	1
150	1
{null}	1

The C_DLL_2 column has four distinct values: the default (84) and three explicitly inserted values 130, 150, and NULL. When you use a predicate against the C_DLL_2 column to retrieve a row from a table block, Oracle will transform the C_DLL_2 into a new column value (CBO_DLL in this case) based on the hidden SYS_NC00006\$ column to compare it against your input bind (or literal) variable. Consequently, it can mimic correctly all the values of the C_DLL_2 column including those having a default value (84) and that have not been physically updated to reflect this default value.

Summary

Oracle Database 11g introduced a wonderful feature that eliminates worry about the

continuity of our application when adding, online, a non-null column with default value to a real-life big production table. This feature, called DDL optimization, allows `alter table` operations to be done not only instantaneously but also without any need to lock the table. We are aware of no restrictions on such an added column that might discourage its use. Oracle Database 12c extended this new `alter table` feature to include null columns with a default value. The icing on the cake is that there seems to be no noticeable performance side effects when using the altered column in a predicate `where` clause. The Oracle Database 12c exotic but inoffensive predicate part ensuring correct values of altered columns seems also to be without side effects. This technique would have been dramatically performant if it had been extended to indexes. However, knowing the complexity of index structures compared to table structures, Oracle is not ready to extend this technique to create an index with columns pointing only to their metadata defined in the table data dictionary.

7. Managing, Optimizing, and Tuning VLDBs

Data is the heart and lifeblood of every business. Data growth in every industry is phenomenal and is increasing at a very rapid pace with each passing year, sometimes more than 100 percent from its previous year. Data warehousing is a common requirement for any organization, whatever the size. At the same time, configuring and maintaining very large databases (VLDBs) and extremely large databases (XLDBs) are challenging tasks that require advanced skills. This chapter presents a 360-degree overview of managing VLDBs, offering tips for best practices and strategies in configuration, performance, backup and recovery, and maintenance.

Overview of Very Large Databases

As many Oracle and data warehousing gurus have voiced, there is no fixed definition or standard rule to categorize a certain sized database as a VLDB. Not that long ago, databases that were only hundreds of gigabytes in size were considered to be VLDBs; however, over the past few years, the definition of VLDBs has changed significantly due to enormous data growth. It is not unusual to encounter databases that have grown to the size of several terabytes or even petabytes and that hold billions or even trillions of records. VLDBs and XLDBs therefore demand highly efficient software and hardware resources and require extremely large amounts of storage capacity.

VLDBs and XLDBs used for decision support systems (DSS) are often referred to as *data warehouse* (DW) databases. DSSs are critical to an organization's business management to analyze the functionality and business growth of the products and services that the organization offers. Data warehouses generally hold historical data that is loaded from various data sources, including another database, a flat file, an Excel spreadsheet, and other sources. Extract-transform-load (ETL) tools are typically used to load data from those sources, while applications and utilities such as business intelligence (BI) and business objects are used to generate management reports and dashboards for the business's needs.

VLDBs and XLDBs therefore engender a unique set of challenges and can raise some daunting tasks for the information technology (IT) department of an organization. They demand state-of-the-art technologies to meet myriad business needs while simultaneously delivering optimal performance. When designing such a database, you will almost certainly encounter the need for high-end hardware resources, huge storage capacities, large network bandwidths, and special considerations for backup and recovery. The following sections present best practices and discuss how to apply the tools, tips, and tricks to ensure a solid VLDB and XLDB foundation so that you can manage them with ease.

Optimal Basic Configuration

One of the key factors in having an optimal setup is getting the basics right and building a

solid foundation. This segment briefly reviews some of the initial configuration tips for VLDBs:

- Choosing the right database configuration template in DBCA
- Selecting the optimal data block size
- Sizing adequate system global area (SGA), program global area (PGA), and other memory components
- Leveraging data compression
- Using temporary tablespaces effectively
- Implementing partitioning for easy data management
- Making the right choices for partitioned indexes (especially global vs. local)
- Enabling parallelism to take advantage of multiple CPUs
- Verifying application code for effectiveness before its production deployment
- Implementing appropriate backup and recovery strategies

Data Warehouse Template

When deploying a new database, it is advisable to follow some basic rules specific to the application category: online transaction processing (OLTP), DSS, or a combination of both. To create a new Oracle database, you can use the Database Configuration Assistant (DBCA), which is a Java-based GUI tool, or you can use the `CREATE DATABASE` statement, a manual approach that requires running scripts. Depending on the nature of the application—that is, whether it's an OLTP or a DSS application—you should choose the appropriate database creation template through DBCA, as shown in [Figure 7.1](#). If the database is intended for DSS or VLDB, choose the Data Warehouse template so that the appropriate database initialization parameters, online redo log sizing, and tablespace sizing are set properly.

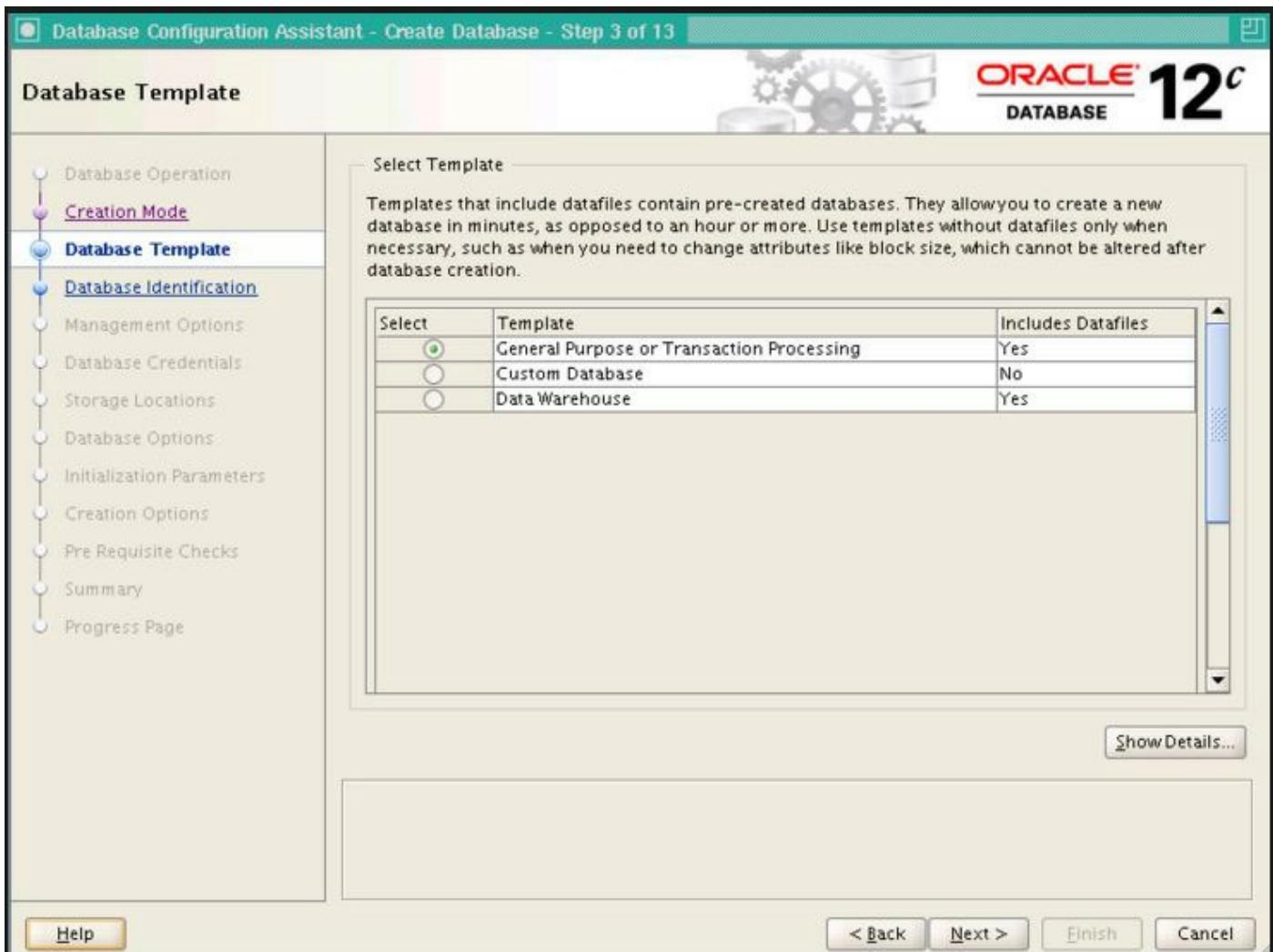


Figure 7.1 Database Configuration Assistant: Create database template

Optimal Data Block Size

It is essential to choose the right database block size for databases, especially for VLDBs and XLDBs. The data block size has an impact on the overall database read and write performance. Although the default 8 KB block size is most appropriate and can meet the demands of an OLTP application, DSS systems generally require a larger block size. If the database is already configured with a default 8 KB block size, the DBA can use Oracle's multiple block feature. Under some circumstances, using a 16 KB database block size or even a 32 KB size for VLDBs and XLDBs would yield performance benefits over a 4 KB or 8 KB block size. The block size is controlled with the `db_block_size` initialization parameter.

Oracle supports multiple data block sizes within the same database, but the practice is not encouraged unless your application demands it. Any VLDB might have data block sizes of 16 KB or even 32 KB, but only the DBA can determine which block size best supports the application's performance and behavior. Remember, once a database is created with a default block size, the default block size cannot be modified unless the database is re-created.

Following are a few tips for choosing the right block size that meets the application's needs:

- A smaller block size is efficient when rows are smaller and data access is random.
- Choose a larger block size to improve the read performance when the rows are smaller and access is sequential or when you have a mixture of random and sequential reads.
- Larger block size produces significant read performance when the rows are larger, such as in large object (LOB) columns.
- With large block size for high concurrency systems, ensure you have set appropriate values for the INITTRANS and MAXTRANS parameters.
- Larger block size has less overhead and stores more rows in a single block.
- With a larger block size, several rows are put into buffer cache with a single read.

Bigfile Tablespaces

Today's VLDBs and XLDBs commonly grow into terabyte (TB) or even petabyte (PB) sizes. Traditionally, the larger the database, the more tablespaces and datafiles will be required to support the increasing data demands. Depending on the operating system platform, the datafiles of a smallfile tablespace are allowed to grow to a maximum size of 32 GB for an 8 KB block size (128 GB for a 32 KB block size).

To limit the number of datafiles for VLDBs, Oracle introduced the bigfile tablespace concept, which allows a single large datafile per tablespace. However, that single datafile can grow to a maximum size of 32 TB for an 8 KB block size (and up to 128 TB for a 32 KB block size). This feature eliminates the need for numerous datafiles and simplifies tablespace management. Keep in mind that bigfile tablespaces are valid only for locally managed tablespaces with automatic segment space management.

When you create bigfile tablespaces, you need to ensure there is enough free space for growth on the storage/filesystem and that the system supports striping.

[Click here to view code image](#)

```
SQL> CREATE BIGFILE tablespace data_ts DATAFILE size 10G;
SQL> CREATE BIGFILE TABLESPACE data_ts DATAFILE
  '/oradata/data_ts01.dbf' SIZE 10G;
```

Refer to the BIGFILE column in the V\$TABLESPACE dictionary view to identify whether the tablespace is a traditional smallfile or bigfile tablespace.

[Click here to view code image](#)

```
SQL> select name,bigfile from v$tablespace;
```

NAME	BIG
SYSTEM	NO
UNDOTBS1	NO
SYSAUX	NO
USERS	NO

TEMP
DATA_TS

NO
YES

Adequate SGA and PGA

Another critical decision a DBA must make concerns the amount of memory resources that should be assigned for the system and program global areas for the database instance used for a VLDB or XLDB. The applications that run against VLDBs typically require much higher rates of data access and process complex calculations, so it is essential to have optimal SGA and PGA to avoid performance pitfalls.

Fortunately, Oracle provides multiple options to conquer memory management and configuration hassles for large-sized database demands. Although you can manually configure memory components such as SGA and PGA separately, you might want to take advantage of the automatic memory management (AMM) feature to handle the SGA and PGA together automatically. It is also critical to ensure there is no frequent dynamic memory allocation and deallocation happening among the SGA components, especially during peak business hours. Review V\$SGA_DYNAMIC_COMPONENTS, V\$SGA_CURRENT_RESIZE_OPS, V\$SGA_RESIZE_OPS, and V\$SGA_DYNAMIC_FREE_MEMORY dynamic performance views to analyze whether the current SGA sizing is optimal for your environment.

There is no rule of thumb to define the perfect memory allocation for SGA and PGA, but you may consider starting with an optimal size and then monitoring the SGA and PGA statistics to adjust these settings as needed. Since VLDB applications tend to perform a lot of sorting, ensure that you configure adequate PGA as well. You can use the V\$MEMORY_TARGET_ADVICE dynamic view to adjust the value of AMM if it is in use; also, the V\$SGA_TARGET_ADVICE and V\$PGA_TARGET_ADVICE dynamic views are helpful for tuning and adjusting the values for SGA_TARGET and PGA_AGGREGATE_TARGET, respectively. You can always adjust the SGA settings; sometimes a database restart is necessary for the new values to take effect.

Temporary Tablespace Groups

Managing and configuring the proper sizes of temporary tablespaces can become a significant challenge for a DBA as VLDB/DSS applications tend to perform heavy sorting operations quite often.

Essentially, the Oracle database uses temporary tablespaces to segregate the temporary work from the real work in the system. Performance bottlenecks on the temporary tablespace are often caused by an undersized tablespace, which results in the system running out of adequate temporary space when concurrent user queries increase the demand for temporary space. Temporary tablespace groups (TTGs), introduced in Oracle Database 10g, provides the ability to offer performance benefits by spreading I/O across multiple temporary tablespaces. A TTG can be created initially with a single temporary tablespace, and multiple temporary tablespaces can be added later. When configuring a TTG, consider having equally sized temp files and the same number of

temp files for each temporary tablespace that will be part of the TTG. A TTG is a better option than having one large temporary tablespace or multiple temporary tablespaces assigned to multiple users in the database.

Data Partitioning

Managing the massive amounts of data typically stored in a VLDB or XLDB is not a simple task. It demands manageability, availability, and good performance. Oracle's partitioning features offer a wide range of partitioning types to support your data and application needs. Data partitioning enhances easy data management, improves data availability, and delivers significant performance by dividing huge amounts data into much smaller segments.

Oracle continues to improve and expand partitioning capabilities in each new release of the database. Oracle supports table-level, index-level, and index-organized table (IOT)-level partitioning. When partitioning is applied to an object, each partition is treated as a separate segment. According to the need of your application, you can implement an appropriate partition strategy; when you can't decide on a strategy, or when your data doesn't call for range or list partitioning, you may choose to go with hash partitioning.

Oracle Database 11g made partitioning management easier with the introduction of interval partitioning, an extension of range partitioning, which allows a DBA to define automated partitioning of data on the basis of specified timestamp or numeric intervals. With this new feature, partitions are created automatically whenever the data demands. And when existing columns do not present an appropriate partitioning key, Oracle 11g also supports the ability to partition data on the basis of the value of an expression that can be stored in a virtual column.

DBAs also often agonize over when partitioning should be applied to a table, which column is the best choice for the table's partitioning key, and whether partitioning should be applied to a table in an OLTP environment. There are many opinions on how best to answer these questions, but a good rule of thumb is that if the table is huge and often requires complex data maintenance, it makes excellent sense to at least consider partitioning regardless of whether an OLTP or DSS application is accessing the table.

Index Partitioning: Local versus Global

One of the tricky tasks that a DBA typically has to address when dealing with DSS applications is deciding between the local and global partitioned indexes for a partitioned table. The type of index partitioning serves an important role in data access, data loading, and partition maintenance, so it generally pays to invest considerable time when making these decisions.

The general perception is that global indexes deliver better performance for OLTP applications, whereas local indexes provide better performance benefits for DSS applications running against VLDBs.

Local index partitions not only provide better performance but also are easier to

maintain and offer great application availability. Because each local index partition maps to a single corresponding individual table partition, this generally provides good application availability when a MERGE, SPLIT, EXCHANGE, DROP, or TRUNCATE partition operation is applied against one or more table partitions. However, you can't create a local index on the primary key and the unique index. When MERGE, SPLIT, EXCHANGE, DROP, and similar operations are performed, you need to either rebuild the index or add the UPDATE GLOBAL INDEXES clause.

Data Compression

Data retention may differ dramatically among organizations according to their different business needs, regulatory requirements, and even the data owner's desire to keep historical data long past its perceived value. When the retention period is longer, a VLDB tends to encompass huge volumes of historical data, which will of course need a correspondingly huge amount of physical storage capacity. DBAs must therefore carefully consider exactly how much data is being retained and implement appropriate maintenance policies to insure that storage space is not exhausted unexpectedly or prematurely.

Oracle's various data compression features may provide significant storage savings. If you are fortunate enough to be using Exadata or the ZFS logical volume manager for your VLDB, those storage platforms offer *Hybrid Columnar Compression* (HCC) features through which you may be able to save significant storage capacity. If your VLDB is not using these storage platforms, however, you may still be able to make use of *Oracle Advanced Compression* features to compress data within the database.

Oracle Advanced Compression is a separately licensed feature that provides comprehensive storage savings. As of Oracle Database 12c, Advanced Compression offers myriad features, including Advanced Data Compression, Heat Map, Automatic Data Optimization (ADO), Advanced Row Compression, Advanced Index Compression, and Advanced Network Compression.

Table Compression

Table compression allows you to compress the data within a table. Basic table compression, which doesn't require any additional license, applies an average of 10X compression to a table's data but applies only to initial direct-path loading. In contrast, Advanced Compression's *Advanced Row Compression*, called OLTP compression in earlier releases, applies row compression to tables during all data manipulation language (DML) operations. If you are using either Exadata or ZFS storage, then HCC offers warehouse and archive compression methods in addition to basic and advanced compression.

Here are some examples of table-level compression options:

[Click here to view code image](#)

```
SQL> CREATE TABLE table_name (column_list...) ROW COMPRESS  
ADVANCED;
```

```
SQL> ALTER TABLE table_name ROW STORE COMPRESS BASIC;
SQL> ALTER TABLE table_name ROW STORE COMPRESS |ADVANCED;
SQL> ALTER TABLE table_name MOVE PARTITION partition1 ROW STORE
COMPRESS BASIC | ADVANCE;
SQL> ALTER TABLE table_name NOCOMPRESS;
```

The following query lists the objects on which compression is applied:

[Click here to view code image](#)

```
SQL> SELECT table_name,compression, compress_for FROM user_tables
WHERE compression = 'ENABLED';
```

Heat Map and Automatic Data Optimization

Often, historical data in a VLDB becomes less active as it grows older. If this is true of data contained in a partitioned table, Oracle 12c's new (and complementary) Heat Map and ADO features may be particularly useful. You can define ADO policies and conditions under which data should be moved to a different storage tier or compressed at a higher compression level; ADO will then automatically apply the appropriate compression based on heat map statistics, but only when the corresponding conditions are satisfied.

To enable the Heat Map feature, all that is required is to set the HEAT_MAP initialization parameter to ON (the default value is OFF):

[Click here to view code image](#)

```
SQL> ALTER SESSION | SYSTEM SET HEAT_MAP = ON;
SQL> ALTER SESSION | SYSTEM SET HEAT_MAP = OFF; -- disable Heat Map
```

To list the statistical information about the Heat Map settings for database objects, you can use views V\$HEAT_MAP_SEGMENT and USER_HEAT_MAP_SEGMENT. Views DBA_HEAT_MAP_SEGMENT and DBA_HEAT_MAP_SEQ_HISTOGRAM provide additional information about the individual data segments and their corresponding heat map settings.

ADO allows you to define the policies at table level, tablespace level, and row level for smart compression and automatic data movement. The following examples demonstrate how to deploy ADO policies at various levels for a table. In this example, when 60 days has elapsed since any DML has been performed against any data in the sales table, that table's segments will be automatically compressed using ADVANCED compression:

[Click here to view code image](#)

```
SQL> ALTER TABLE sales
      ILM ADD POLICY
      ROW STORE
      COMPRESS ADVANCED
      SEGMENT
      AFTER 60 DAYS OF NO MODIFICATION;
```

In the subsequent example, when 30 days has elapsed since any DML has been performed against any data in the sales_decl4 partition of the sales table, it will be automatically compressed:

[Click here to view code image](#)

```
SQL> ALTER TABLE sales
      MODIFY PARTITION sales_decl4
      ILM ADD POLICY
      ROW STORE
      COMPRESS ADVANCED
      ROW
      AFTER 30 DAYS OF NO MODIFICATION;
```

To disable and then remove all ADO policies against the sales table, use the following syntax:

[Click here to view code image](#)

```
SQL> ALTER TABLE sales ILM DISABLE_ALL;
SQL> ALTER TABLE sales ILM DELETE_ALL;
```

Use the following example to query the information on ADO:

[Click here to view code image](#)

```
SQL> SELECT policy_name, policy_type, enabled FROM
      user_ilm_policies;
```

Advanced Index Partition Compression

Oracle Database 12c supports Advanced Index Compression, which yields storage reduction. The Advanced Index Compression feature supports unique and nonunique indexes while maintaining good performance during the index access operation. The compression can be applied at partition level, as demonstrated in the following:

[Click here to view code image](#)

```
SQL> CREATE INDEX index_name ON(column) COMPRESS ADVANCED HIGH
      LOCAL (PARTITION
              ip1 COMPRESS ADVANCED LOW, PARTITION ip2 COMPRESS HIGH,
              PARTITION ip3 NOCOMPRESS);
SQL> CREATE INDEX index_name ON(column) LOCAL (PARTITION ip1
      COMPRESS ADVANCED LOW,
              PARTITION ip2 COMPRESS HIGH, PARTITION ip3);
SQL> CREATE INDEX index_name ON(columns_list...) COMPRESS ADVANCED
      LOW;
```

VLDB Performance Tuning Principles

Delivering optimal performance for applications that access a VLDB is essential. If Oracle database best practices are applied, many tuning nightmares may disappear. Following are the most common performance issues that can affect VLDBs:

- Suboptimal application coding
- Inefficient application and database design
- Inaccurate or missing optimizer statistics
- Lack of indexes or too many indexes
- Limited hardware resources
- Bad network design

Real-World Scenario

To demonstrate the issue of suboptimal application coding, let's look at a classic example of a poor performance incident—an issue that was ultimately resolved with a simple code modification.

A reconciliation batch scheduled to run at the end of each business day was performing inconsistently. The job sometimes completed within only 2 hours and sometimes took more than 6 hours; during the database's worst performance, it took nearly 24 hours to complete. This inconsistent performance not only gave the DBA team nightmares, it also significantly impacted the subsequent dependent jobs, and as a result, several key business users were unable to get their daily reports on time.

The job itself was straightforward, involving no complex queries or convoluted logic. The job was to perform the following simple tasks:

1. Delete data from the target tables.
2. Load data from text files into the target tables.
3. Run queries against the target tables to generate reports.

During the course of investigation, the following behavior was observed:

- The database was growing in size at a uniform rate each day.
- Important queries were favoring the nested loops in the query execution plan.

As a temporary workaround the following action plan was used:

1. Reorganize the target tables using the `alter table ... move` command.
2. Rebuild all indexes in the schema.
3. Gather application schema statistics.
4. If the preceding steps fail to resolve the inconsistent performance, export the schema into another database residing on a different server that has more CPU and memory capacity.

Despite these workarounds, the problem reappeared and gradually returned to the worst state (taking nearly 24 hours to complete). Interestingly, when the person responsible for the application was contacted, he mentioned that data in all tables was deleted and subsequently loaded from different sources, and the observed growth was unlikely with the amount of data he was loading daily.

Then the focus shifted to the data load script. Upon reviewing the code, a classic

mistake was observed: data was removed from the tables with the `DELETE` command followed by direct path loads.

If you know the basics, you can easily spot the reason for the consistent growth in the database. When `DELETE` is used to remove complete data from a table, the high-watermark is not reset; therefore, subsequent direct-path load inserts start loading data above the high-watermark, ignoring the free data blocks below the high-watermark. This approach causes continuing data growth and consumed significant amounts of free space in the datafiles. Hence, queries on these tables were performing badly.

The simple solution to this issue was to replace the `DELETE` command with `TRUNCATE TABLE`. Once this change was applied, the job finished in just 2 hours. There have been no issues with performance in the 6 years since the problem was encountered.

The moral of the story is that if you get the basics right, tuning the code just a little could offer significant benefits for many performance issues.

Limiting the Impact of Indexes on Data Loading

Data loading is a common practice in data warehouse databases. In general, indexes improve queries' data selection performance; however, having too many indexes sometimes hampers DML (insert, update, and delete) performance. When data loading doesn't produce expected throughput and when a DBA receives performance-related concerns, one of the key areas to investigate is the number of indexes on the table.

One of the typical methods used to improve the performance of bulk data loading is to disable indexes and constraints on the table and then rebuild them after data loading completes. Although this may seem like a good workaround, it can be quite time consuming to rebuild indexes on extremely large tables. Therefore, it is recommended to minimize the number of indexes on the table to improve the data loading process.

There are a few approaches to find out which indexes are being used and which are not. One of the methods is to enable monitoring of index usage with the `ALTER INDEX ... MONITORING USAGE` command. Once you activate monitoring for indexes, you can verify their usage through the `V$OBJECT_USAGE` dynamic view; if monitoring shows that some of the indexes are not being used, you may consider putting them in invisible mode and dropping them after obtaining permission from the application owner/developers.

Oracle Database 12c gives you the ability to create partial indexes on a partitioned table. This cool feature provides the flexibility to turn on and off local and global indexing for individual partitions of a table. The following example demonstrates how to create a table with an `INDEXING OFF | ON` option:

[Click here to view code image](#)

```
SQL> CREATE TABLE emp (eno number(9),ename varchar2(100),dob date,
dept number(2), salary
      number(6)) INDEXING OFF
PARTITION BY RANGE (salary)
```

```

(PARTITION p10000 values less than (10001) INDEXING OFF,
partition p20000 values less than (20001) INDEXING ON,
partition p30000 values less than (30001));

SQL> CREATE INDEX index_p1 ON emp(eno) GLOBAL INDEXING
FULL|PARTIAL;
SQL> CREATE INDEX index_p2 ON emp(salary) LOCAL INDEXING PARTIAL;
SQL> ALTER TABLE emp MODIFY PARTITION p10000 INDEXING ON;

```

When indexes are created with the PARTIAL option on a table:

- For a local index, partitions that are tagged as INDEXING OFF will have an UNUSABLE index status, and partitions that are tagged as INDEXING ON will have a USABLE index status.
- For a global index, only those partitions that are tagged as INDEXING ON will be active, and the rest will be excluded.

To view an index's status, refer to the INDEXING column in DBA | USER _ PART _ TABLES, DBA | USER _ TAB _ PARTITIONS, and DBA | USER _ TAB _ SUBPARTITIONS. Keep in mind that a partial index can't be applied to unique indexes.

Maximizing Resource Utilization

Two things are very common in VLDB environments: huge hardware resources and massive data. The queries that access VLDBs often either scan large amounts of data or perform massive computation on huge amounts of data. When we have a solid hardware presence, it is up to us to best utilize those resources to improve the overall application performance. One of the best ways to do this is to divide and drive: that is, divide the workload of the query into smaller pieces so it will complete as soon as possible. Don't panic. We are not talking about rocket science here—just leveraging Oracle's capabilities in regard to parallelism.

When parallelism is applied at any level—database, table, or query—Oracle distributes the workload of the query among multiple slave processes and completes the job more quickly than if it were executed serially. Parallel execution utilizes multiple CPU and I/O resources on the servers to achieve improved performance by reducing the response time needed to execute the query.

As just mentioned, you can define parallelism at various levels in an Oracle database:

- Object level (table or index)
- Session level
- Database level
- At statement runtime

If you are uncertain about the degree of parallelism to use, you can leave this to Oracle, as Oracle will automatically determine the number of parallel processes to use during execution based on the settings for initialization parameters `cpu_count` and

`parallel_threads_per_cpu`. Following are some of the key initialization parameters pertaining to parallel execution and their default values:

[Click here to view code image](#)

<code>parallel_degree_policy</code>	string	MANUAL
<code>parallel_execution_message_size</code>	integer	16384
<code>parallel_max_servers</code>	integer	240
<code>parallel_min_percent</code>	integer	0
<code>parallel_min_servers</code>	integer	0
<code>parallel_server</code>	boolean	TRUE
<code>parallel_servers_target</code>	integer	96
<code>parallel_threads_per_cpu</code>	integer	2

Using parallelism is generally advised while dealing with huge amounts of data. For example, the following are some of the statements that might benefit from parallelism:

- CTAS (CREATE TABLE AS SELECT)
- ALTER TABLE | PARTITION MOVE | SPLIT | PARTITION
- ALTER INDEX REBUILD
- CREATE INDEX
- INSERT AS SELECT
- INSERT /*+ APPEND */ (direct path INSERTs)

If you are working with an Oracle Real Application Clusters (RAC) database, when you set the `PARALLEL_FORCE_LOCAL` initialization parameter value to TRUE, it restricts the SQL statement parallel execution to a single instance.

Gathering Optimizer Statistics

Maintaining accurate and up-to-date optimizer statistics always is critical to generating optimal SQL execution plans by the Oracle optimizer. Many performance issues can be resolved simply by gathering fresh optimizer statistics. The challenge comes when you have to gather optimizer statistics on large tables. You must consider the time required to gather the statistics, determine how often statistics must be gathered, and find the most efficient method of gathering them. The following sections provide some best practices for collecting accurate optimizer statistics quickly on large tables and table partitions.

Incremental Statistics Synopsis

Accurate and consistent optimizer statistics on tables helps to improve query performance; however, that need must be balanced against how to collect and maintain accurate statistics consistently on large partitioned tables in a timely manner. Although several workarounds and methods exist to accomplish this, we focus on the benefits of gathering incremental statistics and look at real-world examples.

The concept of gathering incremental statistics was first introduced in Oracle Database 11g with the intention to enhance the performance of gathering statistics at global and partition levels on large partitioned tables. Incremental statistics on

partitioned tables are gathered at the global, partition, and subpartition levels. When incremental preferences are enabled, the statistical metadata for each partition of the table are stored in the form of a synopsis in the SYSAUX tablespace.

The synopsis helps to produce accurate global-level statistics by aggregating partition-level statistics, which eliminates the need to scan the full table. In the context, whenever a new partition is added, the global statistics are updated automatically using the existing partitions' synopses and the new partition's synopsis. Whenever 10 percent of the data is changed on a partitioned table, the statistics become stale and need to be regathered.

Note

Bug 16851194 reports continued growth in the SYSAUX tablespace without growth in actual data. Therefore, it is advisable to monitor the SYSAUX tablespace's growth periodically to ensure you are not hitting this bug. The bug was fixed in Oracle 12.1.0.2.

Starting in Oracle Database 12c, the default behavior for stale statistics is controlled with the new INCREMENTAL_STALENESS argument. The default value for INCREMENTAL_STALENESS is NULL, which invokes the same behavior as in Oracle 11g. Also in Oracle 12c, global-level statistics are automatically calculated for any table whose statistics are either locked or stale.

Additionally, the USE_STALE_PERCENT argument can be used to define a different threshold value for the statistics' staleness percentage. The default threshold value for statistics staleness is 10 percent, but it can be overridden so that statistics will not be considered stale until the percentage of changed rows has reached the specified threshold value:

[Click here to view code image](#)

```
SQL> exec dbms_stats.set_database_prefs('INCREMENTAL_STALENESS',
  'USE_STALE_PERCENTAGE');
SQL> exec dbms_stats.set_database_prefs('STALE_PERCENT', '25');
```

Use the following example to get the current value for STALE_PERCENT:

[Click here to view code image](#)

```
SQL> exec dbms_stats.get_prefs('STALE_PERCENT', 'SCHEMA',
  'TABLENAME') STALE_VALUE FROM dual;
```

To maintain and gather incremental statistics on a partitioned table, you must run through the following procedure first:

1. The INCREMENTAL statistics preference is not enabled by default; verify its current setting using this code:

[Click here to view code image](#)

```
SQL> SELECT dbms_stats.get_prefs('INCREMENTAL', 'SCHEMA',
```

```

TABLENAME')
    tab_incr_prefs FROM dual;

TAB_INCR_PREFS
-----
---
FALSE

```

2. Turn on the INCREMENTAL statistics preference on the table:

[Click here to view code image](#)

```

SQL> exec dbms_stats.gather_table_stasts('SCHEMA',
    'TABLENAME',ESTMATE_PERCENT=>dbms_stats.auto_sample_size);

SELECT dbms_stats.get_prefs('INCREMENTAL','SCHEMA','TABLENAME')
    tab_inc_perf FROM dual;

```

3. Gather incremental statistics and ensure the incremental flag is turned on:

[Click here to view code image](#)

```

SQL> exec dbms_stats.set_table_prefs('SCHEMA', 'TABLENAME',
    'INCREMENTAL', 'TRUE');

TAB_INC_PERF
-----
---
TRUE

```

The following query identifies which tables have the INCREMENTAL statistics flag turned on:

[Click here to view code image](#)

```

SQL> SELECT owner, object_name
  FROM dba_objects
 WHERE object_id IN
    (SELECT DISTINCT(obj#)
      FROM optstat_user_prefs$
     WHERE PNAME='INCREMENTAL'
       AND VARCHAR='TRUE');

```

This example shows how to activate incremental statistics at the schema level:

[Click here to view code image](#)

```

SQL> exec dbms_stats.set_SCHEMA_prefs('SCHEMA', 'INCREMENTAL',
    'FALSE');

```

Use the following syntax to disable the incremental statistics for an individual table:

[Click here to view code image](#)

```

SQL> exec dbms_stats.set_table_prefs('SCHEMA', 'TABLENAME',
    'INCREMENTAL', 'FALSE');

```

Use the following syntax to disable the incremental statistics at the schema level:

[Click here to view code image](#)

```
SQL> exec dbms_stats.set_schema_prefs('SCHEMA', 'INCREMENTAL',
  'FALSE');
```

Gathering Statistics Concurrently

Gathering statistics in a schema sometimes can be time consuming because of the large number of tables and indexes. Before Oracle 11.2.0.2, there was no direct method available to gather statistics concurrently on multiple objects. The concurrent statistics-gathering feature (also known as *interobject parallelism*) in Oracle 11.2.0.2 provides the ability to gather statistics on multiple tables or partitions concurrently. The objective of this feature is to fully utilize the database server's hardware resources to improve the efficiency of gathering statistics and thus reduce the overall time required to gather them.

This behavior is controlled by the CONCURRENT global preference with the DBMS_STATS.GATHER_TABLE_PREF package. By default, the value for CONCURRENT is FALSE; in order to enable this feature, it must be explicitly set to TRUE:

[Click here to view code image](#)

```
SQL> exec dbms_stats.set_global_prefs('CONCURRENT', 'TRUE');
SQL> SELECT dbms_stats.get_prefs('CONCURRENT') FROM dual;
  DBMS_STATS.GET_PREFS('CONCURRENT')
-----
---  
TRUE
```

When the value is set to TRUE, Oracle takes advantage of Job Scheduler and Advance Queuing (AQ) mechanisms to perform concurrent statistics gathering. The number of tables used for concurrent statistics collection is directly proportional to the value of the JOB_QUEUE_PROCESSES initialization parameter. When schema-level statistics gathering is triggered, the number of tables for which statistics will be gathered concurrently is determined by the JOB_QUEUE_PROCESSES value; the remaining tables in the schema will be queued until statistics gathering for the current batch of tables is completed. When schema-level or database-level statistics are triggered, a separate job is created for each nonpartitioned and partitioned table.

The CONCURRENT argument can be set to MANUAL, AUTOMATIC, ALL, or OFF (the default value). When the CONCURRENT value is set on a partitioned table, a separate job is triggered for each partition of the table. The CONCURRENT parameter can be set to table level, schema level, or database level. Therefore, when you have schema with a large number of tables and partitions, make use of the CONCURRENT feature to reduce the overall time required to gather statistics. Setting the value to AUTOMATIC allows the automatic statistics-gathering job to take advantage of concurrently gathered statistics.

Also, ensure that the RESOURCE_MANAGER_PLAN and JOB_QUEUE_PROCESSES initialization parameters are set to appropriate values.

To review or monitor the concurrent statistics-gathering jobs, use the following examples:

[Click here to view code image](#)

```
SQL> SELECT owner,job_name,state,start_date,max_run_duration
  FROM dba_scheduler_jobs
 WHERE job_class like 'CONCURRENT%' AND state in ('RUNNING',
 'SCHEDULED');

SQL> SELECT job_name, state, comments
  FROM dba_scheduler_jobs
 WHERE job_class LIKE 'CONC%' and STATE = 'RUNNING';

SQL> SELECT job_name, elapsed_time
      FROM dba_scheduler_running_jobs WHERE job_name LIKE 'ST$%';
```

Oracle Database 12c offers a significant enhancement: statistics gathering for very small and empty tables are encapsulated into a single batch job, which reduces the overhead of statistics maintenance.

Setting the ESTIMATE_PERCENT Value

Another vital consideration when gathering table statistics is to decide on an appropriate estimate percentage. Although gathering statistics against 100 percent of a table's data will certainly provide accurate statistics, the time and computing resources required for this task—especially for extremely large tables—may have a significant negative impact on your database's performance.

Obviously, it can be tough to settle on a value for the estimate percentage, as both large and small values have their own set of benefits and disadvantages. The AUTO sampling statistics-gathering feature of Oracle 11g helps to automatically determine an appropriate estimate percentage while gathering table statistics. The accuracy of AUTO sampling is nearly that of a 100 percent sample size; additionally, it takes less time to complete statistics gathering using AUTO sampling. The new AUTO sample algorithm—now implemented by default in Oracle Database 11g Release 2—also influences how index statistics are gathered. The AUTO sample algorithm performs a full table scan instead of a sampling clause to build a synopsis per column for column-level number of distinct values (NDV) calculations.

The following example demonstrates how to use the AUTO sampling size with the DBMS_STATS package. In order to use the incremental statistics-gathering feature mentioned previously, ESTIMATE_PERCENT *must* be specified as AUTO_SAMPLE_SIZE:

[Click here to view code image](#)

```
SQL> exec dbms_stats.gather_table_stasts(null,'tablename',
```

```
estimate_percent=>dbms_stats.auto_sample_size);
```

Backup and Recovery Best Practices

It doesn't matter whether your database is supporting OLTP or DSS workloads—the data stored within is critical to your organization's business continuity. One of the prime responsibilities of a DBA is to protect the data from any potential disasters and, when necessary, recover the database according to the application service level agreement (SLA). For VLDBs and XLDBs, backup and recovery thus becomes a prime concern not only to the DBA but also to the business organization's management. Following are several strategies that will help improve performance for backup and recovery, including reducing overall backup time, optimizing backup duration and size, and reducing database recovery time:

- Configure fast incremental backups using the block change tracking (BCT) feature. This saves a good amount of backup time because it scans only the blocks that are modified since the previous full backup.
- Perform a weekly full backup followed by cumulative daily incremental backups.
- Use RMAN incremental updates/merge backups to significantly reduce database recovery time.
- If sufficient disk space is not a barrier, make backups to disk first and then copy them to tape.
- For tablespaces that contain historical or static data, back them up once, bring them into read-only mode, and then exclude them from the daily backups.
- Make use of multisection backups to back up large datafiles so that data in the files can be backed up in parallel.
- Recovering VLDBs and XLDBs can be extremely time consuming. Therefore, activate flashback logging and use guaranteed restore points to overcome known logical errors.
- Compress historical data to gain storage savings, which ultimately reduces backup time and size.
- If you have implemented Oracle Data Guard, back up your database from the physical standby database instead of the primary database to reduce resource demands.
- Use RMAN compressed backups to reduce backup sizes.
- If your VLDB or XLDB is extremely large and backup processing exceeds the time window allotted, use the PARTIAL option in concert with the DURATION directive to permit RMAN to retain backups of the database even though all datafiles haven't been backed up. This practice essentially scatters backup creation over multiple days within the specified backup execution window defined by DURATION.

Exadata Solutions

When the Exadata Database Machine was introduced, it was marketed as an excellent solution to deliver extreme performance for data warehouse databases. Although later versions of Exadata support OLTP workloads as well as DSS systems, it is still highly recommended to consider Exadata solutions for VLDBs and XLDBs.

With its state-of-the-art technologies and massive hardware resources, Exadata genuinely delivers extreme performance for DSS applications. Exadata combines a significant amount of hardware resources (CPU and memory), fast networking components (InfiniBand at 40 Gbe/s), flash disks, and numerous software features such as query offloading, HCC, and smart features (smart flash cache, smart scan) to deliver excellent performance. If an Exadata solution fits your organization's budget for any data warehouse, VLDB, or XLDB, use it and get the extreme performance benefits.

Utilizing a Data Guard Environment

Many companies implement a Data Guard configuration for their data warehousing databases. If you work for one of these companies, consider using an existing Data Guard environment to maximize the returns on that investment by offloading your workload for the production database.

If you have in place an active standby Data Guard that is configured for your primary production VLDB, you can leverage it to offload or redirect some of the workload to that standby database. For instance, if the business department wants to generate daily reports after the nightly batch or data loading using aggregative or complex queries against huge amounts of data, you can shift these queries to run against the physical standby database instead. This approach will reduce resource utilization on the production database server as well as balance the workload.

Summary

Configuring and managing a VLDB is a challenging task. The DBA needs a set of advanced skills to keep the database application operating smoothly. This chapter addressed the most common recommendations for the basic setup, configuration, performance, and maintenance of very large and extremely large databases.

8. Best Practices for Backup and Recovery with Recovery Manager

Managing database backup and recovery operations remains one of the primary responsibilities of an Oracle DBA. This task includes developing the robust backup and recovery strategies that comply with the service level agreement (SLA). In general, developing a single strategy for all databases is not recommended; you should develop backup and recovery strategies that align with the nature of the application business criticality and demand. Oracle provides various options to perform database backup and recovery operations and to optimize backup and recovery operations to minimize the backup duration. The objective of this chapter is not to provide general concepts, syntax, or examples on how to perform backup and recovery; rather, it proposes the most vital ingredients needed to develop the best backup and recovery strategies for the databases. The chapter also focuses on various database recovery scenarios.

A Perfect Backup and Recovery Plan

Data is a crucial element to any business; that's why organizations invest huge sums of money in state-of-the-art architecture and technologies to ensure data is highly available and protected from all sorts of data losses. In addition, data growth continues to be phenomenal in most organizations, so organizations must carefully plan their backup and recovery plan. There are several options offered by different vendors at both the software and hardware levels to protect the data.

While designing backup and recovery solutions, keep in mind that it is not as simple as backing up the data and doing on-demand recovery. Most important, consequences that could impact the business must be addressed. You must ensure the backup operation doesn't impact the overall performance of the database and the server during the backup and that it doesn't consume huge system resources. You must also consider adherence to the recovery SLA, meet the recovery clauses defined by *recovery point objective* (RPO) and *recovery time objective* (RTO), and achieve almost zero data loss.

When you design and implement database backup strategies, it's important to keep the following crucial recovery factors in mind:

- How much data loss is acceptable?
- How long can the business afford to have the applications that the database supports be inaccessible?
- What is the nature and business criticality of the supported applications?
- What are the possible data loss scenarios, and how can you overcome them?
- How quickly can the database be brought back online, and what is the best and optimal recovery option to achieve that goal?
- What are other possible solutions (i.e., hardware and/or software alternatives) in

case of a prolonged application downtime?

- What are the alternatives available in case your recovery strategy should fail due to unforeseen loss of components necessary for a complete restoration and recovery?

An Overview of RMAN

Oracle offers a variety of solutions to suit every modern business's needs. For business continuity, you can configure Oracle's Data Guard; for high availability, you can configure Oracle Real Application Clusters (RAC). When it comes to protecting the crucial data, Oracle provides multiple tools and utilities to database backups, both logical and physical.

Recovery Manager (RMAN) is a utility that simplifies and performs all sorts of Oracle database-related backup and recovery tasks. Unlike other Oracle advanced features that are separately licensed, RMAN comes at no additional cost and is installed by default. All that you need to do with RMAN is perform some careful planning to back up your databases, test your recovery scenarios, and perhaps do a little tuning to your RMAN configuration.

With RMAN, you can achieve the following goals:

- Create physical backups at various levels: full database, tablespace, or individual datafiles.
- Create compressed and encrypted backups.
- Perform incremental (differential and cumulative) database backups at database and tablespace levels.
- Back up to disk or tape.
- Perform fast incremental backup using *block change tracking* (BCT) to accelerate the backup window.
- Validate the database and its corresponding backups without actually performing recovery operation.
- Clone an existing database for test environments.
- Leverage recovery scenarios at various levels, including a complete database point-in-time recovery, a tablespace point-in-time recovery, and a table-level point-in-time recovery.

Tips for Database Backup Strategies

Each database has a different SLA based on its necessary availability, recoverability, and business criticality; therefore, an Oracle DBA who is responsible for managing variously sized and prioritized databases should design backup and recovery strategies that are appropriate and suits each database's SLA. When you design and implement backup and recovery solutions, it is essential that you seriously consider several important factors, such as backup duration and the server resource consumption versus

the impact on the database's overall performance.

The following key elements should be kept in mind and addressed while designing an appropriate solution:

- The business criticality of the data and its availability
- The RPO and RTO
- The frequency at which backups should be taken to guarantee the database's recoverability
- The size of the database and whether *all* of its components must be available all of the time
- Potential recovery scenarios

Once you have addressed these issues, the next step is to develop an appropriate backup strategy. The subsequent sections offer tips and techniques for constructing the best strategy for your database.

Let's take a close look at some of the best database backup approaches for mission-critical production databases. Once you thoroughly understand the benefits and drawbacks of each strategy, you can choose a strategy that best fits your database's business needs.

Full Backups and Incremental Backups

Under most circumstances for just about any database, it is highly recommended that you perform a full database backup over the weekend and perform incremental backups subsequently through the week. Again, this strategy may not fit for every database, as it will depend on the size of the database; however, for most Oracle databases that are less than 1 TB in size, this strategy is commonly adopted.

For example, the following example shows how to create an RMAN full database backup as an incremental level 0 backup set, which backs up the database and all archive logs to tape, preferably run over the weekend to avoid any business impact.

[Click here to view code image](#)

```
RMAN> run
{
  allocate channel ch1 type 'SBT_TAPE';
  allocate channel ch2 type 'SBT_TAPE';
  BACKUP INCREMENTAL LEVEL=0
    FORMAT 'bk_u%u_s%s_p%p_t%t'
    DATABASE PLUS ARCHIVELOG;
  release channel ch1;
  release channel ch2;
}
```

Compressed Backups

Data compression offers great storage savings. Oracle RMAN supports multiple

compression methods. *Null block compression*, which is the default compression method, is part of incremental level 0 backups, and it ignores all blocks in a tablespace datafile that were never used. In addition, *unused block compression* offers even more storage savings by skipping empty data blocks that currently hold no data, regardless of whether they previously held any data.

RMAN also offers binary compression of the backup files it creates. The basic and medium compression levels require licensing the Oracle Advanced Compression option; the higher settings offer greater and greater compression, but be warned that they also utilize significant CPU during backup.

While null and unused block compression take place by default whenever an incremental level 0 backup of a datafile is performed, binary compression must be defined explicitly within the RMAN run script, as demonstrated in the following example:

[Click here to view code image](#)

```
--- RMAN binary compressed full database online backups
RMAN> run {
  BACKUP
  AS COMPRESSED BACKUPSET
  DATABASE PLUS ARCHIVELOG;
}
```

Decompression takes place automatically during the restore operation; however, the decompression operation will use a significant amount of the server's CPU, which will affect the speed of any recovery operation. It's therefore wise to use the compressed backups option when there is lack of sufficient space to keep the backup files or when backups are created at a remote location over a network with limited bandwidth.

Incremental Backups

Yet another serious concern for backup operations is to reduce the amount of data that needs to be backed up, decreasing the amount of time required to back up. The objective of an incremental level 1 backup is to back up less data and to minimize the time required to capture all changed database blocks during a regularly scheduled backup cycle.

RMAN has the ability to perform incremental backups at two different levels: differential and cumulative. A *differential* incremental backup is the default type, which backs up all data blocks that have been modified since the previous level 0 or level 1 backup. A *cumulative* incremental backup backs up all data blocks that have been modified since the last level 0 backup. In contrast to a differential incremental backup, a cumulative incremental backup consumes more backup space and will probably take additional time too.

On the other hand, when database restore and recovery is performed, cumulative backups require the level 0 backup and only the latest valid cumulative backup for restore, which will offer faster recovery in contrast to differential incremental backups.

If the additional storage and a little extra backup time is not a concern for your business, cumulative incremental backups is the best option considering the recovery factor.

Faster Incremental Backups

Data growth these days can be quite unpredictable; it is not unusual for a database's growth to suddenly accelerate and then continue at an exceptional rate. Backing up a multi-terabyte-sized database can be very time consuming and resource hungry. Although incremental backups help to overcome this situation, Oracle still has to visit every single data block in each datafile to verify whether it needs to be backed up, which could be time consuming. Oracle offers faster incremental backups through its BCT feature. When BCT is enabled, each modified data block is recorded in a binary file outside the database. RMAN refers to the file to identify the data blocks that are candidates for incremental backups, which improves the overall time required for incremental backups.

To enable BCT, use the following command to create and maintain the BCT file in the location specified via the DB_CREATE_FILE_DEST parameter in the target database:

[Click here to view code image](#)

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

Alternatively, to create and maintain the BCT file in a specific location, use the following command:

[Click here to view code image](#)

```
SQL> ALTER DATABASE BLOCK CHANGE TRACKING USING FILE
'<location/filename>' ;
```

Once BCT is enabled, initiate a full database backup before an incremental backup is performed. The following RMAN run block will create a cumulative incremental level 1 backup:

[Click here to view code image](#)

```
# RMAN daily cumulative incremental database online backups
RMAN> run
{
  allocate channel ch1 type 'SBT_TAPE';
  BACKUP
    INCREMENTAL LEVEL=1 CUMULATIVE
    FORMAT 'bk_u%u_s%s_p%p_t%t'
    DATABASE PLUS ARCHIVELOG;
  release channel ch1;
}
```

It is really important to have a valid full database level 0 backup before executing an incremental level 1 backup with BCT.

Rewinding in Oracle Flashback Technology

Oracle Flashback technology offers an additional layer of data protection and provides the ability to quickly “rewind” all or part of a database to a prior state. Oracle Flashback supports a wide range of features:

- Perform an incomplete recovery of the entire database to a prior point in time without actually restoring any datafile with Flashback Database.
- Recover a tablespace set to a prior point in time through tablespace point-in-time recovery (TSPITR).
- Recover a single table to a prior point in time with table point-in-time recovery (new in Oracle Database 12c).
- Rewind all or part of an individual transaction to a prior point in time with Flashback Transaction.

Flashback Database and the ability to set a guaranteed restore point is most useful when you plan to perform database upgrades and while testing different recovery scenarios. It will also help to recover from severe human error by rewinding the database to a consistent state before the error more quickly than an incomplete database recovery operation, because it doesn’t require the restoration of any datafile. (Of course, Flashback Database is not viable when a datafile has been a victim of media corruption or was physically deleted; those issues require regular media recovery techniques to complete the recovery.)

To enable Flashback Database features, the following needs to be done:

- Configure the database’s *fast recovery area* (FRA) by setting appropriate values for its location and size via the DB_RECOVERY_FILE_DEST_SIZE and DB_RECOVERY_FILE_DEST initialization parameters.
- Enable flashback logging by setting the DB_FLASHBACK_RETENTION_TARGET initialization parameter to an appropriate value (the default is 1440 minutes, or one full day).
- Activate Flashback Database by issuing the ALTER DATABASE FLASHBACK ON; command.

Flashback features are quite flexible; in fact, it’s possible to exclude an entire tablespace via the ALTER TABLESPACE <tablespace_name> FLASHBACK OFF; command.

To disable flashback features at the database level, simply issue the ALTER DATABASE FLASHBACK OFF; command.

Disk-Based Backup Solutions

In general, disk-based backup and recovery operations are significantly faster than tape-based backup and recovery. Since disk storage is getting less expensive these days, organizations that can afford sufficient storage for their backups can adopt a disk-based backups solution to achieve better backup and recovery.

Recover Forward Forever

Many Oracle DBAs are not even aware that when they select Oracle's suggested backup strategy through Oracle Enterprise Manager, they are actually choosing to implement its recommended *recover forward forever* (RFF) backup strategy. This strategy—sometimes called *incrementally updatable image copies* (IUIC)—makes it possible to perform an initial incremental level 0 image copy of all datafiles for your database *just one time* and subsequently gather incremental level 1 differential backups for those datafiles.

Once a database's RPO has been satisfied, the oldest incremental level 1 backups are applied to the initial image copy, thus recovering it forward by just one day. This cycle is repeated for the remainder of the database's life. The advantage of this process is obvious: the initial level 0 image copy will never need to be taken again, and only level 1 backups will need to be taken, thus significantly reducing the amount of time required to perform database backups.

During a complete media recovery operation, the required datafiles are restored from the most recent image copy of the datafiles, level 1 incremental backups are used to roll the datafiles forward as far as possible, and then archived redo and online redo logs are applied to bring the datafiles to the current SCN. For an incomplete recovery operation, this same technique can be leveraged except that recovery is terminated short of the current SCN. This strategy helps to minimize the overall database restore and recovery window and is much faster than traditional recovery methods.

Here is an example of an RMAN run block that implements the nightly backup processing for the RFF strategy:

[Click here to view code image](#)

```
RMAN> run
{
  RECOVER COPY OF DATABASE WITH TAG 'weekly_iud';
  BACKUP
    INCREMENTAL LEVEL 1
    FOR RECOVER OF COPY WITH TAG 'weekly_iud'
    DATABASE
    PLUS ARCHIVELOG;
}
```

One perceived disadvantage of this backup method is that the image copies of the datafiles require the same amount of storage as the database itself. However, this method does offer the fastest way to recover from loss of a datafile because it's a simple matter to switch to the image copy stored in the FRA and perform a media recovery using that datafile.

The example that follows shows how RFF would be implemented over three separate backup cycles:

[Click here to view code image](#)

```
#####
# Results from 1st run:
```

```
# - Incremental Level 0 image copies of all datafiles now exist
# because even though a Level 1 incremental backup was requested,
# no Level 0 incremental backup has yet been created with the tag
# of img_cpy_upd.
#####
```

```
RMAN> # Implement Incrementally-Updateable Image Copy (IUIC)
strategy
2> # for RPO of 24 hours (i.e., RECOVERY WINDOW OF ONE DAY)
3> RUN {
4>   RECOVER
5>     COPY OF DATABASE
6>       WITH TAG 'img_cpy_upd';
7>   BACKUP
8>     INCREMENTAL LEVEL 1
9>     FOR RECOVER OF COPY WITH TAG 'img_cpy_upd'
10>    SECTION SIZE 100M
11>    DATABASE
12>    PLUS ARCHIVELOG DELETE INPUT;
13> }
```

```
Starting recover at 2014-04-07 13:17:07
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=496 device type=DISK
allocated channel: ORA_DISK_2
channel ORA_DISK_2: SID=13 device type=DISK
allocated channel: ORA_DISK_3
channel ORA_DISK_3: SID=38 device type=DISK
allocated channel: ORA_DISK_4
channel ORA_DISK_4: SID=486 device type=DISK
no copy of datafile 1 found to recover
no copy of datafile 2 found to recover
no copy of datafile 3 found to recover
no copy of datafile 4 found to recover
no copy of datafile 5 found to recover
no copy of datafile 6 found to recover
no copy of datafile 7 found to recover
no copy of datafile 8 found to recover
no copy of datafile 9 found to recover
no copy of datafile 10 found to recover
no copy of datafile 11 found to recover
no copy of datafile 12 found to recover
no copy of datafile 13 found to recover
no copy of datafile 14 found to recover
no copy of datafile 15 found to recover
Finished recover at 2014-04-07 13:17:08
```

```
Starting backup at 2014-04-07 13:17:09
current log archived
using channel ORA_DISK_1
using channel ORA_DISK_2
```

```

using channel ORA_DISK_3
using channel ORA_DISK_4
channel ORA_DISK_1: starting archived log backup set
channel ORA_DISK_1: specifying archived log(s) in backup set
input archived log thread=1 sequence=508 RECID=9 STAMP=844262229
channel ORA_DISK_1: starting piece 1 at 2014-04-07 13:17:09
channel ORA_DISK_1: finished piece 1 at 2014-04-07 13:17:10
piece
handle=+FRA/NCDB121/BACKUPSET/2014_04_07/annnf0_img_cpy_upd_0.307.8'
tag=IMG_CPY_UPD comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:02
channel ORA_DISK_1: deleting archived log(s)
archived log file
name=+FRA/NCDB121/ARCHIVELOG/2014_04_07/thread_1_seq_508.272.844262:
RECID=9 STAMP=844262229
Finished backup at 2014-04-07 13:17:11

```

Starting backup at 2014-04-07 13:17:11

```

using channel ORA_DISK_1
using channel ORA_DISK_2
using channel ORA_DISK_3
using channel ORA_DISK_4
no parent backup or copy of datafile 3 found
no parent backup or copy of datafile 1 found
no parent backup or copy of datafile 4 found
no parent backup or copy of datafile 12 found
no parent backup or copy of datafile 2 found
no parent backup or copy of datafile 10 found
no parent backup or copy of datafile 14 found
no parent backup or copy of datafile 15 found
no parent backup or copy of datafile 8 found
no parent backup or copy of datafile 9 found
no parent backup or copy of datafile 11 found
no parent backup or copy of datafile 13 found
no parent backup or copy of datafile 5 found
no parent backup or copy of datafile 7 found
no parent backup or copy of datafile 6 found
channel ORA_DISK_1: starting datafile copy
input datafile file number=00003
name=+DATA/NCDB121/DATAFILE/sysaux.283.836524477
channel ORA_DISK_2: starting datafile copy
input datafile file number=00001
name=+DATA/NCDB121/DATAFILE/system.282.836524551
channel ORA_DISK_3: starting datafile copy
input datafile file number=00004
name=+DATA/NCDB121/DATAFILE/undotbs1.284.836524619
channel ORA_DISK_4: starting datafile copy
input datafile file number=00012
name=+FRA/NCDB121/DATAFILE/ado_cold_data.283.841267371
output file name=+FRA/NCDB121/DATAFILE/ado_cold_data.274.844262235

```

```

tag=IMG_CPY_UPD RECID=55 STAMP=844262264
channel ORA_DISK_4: datafile copy complete, elapsed time: 00:00:35
channel ORA_DISK_4: starting datafile copy
input datafile file number=00002
name=+DATA/NCDB121/DATAFILE/example.280.836524747
output file name=+FRA/NCDB121/DATAFILE/undotbs1.303.844262235
tag=IMG_CPY_UPD RECID=56 STAMP=844262271
channel ORA_DISK_3: datafile copy complete, elapsed time: 00:00:41
...
{ omitted for sake of brevity }
...
piece
handle=+FRA/NCDB121/BACKUPSET/2014_04_07/nnsnn1_img_cpy_upd_0.295.8-
tag=IMG_CPY_UPD comment=NONE
channel ORA_DISK_4: backup set complete, elapsed time: 00:00:02
Finished backup at 2014-04-07 13:18:42

```

```

Starting backup at 2014-04-07 13:18:42
current log archived
using channel ORA_DISK_1
using channel ORA_DISK_2
using channel ORA_DISK_3
using channel ORA_DISK_4
channel ORA_DISK_1: starting archived log backup set
channel ORA_DISK_1: specifying archived log(s) in backup set
input archived log thread=1 sequence=509 RECID=10 STAMP=844262322
channel ORA_DISK_1: starting piece 1 at 2014-04-07 13:18:43
channel ORA_DISK_1: finished piece 1 at 2014-04-07 13:18:44
piece
handle=+FRA/NCDB121/BACKUPSET/2014_04_07/annnf0_img_cpy_upd_0.304.8-
tag=IMG_CPY_UPD comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:01
channel ORA_DISK_1: deleting archived log(s)
archived log file
name=+FRA/NCDB121/ARCHIVELOG/2014_04_07/thread_1_seq_509.271.844262:
RECID=10 STAMP=844262322
Finished backup at 2014-04-07 13:18:45

```

```

#####
# Results from 2nd run:
# - Two new tablespaces have been created to show how they will be
wrapped
#   into the IUIC strategy automatically (as Incremental Level 0
image
#   copy backups).
# - Incremental Level 0 image copies of all datafiles now exist,
but since
#   no Incremental Level 1 image copy backups yet exist with a tag
of
#   img_cpy_upd, none will be applied to the existing Level 0

```

```

backups.
# - Incremental Level 1 backups of all datafiles will be taken
because
# Incremental Level 0 backups (their eventual "parents") now
exist.
#####

# Create new tablespaces:
CREATE TABLESPACE rollon_staging_data
  DATAFILE '+DATA'
  SIZE 200M
  NOLOGGING;

CREATE TABLESPACE rollon_staging_idx
  DATAFILE '+DATA'
  SIZE 100M
  NOLOGGING;

RMAN> # Implement Incrementally-Updateable Image Copy (IUIC)
strategy
2> # for RPO of 24 hours (i.e., RECOVERY WINDOW OF ONE DAY)
3> RUN {
4>   RECOVER
5>     COPY OF DATABASE
6>     WITH TAG 'img_cpy_upd';
7>   BACKUP
8>     INCREMENTAL LEVEL 1
9>     FOR RECOVER OF COPY WITH TAG 'img_cpy_upd'
10>    SECTION SIZE 100M
11>    DATABASE
12>    PLUS ARCHIVELOG DELETE INPUT;
13> }

Starting backup at 2014-04-07 13:29:50
using channel ORA_DISK_1
using channel ORA_DISK_2
using channel ORA_DISK_3
using channel ORA_DISK_4
no parent backup or copy of datafile 16 found
no parent backup or copy of datafile 17 found
channel ORA_DISK_1: starting incremental level 1 datafile backup
set
channel ORA_DISK_1: specifying datafile(s) in backup set
input datafile file number=00003
name=+DATA/NCDB121/DATAFILE/sysaux.283.836524477
backing up blocks 1 through 12800
channel ORA_DISK_1: starting piece 1 at 2014-04-07 13:29:52
channel ORA_DISK_2: starting incremental level 1 datafile backup
set
channel ORA_DISK_2: specifying datafile(s) in backup set

```

```

input datafile file number=00001
name=+DATA/NCDB121/DATAFILE/system.282.836524551
backing up blocks 1 through 12800
channel ORA_DISK_2: starting piece 1 at 2014-04-07 13:29:53
channel ORA_DISK_3: starting incremental level 1 datafile backup
set
channel ORA_DISK_3: specifying datafile(s) in backup set
input datafile file number=00004
name=+DATA/NCDB121/DATAFILE/undotbs1.284.836524619
backing up blocks 1 through 12800
channel ORA_DISK_3: starting piece 1 at 2014-04-07 13:29:53
channel ORA_DISK_4: starting incremental level 1 datafile backup
set
channel ORA_DISK_4: specifying datafile(s) in backup set
input datafile file number=00012
name=+FRA/NCDB121/DATAFILE/ado_cold_data.283.841267371
backing up blocks 1 through 12800
channel ORA_DISK_4: starting piece 1 at 2014-04-07 13:29:55
channel ORA_DISK_1: finished piece 1 at 2014-04-07 13:29:55
piece
. . .
{ omitted for sake of brevity }
. . .
handle=+FRA/NCDB121/BACKUPSET/2014_04_07/nnndn1_img_cpy_upd_0.279.8·
    tag=IMG_CPY_UPD comment=NONE
channel ORA_DISK_2: backup set complete, elapsed time: 00:00:01
channel ORA_DISK_4: finished piece 1 at 2014-04-07 13:30:38
piece
handle=+FRA/NCDB121/BACKUPSET/2014_04_07/ncnnn1_img_cpy_upd_0.356.8·
    tag=IMG_CPY_UPD comment=NONE
channel ORA_DISK_4: backup set complete, elapsed time: 00:00:01
Finished backup at 2014-04-07 13:30:38

Starting backup at 2014-04-07 13:30:39
current log archived
using channel ORA_DISK_1
using channel ORA_DISK_2
using channel ORA_DISK_3
using channel ORA_DISK_4
channel ORA_DISK_1: starting archived log backup set
channel ORA_DISK_1: specifying archived log(s) in backup set
input archived log thread=1 sequence=515 RECID=16 STAMP=844263040
channel ORA_DISK_1: starting piece 1 at 2014-04-07 13:30:40
channel ORA_DISK_1: finished piece 1 at 2014-04-07 13:30:41
piece
handle=+FRA/NCDB121/BACKUPSET/2014_04_07/annnf0_img_cpy_upd_0.359.8·
    tag=IMG_CPY_UPD comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:01
channel ORA_DISK_1: deleting archived log(s)
archived log file

```

```
name=+FRA/NCDB121/ARCHIVELOG/2014_04_07/thread_1_seq_515.358.8442631
RECID=16 STAMP=844263040
Finished backup at 2014-04-07 13:30:43
```

```
#####
# Results from 3rd run:
# - The Incremental Level 1 backups from the 1st night's backups
# will now
#   be applied to their Incremental Level 0 image copy "parents."
# - Incremental Level 1 backups of any changed datafiles will be
# taken.
# - The Incremental Level 0 image copy backup for the two new
# datafiles
#   created during the 2nd night's run will not yet be updated from
#   Incremental Level 1 backups until the next night's run.
#####
#
```

```
RMAN> # Implement Incrementally-Updateable Image Copy (IUIC)
strategy
2> # for RPO of 24 hours (i.e., RECOVERY WINDOW OF ONE DAY)
3> RUN {
4>   RECOVER
5>     COPY OF DATABASE
6>       WITH TAG 'img_cpy_upd';
7>   BACKUP
8>     INCREMENTAL LEVEL 1
9>     FOR RECOVER OF COPY WITH TAG 'img_cpy_upd'
10>    SECTION SIZE 100M
11>    DATABASE
12>    PLUS ARCHIVELOG DELETE INPUT;
13> }
13> }
```

Starting recover at 2014-04-07 14:27:31

```
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=496 device type=DISK
allocated channel: ORA_DISK_2
channel ORA_DISK_2: SID=13 device type=DISK
allocated channel: ORA_DISK_3
channel ORA_DISK_3: SID=38 device type=DISK
allocated channel: ORA_DISK_4
channel ORA_DISK_4: SID=486 device type=DISK
no copy of datafile 16 found to recover
no copy of datafile 17 found to recover
channel ORA_DISK_1: starting incremental datafile backup set
restore
channel ORA_DISK_1: specifying datafile copies to recover
recovering datafile copy file number=00003
name=+FRA/NCDB121/DATAFILE/sysaux.353.844262767
channel ORA_DISK_1: restoring section 1 of 15
```

```

channel ORA_DISK_1: reading from backup piece
  +FRA/NCDB121/BACKUPSET/2014_04_07/nnndnl_img_cpy_upd_0.338.84421
channel ORA_DISK_2: starting incremental datafile backup set
restore
channel ORA_DISK_2: specifying datafile copies to recover
recovering datafile copy file number=00001
name=+FRA/NCDB121/DATAFILE/system.357.844262767
channel ORA_DISK_2: restoring section 1 of 8
channel ORA_DISK_2: reading from backup piece
  +FRA/NCDB121/BACKUPSET/2014_04_07/nnndnl_img_cpy_upd_0.267.84421
channel ORA_DISK_3: starting incremental datafile backup set
restore
channel ORA_DISK_3: specifying datafile copies to recover
recovering datafile copy file
  number=00004 name=+FRA/NCDB121/DATAFILE/undotbs1.354.844262767
channel ORA_DISK_3: restoring section 1 of 6
channel ORA_DISK_3: reading from backup piece
  +FRA/NCDB121/BACKUPSET/2014_04_07/nnndnl_img_cpy_upd_0.268.84421
...
{ omitted for sake of brevity }
...
{ omitted for sake of brevity }
...
channel ORA_DISK_4: piece handle=
  +FRA/NCDB121/BACKUPSET/2014_04_07/nnndnl_img_cpy_upd_0.303.84421
tag=IMG_CPY_UPD
channel ORA_DISK_4: restored backup piece 1
channel ORA_DISK_4: restore complete, elapsed time: 00:00:01
channel ORA_DISK_2: piece handle=
  +FRA/NCDB121/BACKUPSET/2014_04_07/nnndnl_img_cpy_upd_0.279.84421
tag=IMG_CPY_UPD
channel ORA_DISK_2: restored backup piece 1
channel ORA_DISK_2: restore complete, elapsed time: 00:00:00
Finished recover at 2014-04-07 14:27:49

```

```

Starting backup at 2014-04-07 14:27:50
current log archived
using channel ORA_DISK_1
using channel ORA_DISK_2
using channel ORA_DISK_3
using channel ORA_DISK_4
channel ORA_DISK_1: starting archived log backup set
channel ORA_DISK_1: specifying archived log(s) in backup set
input archived log thread=1 sequence=516 RECID=17 STAMP=844266471
channel ORA_DISK_1: starting piece 1 at 2014-04-07 14:27:51
channel ORA_DISK_1: finished piece 1 at 2014-04-07 14:27:52
piece
handle=+FRA/NCDB121/BACKUPSET/2014_04_07/annnf0_img_cpy_upd_0.360.8
  tag=IMG_CPY_UPD comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:01

```

```

channel ORA_DISK_1: deleting archived log(s)
archived log file name=
  +FRA/NCDB121/ARCHIVELOG/2014_04_07/thread_1_seq_516.358.8442664'
    RECID=17 STAMP=844266471
Finished backup at 2014-04-07 14:27:52

Starting backup at 2014-04-07 14:27:52
using channel ORA_DISK_1
using channel ORA_DISK_2
using channel ORA_DISK_3
using channel ORA_DISK_4
no parent backup or copy of datafile 3 found
no parent backup or copy of datafile 1 found
no parent backup or copy of datafile 4 found
no parent backup or copy of datafile 12 found
no parent backup or copy of datafile 2 found
no parent backup or copy of datafile 10 found
no parent backup or copy of datafile 14 found
no parent backup or copy of datafile 15 found
no parent backup or copy of datafile 8 found
no parent backup or copy of datafile 9 found
no parent backup or copy of datafile 11 found
no parent backup or copy of datafile 13 found
channel ORA_DISK_1: starting datafile copy
input datafile file number=00003
name=+DATA/NCDB121/DATAFILE/sysaux.283.836524477
channel ORA_DISK_2: starting datafile copy
input datafile file number=00001
name=+DATA/NCDB121/DATAFILE/system.282.836524551
channel ORA_DISK_3: starting datafile copy
input datafile file number=00004
name=+DATA/NCDB121/DATAFILE/undotbs1.284.836524619
channel ORA_DISK_4: starting datafile copy
input datafile file number=00012
name=+FRA/NCDB121/DATAFILE/ado_cold_data.283.841267371
output file name=+FRA/NCDB121/DATAFILE/ado_cold_data.363.844266475
  tag=IMG_CPY_UPD RECID=114 STAMP=844266520
channel ORA_DISK_4: datafile copy complete, elapsed time: 00:00:56
...
{ omitted for sake of brevity }
...
{ omitted for sake of brevity }
...
handle=+FRA/NCDB121/BACKUPSET/2014_04_07/ncnnn1_img_cpy_upd_0.293.8'
tag=IMG_CPY_UPD
  comment=NONE
channel ORA_DISK_2: backup set complete, elapsed time: 00:00:01
Finished backup at 2014-04-07 14:30:31

Starting backup at 2014-04-07 14:30:31

```

```
current log archived
using channel ORA_DISK_1
using channel ORA_DISK_2
using channel ORA_DISK_3
using channel ORA_DISK_4
channel ORA_DISK_1: starting archived log backup set
channel ORA_DISK_1: specifying archived log(s) in backup set
input archived log thread=1 sequence=517 RECID=18 STAMP=844266631
channel ORA_DISK_1: starting piece 1 at 2014-04-07 14:30:31
channel ORA_DISK_1: finished piece 1 at 2014-04-07 14:30:32
piece
handle=+FRA/NCDB121/BACKUPSET/2014_04_07/annnf0_img_cpy_upd_0.301.8-
tag=IMG_CPY_UPD
comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:01
channel ORA_DISK_1: deleting archived log(s)
archived log file
name=+FRA/NCDB121/ARCHIVELOG/2014_04_07/thread_1_seq_517.270.8442661
RECID=18 STAMP=844266631
Finished backup at 2014-04-07 14:30:33
```

Validating RMAN Backups

You've developed an appropriate backup strategy for your database using RMAN, and you've performed sufficient backups to guarantee your database's RPO. However, you could still encounter a serious situation in which recovery can't be performed as you intended due to issues with your existing backup copies. For example, what if a crucial image copy backup or backup set were corrupted or damaged *after* their successful creation? To avoid any such situation, it is highly recommended that you test your backups frequently. In an ideal situation, testing is accomplished by restoring the backups to another host and performing a full database recovery scenario; however, this method of testing is sometimes infeasible due to the lack of facilities.

RMAN is completely capable of validating that your backups as well as your existing Oracle database are free of any possible logical or physical block corruption. The validation can be performed without actually performing the restoration of the backups and/or recovery of the database, and you will be assured of having reliable backups that can be used for database restoration and recovery.

To validate your database's control files, all datafiles, and SPFILE (if one exists), use the following command in RMAN:

```
RMAN> VALIDATE DATABASE;
```

The following RMAN command will validate just one datafile—in this case, one of the datafiles for the SYSTEM tablespace:

```
RMAN> VALIDATE DATAFILE 1;
```

Use VALIDATE BACKUPSET when you suspect that one or more backup pieces in a

backup set are missing or have been damaged. The following command checks every block in a backup set to ensure that the backup can be restored:

[Click here to view code image](#)

```
RMAN> run
{
    allocate channel ch1 type 'SBT_TAPE';
    VALIDATE BACKUPSET <backupset_number>;
    release channel ch1;
}
```

The following RMAN command script will validate that datafiles or archived redo logs that have been backed up contain no logical *or* physical corruption:

[Click here to view code image](#)

```
RMAN> run {
    BACKUP VALIDATE
    CHECK LOGICAL
    DATABASE AND ARCHIVELOG ALL;
}
```

Backup Optimization and Tuning

Having a good backup and recovery strategy to protect an organization's precious data is undoubtedly one of the key goals of an Oracle DBA. At the same time, it is very important to minimize the performance impact of your backup strategy. Here are some basic optimization and tuning principles for database backup and restore/recovery operations that will help in reducing the required backup window and improve the performance of recovery operations:

- Split backups for extremely large datafiles into multiple sections by specifying an appropriate value for the SECTION SIZE parameter. For example, this RMAN command leverages multisection backups to divide a datafile backup into multiple sections of 2 GB, which will boost the speed of both backup and restore operations:

[Click here to view code image](#)

```
RMAN> BACKUP SECTION SIZE 2G TABLESPACE <tablespace_name>;
```

- Use the BACKUP OPTIMIZATION setting in RMAN to skip any datafile during backup or restore operations when certain conditions are met. For example, during backup, if the datafile has not changed its SCN from its last backup, RMAN won't back up the file. Similarly, during datafile restoration, if a datafile exists in its location with a matching SCN of the datafile being restored from backup, RMAN won't bother to attempt to restore the datafile.
- To restrict the backup window to avoid any performance pitfalls on the server for long-running backups, use BACKUP DURATION clause.
- Review dynamic views V\$BACKUP_ASYNC_IO and V\$BACKUP_SYNC_IO to

identify any performance bottlenecks and learn about the status of ongoing and recently performed backup and recovery operations.

- If your operating system doesn't support asynchronous I/O, set a nonzero value for the DBWR_IO_SLAVE initialization parameter.
- Adjust the tape I/O buffer size if you are creating backups on tape media or using a virtual tape library.
- Ensure you have set the LARGE_POOL_SIZE initialization parameter to a sufficient size, especially when backing up directly to tape. Use the formula $LARGE_POOL_SIZE = \text{number_of_allocated_channels} * (16\text{MB} + (4 * \text{size_of_buffer_tape}))$ to determine the optimal size.
- Allocate multiple channels to improve the database backup performance.
- Generate an Automatic Workload Repository (AWR) report and verify the top wait events to identify any RMAN backup related wait events—for example, sbtwrite2, sbtbackup, and so on.
- Finally, when all else fails during diagnosis of performance issues, consider using the DEBUG and TRACE parameters when invoking RMAN to get detailed information about the issue to analyze the bottleneck:

[Click here to view code image](#)

```
RMAN> CONFIGURE CHANNEL DEVICE TYPE DISK DEBUG=5 TRACE 1;
```

Tuning Disk-Based Backup Performance

To improve the performance of disk-based backups, you need to fine tune the RMAN memory-related hidden initialization parameters: _backup_disk_bufsz, _backup_disk_bufcnt, _backup_file_bufcnt, and _backup_file_bufsz. The values can be set dynamically in the RMAN's run script or set at database level.

The following query displays the list of hidden parameters pertaining to RMAN:

[Click here to view code image](#)

```
SQL> select
      ksppinm,
      ksppdesc
  from
      x$ksppi
 where
     substr(ksppinm,1,5) = '_back'
 order by
     1,2;
```

KSPPINM	KSPPDESC
_backup_align_write_io	align backup write I/Os

_backup_disk_bufcnt	number of buffers used for DISK
channels	
_backup_disk_bufsz	size of buffers used for DISK
channels	
_backup_disk_io_slaves	BACKUP Disk I/O slaves
_backup_dynamic_buffers	dynamically compute backup/restore
buffer sizes	
_backup_encrypt_opt_mode	specifies encryption block
optimization mode	
_backup_file_bufcnt	number of buffers used for file
access	
_backup_file_bufsz	size of buffers used for file access
_backup_io_pool_size	memory to reserve from the large
pool	
_backup_kgc_blk siz	specifies buffer size to be used by
HIGH compression	
_backup_kgc_bufsz	specifies buffer size to be used by
BASIC compression	
_backup_kgc_memlevel	specifies memory level for MEDIUM
compression	
_backup_kgc_niters	specifies number of iterations done
by BASIC compression	
_backup_kgc_perflevel	specifies compression (performance)
level for MEDIUM compression	
_backup_kgc_scheme	specifies compression scheme
_backup_kgc_type	specifies compression type used by
kgc BASIC compression	
_backup_kgc_windowbits	specifies window size for MEDIUM
compression	
_backup_ksfq_bufcnt	number of buffers used for
backup/restore	
_backup_ksfq_bufcnt_max	maximum number of buffers used for
backup/restore	
_backup_ksfq_bufsz	size of buffers used for
backup/restore	
_backup_lzo_size	specifies buffer size for LOW
compression	
_backup_max_gap_size	largest gap in an
incremental/optimized backup	buffer, in bytes
_backup_seq_bufcnt	number of buffers used for non-DISK
channels	
_backup_seq_bufsz	size of buffers used for non-DISK
channels	

Using RMAN for RAC Databases

Managing Oracle database backup and recovery operations through RMAN is identical for RAC and non-RAC database environments, with just a few exceptions. In a RAC environment, RMAN backup workloads can be distributed evenly across multiple instances of the RAC database to utilize node resources efficiently.

Before configuring load-balance settings for a RAC database, enable parallelism as follows:

[Click here to view code image](#)

```
RMAN> configure device type [sbt|disk] parallelism 2;
```

The following example demonstrates automatic load balancing of RMAN database backups in a RAC database with two instances. First, create a database service for load balancing:

[Click here to view code image](#)

```
$ srvctl add service -d RDB -s rdb_main -r rdb1,rdb2
$ srvctl start service -d RDB -s rdb_main
```

Next, add entries to each node's tnsnames.ora network configuration file to leverage the new service name for load balancing:

[Click here to view code image](#)

```
RDB_MAIN =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP) (HOST = rdbscan) (PORT = 1522))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = RDB_MAIN)
    )
  )
```

The following example demonstrates an RMAN database backup script that uses a dynamic channel allocation for load balancing:

[Click here to view code image](#)

```
RMAN> run
{
  allocate channel ch1 connect 'sys/password@rdb_main';
  backup database;
}
```

Alternatively, you can allocate dedicated channels to each instance. The following example entries for tnsnames.ora permits dedicated channel allocation:

[Click here to view code image](#)

```
RDB1 =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP) (HOST = prdn1) (PORT = 1522))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = RDB)
    )
  )
```

```

)
RDB2 =
(DESCRIPTION =
(ADDRESS_LIST =
(ADDRESS = (PROTOCOL = TCP) (HOST = prdn22) (PORT = 1522))
)
(CONNECT_DATA =
(SERVICE_NAME = RDB)
)
)
)

```

And here is an RMAN database backup script that demonstrates how to allocate specific channels to specific RAC instances for load balancing:

[Click here to view code image](#)

```

RMAN> run
{
    allocate channel ch1 connect 'sys/password@rdb1';
    allocate channel ch2 connect 'sys/password@rdb2';
    backup database;
}

```

Load balancing RMAN database backup workload across multiple instances can help prevent resource bottlenecks on the nodes while providing efficient use of resources on the node.

Finally, when using these techniques within multiple instances of a RAC database, don't forget that it is essential to store the snapshot control file in a shared location—preferably in an Automatic Storage Management (ASM) disk group—to prevent an “ORA-00245 control file backup operation failed” error and backup failures:

[Click here to view code image](#)

```
RMAN> configure snapshot controlfile name to '+DG_FRA\snapcf_RDB';
```

Retaining Data in a Recovery Catalog

By default, RMAN stores all backup and recovery operational history in the database's control file and retains it for seven days. To retain the backup and restore operational information for longer than the default timeframe, you can increase the `CONTROL_FILE_RECORD_KEEP_TIME` parameter value to a longer period, but no more than 365 days. If you are planning to retain RMAN backups for a period longer than 365 days and keep them readily available for restoration and recovery purposes, the only alternative is to create an RMAN *recovery catalog*. This catalog is essentially a copy of the RMAN information retained in the database's control file, and it stores the metadata of database backup and recovery operations for a much longer period of time.

Another advantage of the recovery catalog is that if you were to lose all copies of your database's control files, you can still restore and recover the database using the

information stored in the catalog, including the control file. Additionally, you can develop RMAN scripts and store them in the catalog to use with different databases. Finally, if you are using Oracle Data Guard as your disaster recovery solution and plan to offload your backup processing to run against a corresponding physical standby database, a recovery catalog is required to capture all backup information to enable the primary database to leverage those backups.

Having a Robust Recovery Strategy

You've prepared a robust backup strategy, you've tuned your backup's performance so that it's optimal, and perhaps you've even simulated a disaster recovery via Oracle Data Guard. But if you can't handle the recovery scenario that is thrown at you, then all of this preparation is virtually useless. It's therefore incumbent on you to be alert and ready for any recovery situation that could arise from any vector. Following are some of the most common recovery scenarios that you might encounter and that you must be prepared to handle:

- Incomplete recovery of an entire database
- Complete database recovery with zero data loss
- Recovery of corrupted or missing control files due to loss of all control file copies
- Datafile-level and tablespace-level recovery
- Tablespace point-in-time recovery
- Recovery of a single table or table partition
- Recovery of one or more database blocks
- Recovery of a lost, damaged, or missing SPFILE
- Loss of one or more online redo log members
- Loss of an entire online redo log group (especially the *current* group!)
- Complete disaster recovery when the *entire* database—all datafiles, control file copies, and SPFILE—is lost
- Hardware, firmware, or software failures (including partial or complete loss of an Oracle Clusterware, Grid Infrastructure, or database home and related binary files)

RMAN provides you the ability to test your recovery scenario without actually performing the recovery operation. The following command verifies existing backups and confirms the smooth recovery:

[Click here to view code image](#)

```
RMAN> RESTORE DATABASE VALIDATE;  
RMAN> RESTORE ARCHIVELOG ALL VALIDATE;
```

The following provides best practices for a faster database recovery:

- Use a proper value for FILESPERSET, MAXOPENFILES, and

MAXPIECESIZE.

- Ensure that you turned on the BACKUP OPTIMIZATION setting to avoid restoring datafiles that don't need to be restored.
- Set the LARGE_POOL_SIZE initialization parameter value as per the recommendation previously mentioned to speed up restoration of all database files.
- Enable channels parallelism to quickly restore many database files. For example, use the following CONFIGURE setting to specify the use of four parallel channels for all disk-based backup and restore operations:

[Click here to view code image](#)

```
RMAN> CONFIGURE DEVICE TYPE DISK PARALLELISM 8;
```

- Leverage parallelism to quickly recover datafiles. For example, use the following RMAN command to specify the use of eight degrees of parallelism during application of redo entries during recovery:

[Click here to view code image](#)

```
RMAN> recover database parallel 8;
```

- Use the following query to monitor the progress of the recovery operation:

[Click here to view code image](#)

```
SQL> SELECT name,item,sofar,total FROM v$recovery_progress;
```

Leveraging the Data Recovery Advisor

Not only is an Oracle DBA often tasked with managing multiple databases at varying release levels, she may also encounter a complex recovery situation that she has never encountered before but that must be resolved immediately to restore application access to a crucial database. This pressure may lead her to choose a recovery solution that takes an inordinate amount of time, or even worse, choose a totally incorrect solution.

The good news is that starting in Oracle Database 11g, the *Data Recovery Advisor* (DRA) tool assists in determining, detecting, and suggesting appropriate data repair options. The DRA tool is configured automatically and is quite simple to use. For example, the LIST FAILURE command shows the most recently detected failures:

```
RMAN> LIST FAILURE;
```

```
<example output>?
```

Once the DBA has decided which failure to repair, she can simply request RMAN to diagnose the failure and provide cogent advice:

```
RMAN> ADVISE FAILURE;
```

She can ask DRA to preview exactly what actions it will take to repair the detected

problems:

```
RMAN> REPAIR FAILURE PREVIEW;
```

Finally, she can even ask DRA to execute the recommended repair instructions:

```
RMAN> REPAIR FAILURE;
```

Note

As of this writing, DRA supports only single-instance databases, so if you attempt to use it against a RAC database, you will encounter the following error:

[Click here to view code image](#)

```
RMAN-00571:
```

```
=====
```

```
RMAN-00569: ====== ERROR MESSAGE STACK FOLLOWS
```

```
=====
```

```
RMAN-00571:
```

```
=====
```

```
RMAN-03002: failure of list command at 04/13/2015 13:58:43
```

```
RMAN-05533: LIST FAILURE is not supported on RAC database
```

Summary

This chapter explained the importance of adhering to the recommended best practices for Oracle database backup, restoration, and recovery, as well as how to design and develop appropriate strategies to meet your organization's business demands based on the database's RPO and RTO. We also discussed how to test and validate your backups to ensure smooth recoverability. The chapter also offered some simple backup and recovery performance tuning principles, including leveraging DRA for quick and accurate diagnosis of database recovery issues.

Here is a quick wrap-up of the top Oracle backup and recovery recommendations to fine-tune your database backup and recovery operations:

- Configure an appropriately-sized FRA for each database.
- Turn on BCT for faster incremental backups.
- Validate your backups with RESTORE DATABASE VALIDATE and RESTORE ARCHIVE LOG ALL commands.
- If no RMAN catalog is configured, increase the default for CONTROL_FILE_RECORD_KEEP_TIME to a higher value.
- Always turn on control file auto backups with the CONFIGURE CONTROLFILE AUTOBACKUP ON; command.
- Turn on CONFIGURE BACKUP OPTIMIZATION to avoid any unnecessary backup and restore operations.
- Before performing a database recovery, review and evaluate RMAN's planned

recovery via the RECOVER DATABASE TEST command.

- Validate recovery procedure and requirements with RECOVER DATABASE <DBNAME>.
- Exclude read-only tablespaces once you back up.
- When backing up a RAC database, remember to store the snapshot control file in a shared location.
- Distribute the backup workload across RAC database instances.
- Manage the length of the backup window with the BACKUP DURATION clause.
- Consider increasing the _BACKUP_KSFQ_BUFCNT and _BACKUP_KSFQ_BUFSZ parameters when restoring datafiles to ASM disk groups.
- Review the V\$RMAN_STATUS dynamic view to monitor the backup status and backup start/end times.
- On a regular basis, test your backup strategy via restore and recovery operations to a different database server.
- Delete obsolete and expired backup copies either manually or automatically by configuring recovery window and FRA policies.
- Minimize the number of datafiles per backup set.
- Limit the RMAN backupset size to avoid having a huge backup set size.
- Don't rely on the default RMAN configuration; try to tune the configuration according to need.
- Finally, the most important recommendation is to *test, retest, and retest again* your backup strategy against all possible recovery scenarios!

9. Database Forensics and Tuning Using AWR Analysis: Part I

Oracle utilities such as Statspack and Automatic Workload Repository (AWR) reports provide the database administrator (DBA) or tuner detailed information concerning a snapshot of database execution time. This snapshot provides statistics on wait events, latches, storage input and output volumes and timings, as well as various views of memory and SQL activities.

The statistics and insights into the memory, I/O, and SQL performance characteristics are invaluable aids in determining whether a database is functioning optimally. Unfortunately, there is such a plethora of data in most AWR reports that DBAs often feel overwhelmed and may miss important clues about database performance issues.

This chapter provides a guide to interpreting the AWR results so that even a novice DBA can glean valid, actionable results from review of an AWR report.

What Is AWR?

The AWR provides a set of tables in which snapshots of system statistics are kept. Generally, these snapshots are taken on an hourly basis, but the DBA can use the DBMS_WORKLOAD_REPOSITORY package to change internal settings. Following is an example:

[Click here to view code image](#)

```
BEGIN
    DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS (
        retention => 44640, interval => 15, topnsql => 50,
        dbid => 6611949349);
END;
/
```

This code would reset the retention of data from the default of 7 days to 31 days, the interval between collection of datasets from every 60 minutes to every 15 minutes, and the number of SQL statements collected to the top 50 statements in each category for the database whose ID is 6611949349. The dbid doesn't have to be included if you are modifying the default instance.

The statistics gathered include wait interface statistics, top SQL statements, tablespace- and object-level I/O and memory, and cumulative I/O information up to the time of the capture. The basic AWR report process takes the cumulative data from two snapshots, subtracts the earlier snapshot's cumulative data from the later snapshot, and then generates a delta report showing the statistics and information relevant for the time period requested. AWR is a more advanced version of the old Statspack reports that has been automated and made integral to Oracle's automated tuning processes.

The AWR reports can be run manually from SQL scripts contained in the

\$ORACLE_HOME/rdbms/admin directory on Linux, UNIX, HP-UX, and AIX systems. The basic reports are generated by the following SQL scripts:

- awrrpt.sql: Standard default, single-instance, two-dataset delta report
- awrrpti.sql: Standard specific-instance, two-dataset delta report
- awrsqrpt.sql: Report for a single specified SQL statement in default instance
- awrsqrpti.sql: Report for a single specified-instance single SQL statement, for a range of snapshot IDs
- awrddrpt.sql: Report on default instance comparing two intervals
- awrddrpi.sql: Report for a specific instance for a specified set of two intervals
- awrgrpt.sql: Global report for all instances in the default RAC database
- awrgdrpt.sql: Global report for all instances in a specific RAC database

The global reports collect the basic statistics for an Oracle Real Application Cluster (RAC) database set of instances.

Note

The global reports do not contain as rich a set of data as is available in single-instance reports. A single-instance report can be run in a RAC environment, and running single reports for each instance in the RAC database can provide better insights into some problems than running the global reports.

AWR reports are run internally each hour, and the findings are reported to the OEM interface. The user can use the Oracle Enterprise Manager (OEM) or manual processes to create a full AWR report either in text or in HTML format. These text or HTML reports are what we will be examining in this chapter and the next.

Knowing What to Look For

Before examining AWR reports, DBAs should be familiar with the basic wait events that Oracle may encounter and the typical latches that may cause performance issues. They also should be aware of what a typical performance profile looks like for their system. Usually, by examining several AWR reports from periods of normal or good performance, DBAs can acquaint themselves with the basic performance profile of their database.

Generally, DBAs should examine AWR reports from each of their production databases several times a week. These examinations should be quick and isolated to specific indicators that are helpful in the specific environment. For example, the majority physical I/O statistic for a non-Exadata online transaction processing system should be *db file sequential read*. By knowing the usual duration of a specific timeframe and what the average latency for this type of wait usually is, a DBA can determine if performance is becoming abnormal should this statistic start showing anomalous

behavior.

On Exadata, the cell single-block physical read and cell multiblock physical read indicate that the Exadata system is not doing scans but instead is doing physical single-block and multiblock reads, which aren't as efficient.

Things to notice in the baseline reports are normal levels of specific wait events and latches and what is the normal I/O profile (normal I/O rates and timings for the database files). Other items to look at are the number and types of sorts and the memory layout and program global area (PGA) activity levels.

Note

To learn which waits, latches, and statistics are significant, it is suggested that DBAs become familiar with the *Oracle Tuning Guide and Concepts Manual* (https://docs.oracle.com/cd/E11882_01/server.112/e41573.pdf). Tuning books by outside authors can also provide more detailed insight into Oracle tuning techniques and concepts.

The use of the Oracle Grid Control graphics for performance can also help DBAs spot anomalous behavior and allow them to drill down into AWR, Active Session History (ASH), and Automatic Database Diagnostic Monitor (ADDM) reports to analyze the cause.

Header Section

Unless you are looking for a specific problem, such as a known SQL issue, it is usually best to start with the topmost data in the AWR report and drill down into the later sections, as indicated by the wait events and latch data. The first section of an AWR report shows general data about the instance. [Listing 9.1](#) shows an example header section.

Listing 9.1 AWR Report Header

[Click here to view code image](#)

WORKLOAD REPOSITORY report for

DB Name	DB Id	Instance	Inst Num	Startup
Time	Release	RAC		

AULTDB	4030696936	aultdb1	1	04-Aug-08 10:16
11.1.0.6.0	YES			

Host Name	Platform	CPUs	Cores	Sockets
Memory (GB)				

```

-----
aultlinux3      Linux IA (32-
bit)          2     1      1      2.97

          Snap Id      Snap Time      Sessions Curs/Sess
-----
Begin Snap:    91 04-Aug-08 12:00:15      41      1.2
End Snap:     92 04-Aug-08 13:00:28      47      1.1
Elapsed:        60.22 (mins)
DB Time:       139.52 (mins)

Cache Sizes
~~~~~ Begin      End
~~~~~ -----
          Buffer Cache: 1,312M      1,312M Std Block
Size:      8K
          Shared Pool Size: 224M      224M      Log
Buffer:    10,604K
-----
```

The report header gives us information about the instance upon which this report was run. The AWR header was expanded to include data about the platform, CPUs, cores, and sockets as well as memory available in the server. The instance number, whether this is a RAC database, and the release level of the database are shown here. The header also includes startup time as well as the times for the two snapshots used to generate the report. The delta-t between the snapshots is also calculated and displayed. Following the instance and snapshot data, the cache sizes section gives basic data about the buffer pool, shared pool, standard block, and log buffer sizes.

Tip

All of this information in the first part of the header of the AWR report forms the foundation that we draw from to gauge whether a statistic is reasonable. For example, if we see that the system is a RAC-based system, then we know to expect RAC-related statistics to be included in the report. If we see a large number of CPUs, then we might expect to see parallel query-related numbers. The knowledge of whether a system is Linux, UNIX, AIX, Windows, or some other supported platform is also valuable in tracking down platform-specific issues.

You should pay attention to the duration of a snapshot window the report was run against. If the window is too long, then too much averaging of values could distort the true problem; if the period is too short, important events may have been missed. All in all, you need to understand why a report was generated: Was it for a specific SQL statement? If so, it might be a short duration report. Was it for a non-specific problem you are trying to isolate? Then it could be from 15 minutes to a couple of hours in duration.

Also pay attention to the number of sessions at the beginning and end of the report.

This can tell you if the load was constant, increasing, or decreasing during the report period.

The second section of the report header contains load information and is shown in [Listing 9.2](#).

Listing 9.2 Load-Related Statistics in Header

[Click here to view code image](#)

Load Profile	Per Second	Per Transaction	Per
Exec	Per Call		
~~~~~	-----	-----	-----
-----			
DB			
Time(s) :	2.3	7.1	0.63
DB			
CPU(s) :	0.3	0.9	0.07
Redo size:	800.5	2,461.8	
<b>Logical reads:</b>	<b>6,307.6</b>	<b>19,396.7</b>	
<b>Block changes:</b>	<b>3.6</b>	<b>10.9</b>	
<b>Physical reads:</b>	<b>2,704.9</b>	<b>8,317.8</b>	
Physical writes:	86.9	267.3	
<b>User calls:</b>	<b>2.2</b>	<b>6.8</b>	
<b>Parses:</b>	<b>2.0</b>	<b>6.1</b>	
Hard parses:	0.0	0.1	
<b>W/A MB processed:</b>	<b>932,965.4</b>	<b>2,868,990.9</b>	
Logons:	0.1	0.2	
Executes:	3.7	11.3	
Rollbacks:	0.1	0.3	
Transactions:	0.3		

### Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	100.00	Redo NoWait %:	99.97
Buffer Hit %:	96.09	In-memory Sort %:	100.00
Library Hit %:	98.17	Soft Parse %:	97.88
Execute to Parse %:	45.80	Latch Hit %:	99.95
Parse CPU to Parse Elapsd %:	0.00	% Non-Parse CPU:	99.77

Shared Pool Statistics	Begin	End
-----	-----	-----
Memory Usage %:	81.53	85.39
% SQL with executions>1:	79.29	79.48
% Memory for SQL w/exec>1:	76.73	78.19

## Load Profile

The critical things to watch for in the load section depend on the type of application

issue you are trying to resolve. For example, in the header section in [Listing 9.2](#), we see a large number of physical reads and logical reads with few block changes. This profile is typical for a reporting environment such as a data warehouse or decision support system (DSS). Seeing this large number of logical and physical reads should key us to look at I/O-related issues for any database performance problems. An additional sign that this is probably a warehouse or DSS environment is the large amount of work area (W/A) processing occurring, which indicates a high number of sort operations are taking place. A further indicator is that the user calls and parses are low, indicating that the transactions contain few statements and are long running.

In a predominantly online transaction processing (OLTP) system, we would expect to see more logical reads; few physical reads; many user calls, parses, and executes; as well as rollbacks and transactions. Generally, report environments have fewer and longer transactions that utilize the work area, whereas OLTP environments tend to have numerous small transactions with many commits and rollbacks.

## Instance Efficiencies

The next section of the header shows us the instance efficiency percentages; generally, you want these as close as possible to 100 percent. As you can see, all of our efficiencies are near to 100 percent with the exception of execute to parse and parse CPU to parse elapsed. Since we are dealing with a reporting system, it will probably generate a great number of ad-hoc reports. By their nature, ad-hoc reports are not reusable, so we will have low values for parse-related efficiencies in this type of system. In an OLTP system, where transactions are usually repeated over and over again, we would expect the parse-related efficiencies to be high as long as cursor sharing and bind variables were being properly utilized.

If we saw that the buffer nowait and buffer hit percentages were low (less than 95), we would investigate whether the data block buffers were being properly used and whether we might need to increase data block buffer sizes.

If the library hit percentage was low, we would consider increasing the shared pool allocation or at least look into why its percentage was low (as it could be improper bind variable usage).

The redo nowait percentage tells us how efficiently our redo buffers are being utilized, and if the percentage is low, we need to look at tuning the redo log buffers and redo logs. If processes are waiting on redo, then either the buffers are too small or something is blocking the redo logs from being reused in a proper manner. For example, in an archive log situation, if there are insufficient logs, then the system may have to wait on the archive log process while it copies a log to the archive location, decreasing the nowait percentage.

The memory sort percentage tells us if our `PGA_AGGREGATE_TARGET` or—if manual settings are used—`SORT_AREA_SIZE`, `HASH_AREA_SIZE`, and bitmap settings need to be examined. Numbers less than 100 for the sort percentage indicate that sorts are going to disk. Sorts going to disk are slow and can cause significant

performance issues.

---

### Note

Even 1 to 2 percent of sorts or temporary segments going to storage rather than memory can entail a huge amount of I/O, and you need to find and mitigate sorts and temporary segments to storage as much as possible.

---

The soft parse percentage tells us how often the SQL statement we submit is being found in the cursor caches. This statistic is directly related to proper bind variable usage and how much ad-hoc SQL generation is taking place in our system. Hard parses cause recursive SQL statements and are quite costly in processing time. In a system where SQL is being reused efficiently, this statistic should be near one hundred percent.

Latch hit percentage tells us how often we are waiting on latches. If we are frequently spinning on latches, this value decreases. If this percentage is low, look for CPU-bound processes and issues with latching.

The non-parse CPU percentage tells us how much of the time the CPU is spending on processing of our request versus how much time it is spending doing things like recursive SQL. If this percentage is low, look at the parse-related percentages, as they too will be low. When this percentage is low, it indicates the system is spending too much time processing SQL statements and not enough time doing real work.

## Shared Pool Memory

The next section of the header shows us the shared pool memory statistics. One of the main purposes of the shared pool is to provide a preparsed pool of SQL statements that can quickly be reused. This header section shows the amount of memory that is being utilized for reusable statements. Generally, if the percentages are in the 70 to 80 percent range or higher, good reuse is occurring in the database. If the percentages are less than 70 percent, the application should be reviewed for proper SQL statement reuse techniques, such as Procedural Language/Structured Query Language (PL/SQL) encapsulation and bind variable utilization.

## Wait Events

The next few sections of the header really help point the DBA or tuning user to the source of problems causing the performance issues in the database. This next section is shown in [Listing 9.3](#).

### **Listing 9.3** Wait, CPU, and Memory Statistics

[Click here to view code image](#)

---

Top 5 Timed Foreground Events  
~~~~~

| Event | Waits | Time (s) | Avg Wait (ms) | % DB Time |
|--|---------|------------|---------------|-------------|
| Wait Class | | | | |
| <hr/> | | | | |
| db file sequential read | 465,020 | 3,969 | 9 | 47.4 |
| User I/O | | | | |
| DB CPU | | 995 | | 11.9 |
| db file parallel read | 2,251 | 322 | 143 | 3.8 |
| User I/O | | | | |
| db file scattered read | 15,268 | 153 | 10 | 1.8 |
| User I/O | | | | |
| gc current block 2-way | 108,739 | 116 | 1 | 1.4 |
| Cluster | | | | |
| Host CPU (CPUs: 2 Cores: 1 Sockets: 1) | | | | |
| <hr/> | | | | |
| ~~~~~ Load Average | | | | |
| ~~~~~ Begin | End | %User | %System | %WIO |
| <hr/> | | | | |
| --- | 0.37 | 3.05 | 10.6 | 6.7 |
| | | | | 45.3 |
| | | | | 8: |
| Instance CPU | | | | |
| <hr/> | | | | |
| % of total CPU for Instance: | 14.8 | | | |
| % of busy CPU for Instance: | 85.0 | | | |
| %DB time waiting for CPU - Resource Mgr: | 0.0 | | | |
| <hr/> | | | | |
| Memory Statistics | | | | |
| <hr/> | | | | |
| Begin | End | | | |
| Host Mem (MB) : | 3,041.4 | | | |
| SGA use (MB) : | 1,584.0 | | | |
| PGA use (MB) : | 169.0 | | | |
| % Host Mem used for SGA+PGA: | 57.64 | | | |
| | 57.64 | | | |

Of the statistics in this next section of the header, the top five wait statistics are probably the most important. The wait interface in Oracle stores counters for waits and timings for several hundred wait events that instrument the internals of the database. By examining the wait statistics, you can quickly find the pinch points for performance in your database. In our example in [Listing 9.3](#), we can see the following statistics are the dominant waits:

[Click here to view code image](#)

| Event | Waits | Time (s) | Avg Wait (ms) | % DB Time |
|-------------------|-------|----------|---------------|-----------|
| Wait Class | | | | |
| <hr/> | | | | |
| <hr/> | | | | |

| | | | | |
|-------------------------|---------|-------|-----|------|
| db file sequential read | 465,020 | 3,969 | 9 | 47.4 |
| User I/O | | | | |
| db file parallel read | 2,251 | 322 | 143 | 3.8 |
| User I/O | | | | |
| db file scattered read | 15,268 | 153 | 10 | 1.8 |
| User I/O | | | | |

Notice that all of the events are dealing with the I/O subsystem. In this system, the I/O subsystem consists of dual 1 Gb host bus adapters (HBAs) on each node using multipath talking through a Brocade 2250 switch to two JBOD (just a bunch of disks) arrays, one holding 18 to 72 Gb 10-K disks and one holding 8 to 72 Gb 10-K disks. The disks are using the “stripe and mirror everything” (SAME) methodology as implemented through Oracle ASM: two of the disks are being used for voting and cluster configuration files for the cluster ready services, leaving 24 disks to service the database in two-disk groups, one with 16 disks using normal redundancy and the other using 8 disks with external redundancy. Obviously, with 54 percent of the wait activity (and probably more) being I/O related, our I/O subsystem on this RAC setup is being stressed.

Note

The CPU is showing 11.9 percent usage during this period. In a properly tuned system, the CPU should show the dominant time percentage.

Other wait events that could dominate a RAC environment could be redo log related, interconnect related, or undo tablespace related. Note that the fifth-largest wait was `gc current block 2-way`. This indicates that the same block was being shared back and forth across the interconnect. Of course, since this is a parallel query environment with the parallel query being not only cross table and cross index but also cross node, some amount of interconnect-related waits are expected. However, if `gc`-related waits dominated the top five wait events section, this would indicate there was definite stress on the interconnect and it was a significant source of wait issues.

In this case, the predominant wait is `db file sequential read`, which indicates that single-block reads (i.e., index reads) are causing issues. Normally, this would be resolved by adding more DB block cache memory (server memory); however, our system is memory constrained, so if we can't remove the waits, we must look to reducing the wait time per incident. By increasing the number of disks in the array and increasing the spread of the files causing the reads, we could possibly get this wait to as small as 5 ms (maybe lower if we move to a more expensive cached storage area network [SAN] setup), but this would be the limit in a disk-based system. The only way to further reduce the value would be to increase the amount of server memory through a server upgrade or decrease the read latency by moving to something like all flash arrays. The other read-based waits would also benefit from either more memory or faster I/O subsystems.

The other major wait that is usually seen when an I/O subsystem is stressed is `db file scattered read`, which indicates full table or index scans are occurring.

Full table scans can usually be corrected by proper indexing. However, in DSS or warehouse situations, this may not always be possible. In the case of DSS or data warehouses, use of partitioning can reduce the amount of data scanned. However, each disk read is going to require at least 5 ms, and the only way to beat that is through large-scale caching or the use of all flash arrays to reduce latency.

Tip

In DSS systems, the schema should be in what is known as a *star configuration* in which a central fact table is surrounded by many dimension tables. The fact table acts as an intersection table for the dimensions. By putting bitmap indexes on the foreign keys and setting the initialization parameter

`STAR_TRANSFORMATION_ENABLED` to TRUE, Oracle can use highly efficient star transformation queries to process the large queries related to DSS systems in a fraction of the time and I/O required by full table scans.

When db file sequential read or db file scattered read are the significant wait sources, the DBA needs to look in the SQL sections of the report to review the top SQL statement that is generating excessive logical or physical reads. Usually, if a SQL statement shows up in two or more of the SQL subsections, it is a top candidate for tuning actions.

A key indicator of log file stress (redo logs) would be the log file sync, log file parallel write, log file sequential write, log file single write, or log file switch completion wait events being dominant in the top five wait event listing. However, you must make sure that the wait is truly from I/O-related issues, and not issues such as archive logging, before taking proper action. Usually, log file stress occurs when the log files are being placed on the same physical disks as the data and index files and can usually be relieved by moving the logs to their own disk array section or utilizing a single logical unit number (LUN) for redo logs separate from other files when dealing with a flash memory-based storage system. However, if high wait times for log-related events occur, then moving the logs to a lower-latency I/O subsystem is indicated.

Load Average

The next section of the report shows the breakdown of the CPU timing and run queue status (load average) for the time interval. In this case, the load average corresponds to the ratio of DB time to the amount of time expected to be provided by a single CPU. Since DB time is made up of DB CPU, I/O wait, and non-idle event time, it can reflect excessive I/O wait and not high CPU usage when load is high.

Note

If the period of the report covers more than one AWR collection interval, the Load section will show an entry for each collection.

[Click here to view code image](#)

```

Host CPU (CPUs:      2 Cores:     1 Sockets:     1)
~~~~~ Load Average
      Begin      End      %User      %System      %WIO      %I/O
----- ----- ----- ----- -----
---          0.37      3.05     10.6       6.7      45.3      82

```

The run queue tells you how many processes are waiting to execute. If the run queue exceeds the number of available CPUs, and CPU is not idle, then increasing the number of CPUs or upgrading the speed of CPUs is indicated, assuming all other tuning actions, such as reducing recursion, have been accomplished. As you can see from the report, our CPU was 83 percent idle during the period, while I/O waits was 45 percent; thus, CPU stress was not causing the run queue of 3: it was most likely I/O-related wait activity.

The other statistics in the section show the amount of time utilized in user and system modes of the CPU as well as the percentage of time the CPU was idle and the average I/O wait. If the I/O wait percentage is high, then increasing the number of disks (after proper tuning has occurred) may help. If you have already tuned SQL and reduced I/O to its lowest possible value through tuning and I/O wait is still a large percentage of the total waits, then your only choice is moving to a lower-latency I/O subsystem, such as an all flash array.

Instance CPU

Following the CPU timing section is the Instance CPU section, which shows how efficiently this instance was using the CPU it was given:

[Click here to view code image](#)

```
Instance CPU
~~~~~  
          % of total CPU for Instance: 14.8  
          % of busy CPU for Instance: 85.0  
%DB time waiting for CPU - Resource Mgr: 0.0
```

This instance utilized the total CPU time available for only 14.8 percent of the time; of that 14.8 percent CPU, it utilized the time 85 percent for processing. Since no resource groups are in use in the database, zero percent of the CPU was used for resource management using the resource manager. This again points out that the system was I/O bound, leaving the system basically idle while it waited on disks to serve data.

Memory Statistics

The last section of the header deals with memory usage:

[Click here to view code image](#)

Memory Statistics

| | | |
|------------------------------|---------|---------|
| Host Mem (MB) : | 3,041.4 | 3,041.4 |
| SGA use (MB) : | 1,584.0 | 1,584.0 |
| PGA use (MB) : | 169.0 | 301.7 |
| % Host Mem used for SGA+PGA: | 57.64 | 57.64 |

According to Oracle you should use only about 60 percent of the memory in your system for Oracle. However, as memory sizes increase, this old saw is showing its age. In memory-constrained 32-bit systems such as the one this report came from, 60 percent is probably a good point to shoot for with Oracle memory usage. As you can see, this instance is using 57.64 percent, so it is pretty close to the 60 percent limit. The rest of the memory is reserved for process and operating system requirements. We can see that our system global area (SGA) remained fairly stable at 1,584 MB, while our PGA usage grew from 169 to 302 MB. This again points to the system being a report DSS or data warehouse system that is utilizing a lot of sort area.

RAC-Specific Pages

After the header and system profiles area is a RAC-specific section that will not be present if RAC is not being used. The first part of the RAC-specific statistics deal with profiling the global cache load. This section of the report is shown in [Listing 9.4](#).

Listing 9.4 RAC Load Profiles

[Click here to view code image](#)

```
RAC Statistics  DB/Inst: AULTDB/aultdb1  Snaps: 91-92
```

| | Begin | End |
|----------------------|-------|-------|
| ----- | ----- | ----- |
| Number of Instances: | 2 | 2 |

Global Cache Load Profile

| ~~~~~ | Per Second | Per |
|-------------|------------|-------|
| Transaction | ----- | ----- |

| | | |
|--------------------------------|--------|--------|
| ----- | ----- | ----- |
| Global Cache blocks | | |
| received: | 26.51 | 81.54 |
| Global Cache blocks | | |
| served: | 26.02 | 80.01 |
| GCS/GES messages | | |
| received: | 156.31 | 480.68 |
| GCS/GES messages | | |
| sent: | 157.74 | 485.06 |
| DBWR Fusion | | |
| writes: | 0.01 | 0.04 |
| Estd Interconnect traffic (KB) | | |
| | 481.59 | |

```
Global Cache Efficiency Percentages (Target local+remote 100%)
~~~~~
Buffer access - local cache %: 95.44
Buffer access - remote cache %: 0.65
Buffer access - disk %: 3.91
```

RAC Statistics (CPU)

The first part of the listing shows how many instances we started with in the RAC environment and how many we ended up with. This information is important because cross-instance parallel operations would be directly affected with loss of or addition of any instances to the RAC environment.

Global Cache Load Statistics

The next section of the RAC report is the Global Cache Load Profile. This profile shows how stressed the global cache may have been during the period monitored. In the report previously shown, we transferred only 481 KB/sec across an interconnect capable of handling 100 MB/sec, so we were hardly using the interconnect in spite of our cross-instance parallel operations. This is further shown by the low Buffer Access - Remote cache statistic of 0.65 percent, which tells us only 0.65 percent of our blocks came from the other node's RAC instance.

Things to watch out for in this section are severe instance unbalancing where the Global Cache blocks received versus the Global Cache blocks served is way out of alignment (they should be roughly equal). Another possible indication of problems is excessive amounts of database writer (DBWR) fusion writes; fusion writes should be done only for cache replacement, which should be an infrequent operation.

Tip

If fusion writes are excessive, it could indicate inadequately sized DB block cache areas, excessive checkpointing, excessive commits, or a combination of all three.

Global Cache and Enqueue Services

The next RAC-specific section deals with the actual timing statistics associated with the global cache. You need to pay close attention to the various block service timings. If the time it takes to serve a block across the interconnect exceeds the time it would take to read it from disk, then the interconnect is becoming a bottleneck instead of a benefit. The section is shown in [Listing 9.5](#).

Listing 9.5 Global Cache and Enqueue Workload Section

[Click here to view code image](#)

Global Cache and Enqueue Services - Workload Characteristics

Avg global enqueue get time (ms) : 0.2

Avg global cache cr block receive time (ms) : 1.8

Avg global cache current block receive time (ms) : 1.8

Avg global cache cr block build time (ms) : 0.0

Avg global cache cr block send time (ms) : 0.1

Global cache log flushes for cr blocks served %: 0.8

Avg global cache cr block flush time (ms) : 17.5

Avg global cache current block pin time (ms) : 0.0

Avg global cache current block send time (ms) : 0.1

Global cache log flushes for current blocks served %: 0.0

Avg global cache current block flush time (ms) : 20.0

Global Cache and Enqueue Services - Messaging Statistics

Avg message sent queue time (ms) : 1.1

Avg message sent queue time on ksxp (ms) : 1.3

Avg message received queue time (ms) : 0.1

Avg GCS message process time (ms) : 0.0

Avg GES message process time (ms) : 0.0

% of direct sent messages: 35.13

% of indirect sent messages: 64.34

% of flow controlled messages: 0.54

The most important statistics in this section are the following:

[Click here to view code image](#)

Avg global cache cr block receive time (ms) : 1.8

Avg global cache current block receive time (ms) : 1.8

These should be compared to an AWR report run on the other instance:

[Click here to view code image](#)

Avg global cache cr block receive time (ms) : 2.1

Avg global cache current block receive time (ms) : 1.7

If the numbers on both or all RCA instances aren't similar, then this could indicate a problem with the interconnect either at the OS buffer level or the NIC or interface cards themselves.

Notice the high flush values in [Listing 9.5](#). These are not the correct values and

probably point to an issue with the way AWR is collecting the data, since they are a direct input to the product that results in the receive times, which are not more than 3 ms.

The global enqueue timing numbers should be less than 2 to 3 ms in most cases. If they get anywhere near the 20 ms, as stated in the Oracle documents, you have a serious issue with the Global Enqueue Service (GES) part of the global dictionary.

Cluster Interconnects

The last part of this RAC-specific section deals with the interconnect:

[Click here to view code image](#)

```
Cluster Interconnect
~~~~~ Begin E1
-----
Interface      IP Address      Pub
Source          IP            Pub Src
-----           ----- C -----
-----
eth0           11.1.1.1        N   Oracle Cluster Repository
```

You should verify that the interconnect is the proper private interconnect and that it is not a public Ethernet. If you see excessive Global Cache Service (GCS) values in previous sections, be sure that the proper interconnect is being used.

Time Model Statistics

Another nice addition to the statistics in AWR over Statspack are the time breakdown statistics that show the components of CPU, OS, and other time fields. The first shown is the CPU statistics breakdown, shown in [Listing 9.6](#).

Listing 9.6 CPU Time Breakdown

[Click here to view code image](#)

```
Time Model Statistics          DB/Inst: AULTDB/aultdb1  Snaps:
91-92
-> Total time in database user-calls (DB Time): 8371.3s
-> Statistics including the word "background" measure background
process
    time, and so do not contribute to the DB time statistic
-> Ordered by % or DB time desc, Statistic name

Statistic Name                Time (s) % of
DB Time
-----
```

| | | |
|---------------------------------------|---------|---------|
| sql execute elapsed | | |
| time | 8,145.5 | 97.3 |
| DB | | |
| CPU | | 995.1 |
| parse time | | |
| elapsed | 7.4 | .1 |
| hard parse elapsed | | |
| time | 5.2 | .1 |
| PL/SQL execution elapsed | | |
| time | 4.8 | .1 |
| Java execution elapsed | | |
| time | 0.7 | .0 |
| hard parse (sharing criteria) elapsed | | |
| time | 0.2 | .0 |
| sequence load elapsed | | |
| time | 0.1 | .0 |
| repeated bind elapsed | | |
| time | 0.1 | .0 |
| PL/SQL compilation elapsed | | |
| time | 0.0 | .0 |
| failed parse elapsed | | |
| time | 0.0 | .0 |
| hard parse (bind mismatch) elapsed | | |
| time | 0.0 | .0 |
| DB time | | 8,371.3 |
| background elapsed time | | 214.7 |
| background cpu time | | 75.8 |

Much of how Oracle determines CPU seconds is a black box; for example, how Oracle can tell us in one section that the CPU was utilized only 14.8 percent for this instance and yet shows `sql execute elapsed` time as 8,145.5 seconds when with 2 CPUs there are only 7,200 seconds of CPU time in an hour is a real mystery. It may be including all calls that completed during the interval, which of course may then include calls whose majority of time was actually outside of the measurement interval; however, that is a topic for another discussion.

In the report excerpt shown in [Listing 9.6](#), we see that a majority of the CPU time that was allotted to us for this measurement interval (97.3%) was spent in `SQL execute elapsed` time, and this really is precisely where we want the CPU to be spending its time. If we saw that `parse time elapsed` or `hard parse elapsed` time was consuming a large portion of time, that would indicate that we had an ad-hoc environment with a majority of unique SQL statements or that our application is not using bind variables properly. Of course, if the CPU was spending its time in any of the other areas for a majority of the reported time, that segment of processing should be investigated.

Tip

If parse time elapsed and hard parse elapsed time are close in value, then excessive hard parses are happening. Hard parses are a drain on CPU and I/O resources, and you should review the SQL versioning and look at shared pool reloads and invalidations for possible clues.

Operating System Statistics

The next section of the AWR report shows OS-related settings and statistics. [Listing 9.7](#) shows an example report section for OS statistics.

Listing 9.7 OS Statistics

[Click here to view code image](#)

| Statistic | Value | End Value |
|--------------------------|---------------|-----------|
| BUSY_TIME | 126,029 | |
| IDLE_TIME | 597,505 | |
| IOWAIT_TIME | 327,861 | |
| NICE_TIME | 766 | |
| SYS_TIME | 48,452 | |
| USER_TIME | 76,784 | |
| LOAD | 0 | 3 |
| PHYSICAL_MEMORY_BYTES | 3,189,190,656 | |
| NUM_CPUS | 2 | |
| NUM_CPU_CORES | 1 | |
| NUM_CPU_SOCKETS | 1 | |
| GLOBAL_RECEIVE_SIZE_MAX | 4,194,304 | |
| GLOBAL_SEND_SIZE_MAX | 262,144 | |
| TCP_RECEIVE_SIZE_DEFAULT | 87,380 | |
| TCP_RECEIVE_SIZE_MAX | 1,048,576 | |
| TCP_RECEIVE_SIZE_MIN | 4,096 | |
| TCP_SEND_SIZE_DEFAULT | 65,536 | |
| TCP_SEND_SIZE_MAX | 1,048,576 | |
| TCP_SEND_SIZE_MIN | 4,096 | |
| ----- | | |

Operating System Statistics - Detail DB/Inst:

AULTDB/aultdb1 Snaps: 91-92

| Snap
Time | Load | %busy | %user | %sys | %idle | %iowait |
|--------------------|------|-------|-------|------|-------|---------|
| <hr/> | | | | | | |
| 04-Aug
12:00:15 | 0.4 | N/A | N/A | N/A | N/A | N/A |
| 04-Aug
13:00:28 | 3.0 | 17.4 | 10.6 | 6.7 | 45.3 | 82.6 |
| <hr/> | | | | | | |

The OS section of the report gives us the time breakdown in CPU ticks to support the percentages claimed in the other sections of the report. The correlation of reported ticks to actual ticks is still a bit foggy. However, examination of this section still shows that the system being examined is not CPU bound but is suffering from I/O contention, since both the idle time and I/O wait time statistics are larger than the busy time value. This part of the report also shows the TCP/UDP buffer settings, which are useful when determining issues in RAC.

Tip

If you don't have access to Oracle Grid Control, you can get a multihour or multiday AWR report and use the Operating System Statistics - Detail section to pinpoint where the load and I/O stress are the highest for further analysis with more targeted reports.

One problem that has been noted is that the I/O wait time reported in this section may not be accurate and should not be used for analysis.

Foreground Wait Events

The next section of the AWR report shows the wait events that occur in foreground processes. Foreground processes are the user- or application-level processes. [Listing 9.8](#) shows an excerpt from the report we are analyzing.

[Listing 9.8](#) Foreground Wait Events

[Click here to view code image](#)

Foreground Wait Class DB/Inst: AULTDB/aultdb1 Snaps:
91-92
-> s - second, ms - millisecond - 1000th of a second
-> ordered by wait time desc, waits desc
-> %Timeouts: value of 0 indicates value was < .5%. Value of null
is truly 0

-> Captured Time accounts for 68.9% of Total DB time 8,371.33 (s)
-> Total FG Wait Time: 4,770.85 (s) DB CPU time: 995.13 (s)

| Wait Class
DB Time | Waits | % Time -outs | Total Wait Time (s) | Avg Wait (ms) | % |
|-----------------------|-----------|--------------|---------------------|---------------|-----|
| <hr/> | | | | | |
| User | | | | | |
| I/O | 518,267 | 0 | 4,449 | 9 | 5: |
| DB | | | | | |
| CPU | | | 995 | | |
| Cluster | 188,753 | 9 | 173 | 1 | |
| Other | 3,806,446 | 100 | 146 | 0 | |
| Concurrency | 1,854 | 2 | 2 | 1 | |
| Commit | 15 | 0 | 1 | 39 | |
| Application | 740 | 0 | 0 | 0 | |
| System | | | | | |
| I/O | 40 | 0 | 0 | 3 | 0.1 |
| Network | 6,970 | 0 | 0 | 0 | 0 |
| Configuration | 0 | | 0 | | |
| <hr/> | | | | | |

Foreground Wait Events DB/Inst: AULTDB/aultdb1 Snaps: 91-92
-> s - second, ms - millisecond - 1000th of a second
-> Only events with Total Wait Time (s) >= .001 are shown
-> ordered by wait time desc, waits desc (idle events last)
-> %Timeouts: value of 0 indicates value was < .5%. Value of null is truly 0

| Wait Event | Waits (ms) | % DB /txn | Time | Waits | % Time -outs | Total Wait Time (s) | Avg |
|-------------------------|------------|-----------|-------|-------|--------------|---------------------|-----|
| <hr/> | | | | | | | |
| <hr/> | | | | | | | |
| db file sequential read | 465,020 | 0 | 3,969 | 9 | 395.8 | 47.4 | |
| db file parallel read | 2,251 | 0 | 322 | 143 | 1.9 | 3.8 | |
| db file scattered read | 15,268 | 0 | 153 | 10 | 13.0 | 1.8 | |
| gc current block 2-way | 108,739 | 11 | 116 | 1 | 92.5 | 1.4 | |

| | | | | | | |
|----------------------|---------|-----------|-----|-----|-------|----------|
| PX Deq: reap credit | | 3,247,703 | 100 | | 107 | 0 |
| 2,764.0 1.3 | | | | | | |
| gc cr grant 2- | | | | | | |
| way | 57,265 | 7 | 28 | 0 | 48.7 | .3 |
| gc cr multi block | | | | | | |
| request | 22,451 | 6 | 23 | 1 | 19.1 | .3 |
| enq: BF - allocation | | | | | | |
| conte | 14 93 | 14 | 983 | 0.0 | .2 | |
| PX qref | | | | | | |
| latch | 555,843 | 100 | | 9 | 0 | 473.1 .1 |
| IPC send completion | | | | | | |
| sync | 1,070 | 52 | 8 | 8 | 0.9 | .1 |
| gc | | | | | | |
| remaster | | 22 | 0 | | 5 221 | 0.0 |

The wait events that accounted for less than 0.1 percent of DB time have been omitted for brevity (the actual listing is two pages long). The thing to note about this section of the report is that we have already looked at the main top five events, which are where we should focus our tuning efforts. However, if you are attempting to tune some specific operation or database section, the other waits in this section may apply to that effort.

If you see a large number of waits in a RAC environment for the `read by other session` event, this usually indicates your block size or latency is too large, which is resulting in contention. If you see that `read by other session` is one of the predominant waits events, look to the `Segments by x sections` of the AWR reports for guidance on which tables and indexes are being heavily utilized. Consider moving these segments to a tablespace with a smaller-than-default block size, such as 4 KB or even 2 KB, or to lower latency storage, such as IBM FlashSystem, to reduce this contention.

Background Wait Events

Background wait events, as their name implies, are waits generated by the numerous background processes in the Oracle process stack. DBWR, log writer (LGWR), system monitor (SMON), process monitor (PMON), and so on, all contribute to the background wait events. The report excerpt, limited to the events with at least 0.1 percent of database time, is shown in [Listing 9.9](#).

Listing 9.9 Background Wait Events Section

[Click here to view code image](#)

| | |
|--|----------|
| Background Wait Events | DB/Inst: |
| AULTDB/aultdb1 Snaps: 91-92 | |
| -> ordered by wait time desc, waits desc (idle events last) | |
| -> Only events with Total Wait Time (s) >= .001 are shown | |
| -> %Timeouts: value of 0 indicates value was < .5%. Value of null is truly 0 | |

| Event | Wait
(s) | Wait
(ms) | Waits
/txn | % Time
Time | Avg | |
|-------------------------|-------------|--------------|---------------|----------------|------|-------|
| | | | | | % | Total |
| | | | | | bg | |
| <hr/> | | | | | | |
| control file sequential | | | | | | |
| re | 8,336 | 0 | 72 | 9 | 7.1 | 33.5 |
| control file parallel | | | | | | |
| writ | 1,287 | 0 | 31 | 24 | 1.1 | 14.5 |
| db file parallel | | | | | | |
| write | 792 | 0 | 11 | 14 | 0.7 | 5.3 |
| log file parallel | | | | | | |
| write | 701 | 0 | 11 | 15 | 0.6 | 4.9 |
| events in waitclass | | | | | | |
| Other | 44,191 | 98 | 5 | 0 | 37.6 | 2.5 |
| library cache | | | | | | |
| pin | 449 | 0 | 2 | 4 | 0.4 | .8 |
| db file sequential | | | | | | |
| read | 221 | 0 | 2 | 7 | 0.2 | .8 |
| gc cr multi block | | | | | | |
| request | 1,915 | 0 | 2 | 1 | 1.6 | .7 |
| os thread | | | | | | |
| startup | 19 | 0 | 1 | 56 | 0.0 | .5 |
| gc cr block 2- | | | | | | |
| way | 246 | 0 | 0 | 1 | 0.2 | .2 |
| db file scattered | | | | | | |
| read | 18 | 0 | 0 | 12 | 0.0 | .1 |
| db file parallel | | | | | | |
| read | 3 | 0 | 0 | 59 | 0.0 | .1 |
| gc current grant 2- | | | | | | |
| way | 98 | 0 | 0 | 1 | 0.1 | .1 |

As we can see, the events that dominate the background waits are also I/O related. If the events that are at the top in the foreground are similar (such as both being control file-related), then that is the I/O area in which we should concentrate our tuning efforts. As we can see from the report excerpt, while the I/O related waits are similar, the predominant ones in each section are different, indicating a general I/O issue rather than an issue with a specific set of files.

Wait Event Histograms

In the next section of the AWR report, Oracle provides a time-based histogram report for the wait events. If the histogram report were ordered by the predominant wait events by time, it would be more useful; instead, it is ordered by event name, making us have to search for the important events. For brevity, the unimportant events have been removed from the example in [Listing 9.10](#).

Listing 9.10 Event Time Histograms

[Click here to view code image](#)

```
Wait Event Histogram                                DB/Inst:  
AULTDB/aultdb1  Snaps: 91-92  
-> Units for Total Waits column: K is 1000, M is 1000000, G is  
1000000000  
-> % of Waits: value of .0 indicates value was <.05%. Value of null  
is truly 0  
-> % of Waits: column heading of <=1s is truly <1024ms, >1s is  
truly >=1024ms  
-> Ordered by Event (idle events last)
```

| Event | <=1s | >1s | % of Waits | | | | | | |
|-----------------------------------|-------------|-------------|------------|------------|-------------|-------------|------------|------------------------|-------|
| | | | Total | Waits | <1ms | <2ms | <4ms | <8ms | <16ms |
| control file parallel writ | 1287 | | | | | | | 59.0 24.1 | |
| 16.9 | | | | | | | | | |
| control file sequential re | 9147 | | | | | | | 23.4 21.3 23.3 | |
| 22.3 6.8 2.9 .0 | | | | | | | | | |
| db file parallel read | | 2256 | | | | | | .3 1.0 7.4 32.6 | |
| 56.8 1.9 | | | | | | | | | |
| db file parallel | | | | | | | | | |
| write | 792 | .5 | .8 | 4.2 | 28.7 | 50.0 | 8.8 | 7.1 | |
| db file scattered | | | | | | | | | |
| read | 15K | | .4 | 2.7 | 31.5 | 59.2 | 5.8 | .5 | |
| db file sequential | | | | | | | | | |
| read | 465K | .0 | .6 | 2.2 | 49.5 | 45.0 | 2.3 | .4 | |
| gc cr grant 2- | | | | | | | | | |
| way | 50K | 87.2 | 11.1 | 1.3 | .3 | .2 | | .0 | |
| gc cr multi block | | | | | | | | | |
| request | 24K | 59.0 | 36.8 | 3.0 | .5 | .6 | | .0 | |
| gc current block 2- | | | | | | | | | |
| way | 84K | 6.5 | 87.7 | 5.2 | .3 | .2 | | .0 | |
| library cache | | | | | | | | | |
| lock | 488 | 82.8 | 10.9 | 4.9 | 1.0 | .2 | | .2 | |
| library cache | | | | | | | | | |
| pin | 4371 | 77.6 | 11.1 | 7.4 | 3.1 | .6 | | .0 | |
| gcs remote | | | | | | | | | |
| message | | 274K | 28.5 | 15.4 | 9.9 | 11.6 | 7.5 | 5.8 21.4 | |
| ges remote | | | | | | | | | |
| message | | 53K | 11.4 | 3.3 | 2.7 | 1.9 | 1.8 | 2.1 76.8 | |

The most important events in the excerpt are in bold. Notice that the read events are more important than the write events. Unless we are talking about direct writes, direct write temp, undo or redo log writes, or control file writes, Oracle uses the concept of delayed block cleanout, writing blocks to disk only when absolutely needed. This delayed block cleanout mechanism means that for most data-related writes, we aren't too concerned with write times unless our application does frequent commits and the data is needed nearly immediately after the commit.

Since this application is a read-dominated application, there aren't a lot of redo log-related and undo tablespace-related events. In an OLTP type environment, we would expect to see log writes and log syncs to rise to the top in an I/O-bound system as dominant events. In systems that generate a lot of transactions, we would also expect undo tablespace-related events to be more prevalent.

By looking at the histograms, we can see that our read events are taking anywhere from 4 ms to 1 s to complete; this is a typical disk-based histogram. However, in our histogram, the largest number of reads by percentage are taking greater than 8 ms to complete; this indicates disk stress is happening. We could possibly reduce this to nearer to 5 ms by increasing the number of disks in our disk array. However, you cannot expect a disk-based system to get to less than 5 ms read or write times unless you place a large amount of cache in front of the disks. Another option, which can be more cost effective and allow greater I/O operations per second (IOPS), is to use IBM FlashSystem technology, which should provide less than 0.5 ms latency.

Service-Related Statistics

Since Oracle Database 10g, Oracle is increasingly using the concept of a database “service.” A service is a group of processes that are used to provide a common function. For example, all of the parallel query slaves and processes used to provide the results for a series of SQL statements for the same user could be grouped into a service. [Listing 9.11](#) shows the service-related section for our example report.

Listing 9.11 Service-Related Statistics Section

[Click here to view code image](#)

| Service Statistics | | DB/Inst: | Physical |] |
|-----------------------------|--------------|------------|--------------|-------|
| AULTDB/aultdb1 Snaps: 91-92 | | | Reads | |
| -> ordered by DB Time | | | | |
| Service Name | DB Time (s) | DB CPU (s) | Physical | |
| (K) | Reads (K) | | Reads | |
| ----- | ----- | ----- | ----- | ----- |
| aultdb | 8,344 | 981 | 9,769 | |
| SYS\$USERS | 23 | 12 | 1 | |

```

SYS$BACKGROUND          1          0          1
aultdbXDB              0          0          0
-----
-----
Service Wait Class Stats           DB/Inst:
AULTDB/aultdb1  Snaps: 91-92
-> Wait Class info for services in the Service Statistics section.
-> Total Waits and Time Waited displayed for the following wait
   classes: User I/O, Concurrency, Administrative, Network
-> Time Waited (Wt Time) in seconds

Service Name
-----
User I/O User I/O Concurcy
Concurcy     Admin    Admin   Network  Network
Total Wts   Wt Time Total Wts   Wt Time Total
Wts        Wt Time
-----
-----
aultdb
  517710      4446      234       1       0       0      5828
SYS$USERS
    555         3      1615       1       0       0      1140
SYS$BACKGROUND
    350         3      3486       4       0       0       0
-----
-----
```

The service-related statistics allow you to see which users are consuming the most resources. By knowing which users are consuming the most resources, you can concentrate your tuning activities on the hard hitters. In the report excerpt in [Listing 9.11](#), we see the generic service aultdb, which holds all the user processes that are non-background and non-sys owned.

Since there was only one set of processes (we know this because we are good DBAs who keep tabs on what is happening in our system), we can track the usage back to a user called tpch. From looking at the second half of the report, we can see that our user experienced over 517,710 I/O-related waits for a total effective wait time of 4,446 seconds, or 8.59 ms per wait.

Since we know that the best wait time we can get with a disk-based, non-cached system is 5 ms, a wait time of 8.59 ms shows the disks are experiencing some stress. The higher this type of wait is, the more stress is being experienced by the I/O subsystem. This section of the report can show where the timing issues are occurring.

The SQL Sections

The next sections of the report slice and dice the SQL in the shared pool by several

different statistics. By using the waits and other statistics we have discussed so far, you can usually figure out which SQL area to examine. A general rule of thumb is that if a SQL statement appears in the top five statements in two or more areas, it is a prime candidate for tuning. The sections are

- Total Elapsed Time
- Total CPU Time
- Total Buffer Gets
- Total Disk Reads
- Total Executions
- Total Parse Calls
- Total Sharable Memory
- Total Version Count
- Total Cluster Wait Time

Let's look at each section and discuss the indicators that would lead you to consider investigating the SQL in each.

Total Elapsed Time

If a SQL statement appears in the Total Elapsed Time area of the report, this means its CPU time plus any other wait times moved it to the top of the pile. If for some reason it is at the top of the Total Elapsed Time but not at the top of Total CPU Time, this indicates that an issue with recursion is associated with this statement. Generally, the same SQL will be in both the Total Elapsed Time and Total CPU Time sections. If you see high recursion indicators such as suboptimal parse ratios in the Instance Activity Statistics (the section following the SQL areas), the *recursive calls* or *recursive CPU usage* statistics are high.

Total CPU Time

When a statement appears in the Total CPU Time section, this indicates it used excessive CPU cycles during its processing. Excessive CPU processing time can be caused by sorting, excessive functions, or long parse times. Indicators that you should be looking at in this section for SQL tuning candidates are high CPU percentages in the services section for the service associated with this SQL statement. To reduce total CPU time, reduce sorting by using multicolumn indexes that can act as sort eliminators and use bind variables to reduce parse times.

Tip

If the SQL is uppercase, it probably comes from a user or application; if it is lowercase, it usually comes from internal or background processes. SQL with all Table fields with single quotes around them are usually generated from user tools.

Total Buffer Gets

High total buffer gets mean a SQL statement is reading a lot of information from the DB block buffers. Generally speaking, buffer gets (or logical reads in Statspack) are desirable, except when they become excessive. Like excessive disk reads, excessive buffer gets can cause performance issues, and they are reduced in the same way. To reduce excessive total buffer gets, use partitioning, use indexes, and look at optimizing SQL to avoid excessive full table scans. Total buffer gets are typified by high logical reads, high buffer cache hit ratios (when they are driven by a poor selectivity index), and high CPU usage.

Total Disk Reads

High total disk reads mean a SQL statement is reading a lot of information from storage rather than from the DB block buffers. Generally speaking, storage reads (or physical reads in Statspack) are undesirable, especially when they become excessive. Excessive storage reads cause performance issues. To reduce excessive storage, use partitioning, use indexes, and look at optimizing SQL to avoid excessive full table scans. You can also increase the DB buffer cache if memory is not an issue. Total disk reads are typified by high physical reads, low buffer cache hit ratios, and low CPU usage with high I/O wait times. If storage reads are a part of your database (such as DSS or data warehouses where full table scans are a natural result of their structure), then moving to an all flash array would improve performance, sometimes dramatically.

Total Executions

High total executions can be an indicator that something in the SQL is incorrect in the database. Statements with high numbers of executions usually are being properly reused. However, be sure that it's okay that statements with high numbers of executions are executed multiple times. For example, a SQL statement might execute over and over again in a PL/SQL or Java, or in a C routine in a loop, when it should only be executed once. Statements with high executions and high logical and/or physical reads are candidates for review to make sure they are not being executed multiple times when a single execution would serve. If the database is seeing excessive physical and logical reads or excessive I/O wait times, then look at the SQL statements that show excessive executions and show high physical and logical reads.

Parse Calls

Whenever a statement is issued by a user or process, regardless of whether it is in the SQL pool, it undergoes a parse. The parse can be a hard parse or a soft parse. If Oracle cannot find an identical hash signature in the SQL pool, it does a hard parse with loads of recursive SQL and all the rest of the parse baggage. If it finds the SQL in the pool, then it simply does a soft parse with minimal recursion to verify user permissions on the underlying objects. Excessive parse calls usually go with excessive executions. If the statement is using what are known as *unsafe bind variables*, then the statement will be

reparsed each time. If the header parse ratios are low, look here and in the version count areas.

Shareable Memory

The shareable memory area provides information on SQL statements that are reused and the amount of memory in the shared pool that they consume. Only statements with greater than 1,048,576 bytes of shared memory usage are shown in the report. Usually, high memory consumption is a result of poor coding or overly large SQL statements that join many tables. In a DSS or data warehouse environment, large complex statements may be normal. In an OLTP database, large or complex statements are usually the result of overnormalization of the database design, using an OLTP system as a data warehouse or DSS, or poor coding techniques. Usually, large statements result in excessive parsing, recursion, and large CPU usage.

Version Count

High version counts are usually due to multiple identical-schema databases, unsafe bind variables, or Oracle bugs. In Oracle 9 and Oracle 10, there are bugs that result in unsafe bind variables driving multiple versions. Multiple versions eat up SQL memory space in the shared pool. High version counts can also result in excessive parsing.

Tip

Setting the undocumented parameter `_sqlexec_progression_cost` to higher than the default of 1,000 decreases versioning in susceptible versions.

High values for sharable memory in the SQL pool can indicate issues if you aren't seeing good performance along with high sharable memory for statements with executions greater than 1.

Cluster Wait Time

As the name implies, the cluster wait time will be present only if you are using a RAC system. SQL that transfers a large number of SQL statements across the interconnect will be listed in this section. High levels of block transfer occur if the block size is too large, the DB caches on each server are too small, or the SQL is using too much of the table data. Large update statements may appear here, as updates require block transfers in many cases for current blocks. High levels of global cache-type wait events indicate you need to check this section for causative SQL statements.

Instance Activity Statistics

The next section deals with instance activity statistics. The biggest problem with the instance activity statistics is that there are so many of them, and many are not useful except in specific tuning situations that may never occur in a normal database. The

example excerpt from the report we have been examining is presented in [Listing 9.12](#). Statistics that aren't normally a concern have been removed.

Listing 9.12 Instance Activity Statistics Section

[Click here to view code image](#)

| Instance Activity Stats | | DB/Inst: |
|-----------------------------------|------------|----------|
| AULTDB/aultdb1 Snaps: 91-92 | | |
| Statistic | | Total |
| Second | per Trans | per |
| CPU used by this session | 77,997 | 21.6 |
| CPU used when call started | 288,270 | 79.8 |
| Number of read IOs issued | 27,685 | 7.7 |
| SQL*Net roundtrips to/from client | 6,970 | 1.9 |
| bytes received via SQL*Net from | 2,385,638 | 660.2 |
| bytes sent via SQL*Net to client | 2,595,626 | 718.4 |
| consistent gets | 22,777,682 | 6,303.9 |
| consistent gets - examination | 6,073,207 | 19,385.3 |
| | 1,680.8 | 5,168.7 |
| consistent gets direct | 3,277,142 | 907.0 |
| consistent gets from cache | 14,648,585 | 4,054.1 |
| consistent gets from cache (fast) | 193,221 | 53.5 |
| db block changes | | 12,466.9 |
| db block gets | 12,812 | 3.6 |
| db block gets direct | 13,389 | 10.9 |
| db block gets from cache | 3.7 | 11.4 |
| db block gets from cache (fastpa) | 13,364 | 1.0 |
| dirty buffers inspected | 3,512 | 0.2 |
| enqueue timeouts | 40 | 3.0 |
| enqueue | | 0.0 |
| | | 0.0 |

| | | | |
|-------------------------------|-----------|-------|---------|
| waits | 499 | 0.1 | 0.4 |
| execute | | | |
| count | 13,287 | 3.7 | 11.3 |
| free buffer | | | |
| inspected | 556,747 | 154.1 | 473.8 |
| free buffer | | | |
| requested | 731,667 | 202.5 | 622.7 |
| gc CPU used by this | | | |
| session | 11,859 | 3.3 | 10.1 |
| gc blocks | | | |
| lost | 0 | 0.0 | 0.0 |
| gc cr block build | | | |
| time | 1 | 0.0 | 0.0 |
| gc cr block flush | | | |
| time | 7 | 0.0 | 0.0 |
| gc cr block receive | | | |
| time | 66 | 0.0 | 0.1 |
| gc cr block send | | | |
| time | 3 | 0.0 | 0.0 |
| gc cr blocks | | | |
| received | 361 | 0.1 | 0.3 |
| gc cr blocks | | | |
| served | 522 | 0.1 | 0.4 |
| gc current block flush | | | |
| time | 2 | 0.0 | 0.0 |
| gc current block pin | | | |
| time | 205 | 0.1 | 0.2 |
| gc current block receive | | | |
| time | 16,726 | 4.6 | 14.2 |
| gc current block send | | | |
| time | 577 | 0.2 | 0.5 |
| gc current blocks | | | |
| received | 95,445 | 26.4 | 81.2 |
| gc current blocks | | | |
| served | 93,484 | 25.9 | 79.6 |
| index fast full scans (direct | | | |
| re | 90 | 0.0 | 0.1 |
| index fast full scans | | | |
| (full) | 4 | 0.0 | 0.0 |
| index fast full scans (rowid | | | |
| ran | 90 | 0.0 | 0.1 |
| index fetch by | | | |
| key | 3,086,965 | 854.3 | 2,627.2 |
| index scans | | | |
| kdiixs1 | 29,551 | 8.2 | 25.2 |
| leaf node 90-10 | | | |
| splits | 19 | 0.0 | 0.0 |
| leaf node | | | |
| splits | 26 | 0.0 | 0.0 |
| opened cursors | | | |

| | | | |
|-------------------------------------|----------------|-------------|--------------|
| cumulative | 13,077 | 3.6 | 11.1 |
| parse count
(failures) | 2 | 0.0 | 0.0 |
| parse count
(hard) | 153 | 0.0 | 0.1 |
| parse count
(total) | 7,202 | 2.0 | 6.1 |
| parse time
cpu | 227 | 0.1 | 0.2 |
| parse time
elapsed | 399 | 0.1 | 0.3 |
| physical read IO
requests | 550,974 | 152.5 | 468.9 |
| physical read
bytes | 32,562,569,216 | 9,011,916.7 | 27,712,824.9 |
| physical read total IO
requests | 605,019 | 167.4 | 514.9 |
| physical read total
bytes | 32,711,421,952 | 9,053,112.7 | 27,839,508.0 |
| physical read total multi
block | 30,330 | 8.4 | 25.8 |
| physical
reads | 9,773,380 | 2,704.9 | 8,317.8 |
| physical reads
cache | 572,745 | 158.5 | 487.4 |
| physical reads cache
prefetch | 153,965 | 42.6 | 131.0 |
| physical reads
direct | 3,402,178 | 941.6 | 2,895.5 |
| physical reads direct
temporary | 124,434 | 34.4 | 105.9 |
| physical reads prefetch
warmup | 58,580 | 16.2 | 49.9 |
| physical write IO
requests | 4,983 | 1.4 | 4.2 |
| physical write
bytes | 1,037,123,584 | 287,031.1 | 882,658.4 |
| physical write total IO
requests | 15,031 | 4.2 | 12.8 |
| physical write total
bytes | 1,085,801,472 | 300,503.1 | 924,086.4 |
| physical write total multi
block | 4,062 | 1.1 | 3.5 |
| physical
writes | 314,090 | 86.9 | 267.3 |
| physical writes
direct | 124,459 | 34.4 | 105.9 |
| physical writes direct
(lob) | 0 | 0.0 | 0.0 |
| physical writes direct | | | |

| | | | | |
|---------------------------------|-------------|----------|-----------|--|
| temporary | 124,434 | 34.4 | 105.9 | |
| physical writes from cache | 2,143 | 0.6 | 1.8 | |
| physical writes non checkpoint | 124,952 | 34.6 | 106.3 | |
| recursive calls | 78,415 | 21.7 | 66.7 | |
| recursive cpu usage | 77,189 | 21.4 | 65.7 | |
| redo entries | 7,832 | 2.2 | 6 | |
| redo log space requests | 2 | 0.0 | 0.0 | |
| redo log space wait time | 28 | 0.0 | 0.0 | |
| redo size | 2,892,568 | 800.5 | 2,461 | |
| redo synch time | 66 | 0.0 | 0.1 | |
| redo synch writes | 72 | 0.0 | 0.1 | |
| redo wastage | 196,192 | 54.3 | 167 | |
| redo write time | 1,110 | 0.3 | 0.9 | |
| redo writes | 701 | 0.2 | 0 | |
| rollback changes - undo records | 0 | 0.0 | 0.0 | |
| session cursor cache hits | 12,415 | 3.4 | 10.6 | |
| session logical reads | 22,791,070 | 6,307.6 | 19,396.7 | |
| sorts (memory) | 3,875 | 1.1 | 3.1 | |
| sorts (rows) | 1,460,468 | 404.2 | 1,243.0 | |
| summed dirty queue length | 3,284 | 0.9 | 2.8 | |
| table fetch by rowid | 1,322,667 | 366.1 | 1,125.7 | |
| table fetch continued row | 13 | 0.0 | 0.0 | |
| table scan blocks gotten | 2,780,775 | 769.6 | 2,366.6 | |
| table scan rows gotten | 158,164,979 | 43,773.3 | 134,608.5 | |
| table scans (direct read) | 776 | 0.2 | 0.7 | |
| table scans (long | | | | |

| | | | |
|----------------------------------|-----------|-------|---------|
| tables) | 776 | 0.2 | 0.7 |
| table scans (rowid
ranges) | 776 | 0.2 | 0.7 |
| table scans (short
tables) | 2,255 | 0.6 | 1.9 |
| transaction
rollbacks | 0 | 0.0 | 0.0 |
| undo change vector
size | 1,870,904 | 517.8 | 1,592.3 |
| user I/O wait
time | 445,246 | 123.2 | 378.9 |
| user
calls | 7,943 | 2.2 | 6 |
| user
commits | 794 | 0.2 | 0 |
| user
rollbacks | 381 | 0.1 | 0 |
| workarea executions -
onepass | 6 | 0.0 | 0.0 |
| workarea executions -
optimal | 2,323 | 0.6 | 2.0 |

Instance Activity Stats - Absolute Values

DB/Inst: AULTDB/aultdb1 Snaps: 91-9

-> Statistics with absolute values (should not be diffed)

| Statistic | Begin Value | End Value |
|----------------------------|----------------|---------------|
| session pga memory max | 544,192,924 | 4,940,081,136 |
| session cursor cache count | 2,266 | 8,279 |
| session uga memory | 73,033,165,084 | 3.393545E+11 |
| opened cursors current | 48 | 54 |
| workarea memory allocated | 0 | 16,041 |
| logons current | 41 | 47 |
| session uga memory max | 4,427,536,236 | 5,963,059,828 |
| session pga memory | 390,773,148 | 826,689,340 |

Instance Activity Stats - Thread Activity

DB/Inst: AULTDB/aultdb1 Snaps: 91-92

-> Statistics identified by '(derived)' come from sources other
than SYSSTAT

| Statistic | Total | per Hour |
|------------------------|-------|----------|
| log switches (derived) | 1 | 1.00 |

It is best to focus on the larger summary-type statistics, at least at first, when dealing with this section of the reports. Even the pared-down list in [Listing 9.12](#) still has many entries that may not really help novices find problems with their database.

Note

One of the biggest hurdles to understanding the statistics in the Instance Activity section is knowing what the time units are for the time-based statistics. Generally, if they aren't stated, they will be in milliseconds for some RAC reports and in centiseconds for single-instance reports.

The DB time value of 8,371.3 seconds is made up of user I/O time of 4,452.46 seconds plus CPU time of 995 seconds, and the remainder is classified as non-idle, non-I/O wait time.

So what else is contained in the mine of data? We have an *effective IO time* and the number of I/Os issued. From this we can see that each I/O cost an effective time of 32 milliseconds. No wonder we spent about 45 percent of the time waiting on I/O!

By looking at SQLNet roundtrips, we can tell if our application is making effective use of array processing. If it is taking hundreds or thousands of roundtrips per transaction, then we really need to examine how our application is handling arrays. By default, languages such as C and Java process only 10 to 20 records at a time. SQL\*Plus defaults to 10. By increasing array processing via precompiler flags or by the SET ARRAYSIZE command in SQL\*Plus, we can greatly reduce roundtrips and improve performance.

Bytes sent to and received from the clients via SQLNet can be used with roundtrips to see how large a chunk is being shipped between the client and the server, allowing insight into possible network tuning. In addition this information can be used to see if the network is being strained (generally, 1 Gb Ethernet can handle transfers of about 100 MB/sec).

Consistent Get Statistics

Consistent gets deal with logical reads and can be heavyweight (using two latches, as in a normal, consistent get) or lightweight (using one latch, as in a consistent get). Large numbers of consistent gets can be good or, if they are excessive because of poor index or database design, bad because they can consume CPU best used for other things. These statistics are used in conjunction with others, such as those involving heavy-hitter SQL statements, to diagnose database issues.

DB Block Get Statistics

DB block gets are current mode gets. A current mode get is for the data block in its current incarnation, with incarnation defined as all permanent changes applied (i.e., if the database shut down now and restarted, this is the block you would get). Now, this

block can be from cache, from another instance cache, from the file system cache, or from disk. Sometimes it will result in a disk read or a block transfer from another instance's cache, as there can be only one version of the current block in the cache of an instance at a time.

Dirty Block Statistics

The dirty buffers inspected statistic tells you how many times a dirty buffer (one that has changes) was looked at when the processes were trying to find a clean buffer. If this statistic is large, then you probably don't have enough buffers assigned, as the processes are continually looking at dirty buffers to find clean ones.

Enqueue Statistics

The enqueue statistics dealing with deadlocks, timeouts, and waits tell you how often processes were waiting on enqueues and if they were successful. High numbers of enqueue deadlocks indicate there may be application-locking issues. High numbers of waits and failures indicate high levels of contention. You need to look in the enqueue section of the report to see the specific enqueues that are causing the problems.

Execution Count

The execution count statistic is used with other statistics to develop ratios to show on average how much of a specific resource or statistic applies to a single execution. This count can be misleading, however, if there are several long-running transactions and many short, supporting transactions. For example, a large DSS query that requires a number of recursive SQL operations to parse it will drive the executions up, but you are really only interested in the large DSS query and not the underlying recursive transactions unless they contribute to the DSS transaction itself.

Free Buffer Statistics

The free buffers requested and the free buffers inspected statistics show how many buffers, while not actually dirty, were being used by other processes and had to be skipped when searching for a free buffer. If the free buffers inspected is overly large and the dirty buffers inspected is also large, then look at commit frequency as well as possibly increasing total DB block buffers, as the cache is probably congested.

Global Cache (GC) Statistics

The GC statistics show the components that make up the send times for the consistent read (CR) and current blocks. The statistics for build, flush, and send for the respective type of block (CR or current) are added together and divided by the blocks of that type sent to determine the latency for that operation. The receive times can be divided by the number of blocks received to determine that latency (and should be compared with the send latency as calculated from the other instance's AWR report). By seeing the

components of the latency for send operations, you can determine if the issue is internal (build or flush times are large) or external (the send time is large). The GC and global enqueue statistics will be present only if RAC is being utilized.

Remember that if send or receive times are greater than the average disk latency, then the interconnect has become a source of performance bottlenecks and needs to be tuned or replaced with a higher-speed/larger-bandwidth interconnect, such as InfiniBand. If only one node is showing issues (send times excessive point to this node, receive times excessive point to the other nodes) then look to excessive load, TCP buffer settings, or NIC issues on that node.

The global enqueue statistics haven't been shown because they are not a large source of performance issues. If the messaging shows large latencies, it will also be shown in the global cache services, since global cache activity depends on the global enqueue service.

Index Scan Statistics

There are two main index scan statistics:

- **Index fetch by key:** This statistic is incremented for each INDEX (UNIQUE SCAN) operation that is part of a SELECT or data manipulation language (DML) statement execution plan.
- **Index scans kdiixs1:** This statistic is incremented for each index range scan operation that is not of the types index fast full scans, index full scan, and index unique scan.

By comparing the two values, you get an idea of the ratio of single index lookups versus range scan. In most systems single index lookups should predominate since they are usually more efficient. However, in DSS or data warehouse systems, scans or fast scans may become the dominant type of index activity.

Leaf Node Statistics

The leaf node statistics refer to index leaf nodes and tell you how much insert activity is happening in your database. The 10–90 splits show activity for monotonically increasing indexes (those that use sequences or dates, generally), and the 50–50 splits show other types of index activity such as text or random value indexes. If you see heavy 10–90 split operations, then you might want to look at index management operations to be sure your indexes aren't getting too broad due to excessive unused space in your sequence or date-based indexes. Usually, index rebuilds are required only in databases that have monotonically increasing indexes that also undergo large amounts of random deletions resulting in numerous partially filled blocks.

Open Cursors

The open cursors cumulative statistic is used with other statistics to calculate ratios for resources used per cursor or cursors open per login, for example.

Parse Statistics

The parse statistics are used to show how efficiently you are using parses. If you have large numbers of parse count failures or large numbers of hard parses, it could indicate a large number of ad-hoc queries. A large number of hard parses (greater than 10 percent of parses) indicates that the system probably isn't using bind variables efficiently. If there is a large discrepancy between parse CPU and parse elapsed times, it indicates that the system is overloaded and may be CPU bound.

Physical Read and Write Statistics

There are several types of physical read and write statistics, defined as follows:

- **Physical reads:** Shows the total number of data blocks read from disk. This value can be greater than the value of `physical reads direct` plus `physical reads cache`, as reads into process private buffers are also included in this statistic.
- **Physical read bytes:** Shows the total size in bytes of all disk reads by application activity (and not other instance activity) only.
- **Physical read I/O requests:** Shows the number of read requests for application activity (mainly buffer cache and direct load operation) that read one or more database blocks per request. This is a subset of the physical read total I/O requests statistic.
- **Physical read total bytes:** Shows the total size in bytes of disk reads by all database instance activity including application reads, backup and recovery, and other utilities. The difference between this value and physical read bytes gives the total read size in bytes by non-application workload.
- **Physical read total I/O requests:** Shows the number of read requests that read one or more database blocks for all instance activity including application, backup and recovery, and other utilities. The difference between this value and physical read total multiblock requests gives the total number of single-block read requests.
- **Physical read total multiblock requests:** Shows the total number of Oracle instance read requests that read in two or more database blocks per request for all instance activity including application, backup and recovery, and other utilities.
- **Physical reads cache:** Shows the total number of data blocks read from disk into the buffer cache. This is a subset of the physical reads statistic.
- **Physical reads direct:** Shows the number of reads directly from disk, bypassing the buffer cache. For example, in high bandwidth, data-intensive operations such as parallel query, reads of disk blocks bypass the buffer cache to maximize transfer rates and to prevent the premature aging of shared data blocks resident in the buffer cache.
- **Physical reads prefetch warmup:** Shows the number of data blocks that were read from the disk during the automatic prewarming of the buffer cache.

- **Physical write bytes:** Shows the total size in bytes of all disk writes from the database application activity (and not other kinds of instance activity).
- **Physical write I/O requests:** Shows the number of write requests for application activity (mainly buffer cache and direct load operation) that wrote one or more database blocks per request.
- **Physical write total bytes:** Shows the total size in bytes of all disk writes for the database instance including application activity, backup and recovery, and other utilities. The difference between this value and physical write bytes gives the total write size in bytes by non-application workload.
- **Physical write total I/O requests:** Shows the number of write requests that wrote one or more database blocks from all instance activity including application activity, backup and recovery, and other utilities. The difference between this stat and physical write total multiblock requests gives the number of single-block write requests.
- **Physical write total multi block requests:** Shows the total number of Oracle instance write requests that wrote two or more blocks per request to the disk for all instance activity including application activity, recovery and backup, and other utilities.
- **Physical writes:** Shows the total number of data blocks written to disk. This statistic's value equals the sum of physical writes direct and physical writes from cache values.
- **Physical writes direct:** Shows the number of writes directly to disk, bypassing the buffer cache (as in a direct load operation).
- **Physical writes from cache:** Shows the total number of data blocks written to disk from the buffer cache. This value is a subset of the physical writes statistic.
- **Physical writes non-checkpoint:** Shows the number of times a buffer is written for reasons other than advancement of the checkpoint. It is used as a metric for determining the I/O overhead imposed by setting the `FAST_START_IO_TARGET` parameter to limit recovery I/Os. (Note that `FAST_START_IO_TARGET` is a deprecated parameter.) Essentially, this statistic measures the number of writes that would have occurred had there been no checkpointing. Subtracting this value from physical writes gives the extra I/O for checkpointing.

Tip

To determine the average I/O unit for reads and writes, use the ratios of physical read total bytes and physical read total I/O requests, and physical write total bytes and physical write total I/O requests.

Recursive Statistics

The recursive calls statistics can be used in ratio with the user calls statistic to get the number of recursive calls per user call. If the number of recursive calls is high for each user call, then this indicates you are not reusing SQL very efficiently. In our example printout, the ratio is about 10 to 1, which is fine; if the ratio was 50 to 1 or greater, it would bear investigation.

Essentially, you need to determine what is a good ratio of recursive calls to user calls for your system, as it will depend on the number of tables on average in your queries, the number of indexes on those tables, and whether or not Oracle has to reparse the entire statement or can instead use a soft parse. This ratio is actually reported in the header information. We have already shown how the recursive CPU statistic is used with the CPU usage and other CPU-related timings.

Redo-Related Statistics

The redo-related statistics can be used to determine the health of the redo log activity and the LGWR processes. By dividing redo log space wait time by redo log space requests, you can determine the wait time per space request. If this time is excessive, it shows that the redo logs are under I/O stress and should be moved to less active disks or flash memory-based assets. In a similar calculation, the redo synch time can be divided by the redo synch writes to determine the time taken during each redo sync operation; this too is an indicator of I/O stress if it is excessive. A final indicator of I/O stress is a ratio of redo write time to redo writes giving the time for each redo write. The redo wastage statistic shows the amount of unused space in the redo logs when they were written. Excessive values of redo wastage per redo write indicates that the LGWR process is being stressed.

The rollback changes – undo records statistics is actually rollback changes – undo records applied, according to Jonathan Lewis. If a session's user rollbacks is large, but its rollback changes – undo records applied is small (and those numbers are relative to your system), then most of the rollbacks are doing nothing. So by comparing these two metrics, you can determine, relative to your system, if you have an undo issue.

Undo issues deal with rollback commands either explicit or implicit. Explicit are generated by issuing the rollback command, whereas implicit can be from data definition language (DDL), data control language (DCL), or improper session terminations.

Session Cursor Statistic

The session cursor cache hits statistic shows how often a statement issued by a session was actually found in the session cursor cache. The session cursor cache is controlled by the `session_cached_cursors` setting and defaults (usually) to 50. If session cursor cache hits divided by the sum of user calls plus recursive calls is low, then you need to increase the setting of `session_cached_cursors`. Settings from 100 to 150 or higher are usually recommended.

Sort Statistics

The sorts statistics—sorts (rows), sorts (memory), and sorts (disk)—show how the system is doing sorts. In later versions, you may see sorts (disk) replaced by the work area executions—one pass and work area executions—multipass statistics. Ideally, you want no sorts (disk) or work area executions—one pass or work area executions—mulitpass; however, in reality, this may be impossible to achieve, so seek to set `sort_area_size`, `hash_area_size`, `merge_bitmap_area_size`, `create_bitmap_area_size`, or `pga_aggregate_target` to large enough values to reduce sorts to disk as much as possible. Note that no statistic really tracks bitmap, hash, or global temporary table operations that overflow to disk, so it is possible to get temporary tablespace IOPS while having zero values for all disk-related sort and work area statistics.

Tip

The sort segment histogram section of the report will help you determine settings for the sort individual or the PGA aggregate parameter settings.

Summed Dirty Queue Length

The summed dirty queue length statistic can be used in concert with the physical writes from cache statistic to determine if the DBWR process is being stressed. If the ratio of summed dirty queue length to physical writes from cache is greater than 100, then more DBWR processes are needed.

Table Fetch Statistics

The table fetch statistics provide details on how table data has been accessed:

[Click here to view code image](#)

| Statistic | | Total | per |
|-----------------------|-------------|----------|-----------|
| Second | per Trans | | |
| <hr/> | | | |
| table fetch by | | | |
| <hr/> | | | |
| rowid | 1,322,667 | 366.1 | 1,125.7 |
| <hr/> | | | |
| table fetch continued | | | |
| row | 13 | 0.0 | 0.0 |
| <hr/> | | | |
| table scan blocks | | | |
| gotten | 2,780,775 | 769.6 | 2,366.6 |
| <hr/> | | | |
| table scan rows | | | |
| gotten | 158,164,979 | 43,773.3 | 134,608.5 |
| <hr/> | | | |
| table scans (direct | | | |
| read) | 776 | 0.2 | 0.7 |
| <hr/> | | | |
| table scans (long | | | |
| tables) | 776 | 0.2 | 0.7 |
| <hr/> | | | |
| table scans (rowid | | | |

| | | | |
|-------------------------------|-------|-----|-----|
| ranges) | 776 | 0.2 | 0.7 |
| table scans (short
tables) | 2,255 | 0.6 | 1.9 |

The three most important statistics are as follows:

- **Table fetch by rowid:** This is the cumulative number of rows fetched by index lookup of rowid.
- **Table scan rows gotten:** This shows the number of rows retrieved via table scan operations (full table scans).
- **Table fetch continued row:** This shows the number of row fetches that resulted in a continued row read, essentially doubling (or more) the I/Os. This could be an indication of chained rows, chained blocks, or binary large object (BLOB) activity.

Depending on your database type, you could have more index-based reads (usually OLTP) or more table scan-based reads (DWH, DSS). It is important to have a feeling for the ratio of these two statistics (index versus scan rows) so any change in the profile will alert you to possible index issues. Also, monitoring the ratio of `continued row` reads to the sum of `scan` and `index row` reads will inform you if you are getting excessive chained row activity.

Table scans (direct reads) are usually indicative of parallel query activity. Table scans (rowed ranges) are usually also caused by parallel query operations.

The two table type scans, long and short, are based on whether a table is less than a certain percentage of the size of the DB cache. For example, some releases set the boundary between short and long table scans at 2 percent of the DB cache size. Generally, the number of short table scans should be much greater than long table scans in a properly indexed environment.

Transaction Rollback

The transaction rollbacks statistic is for rollbacks that actually do work. This statistic will also track the rollbacks done automatically. For example, when an update occurs for multiple rows but has to be backed out and restarted by Oracle because of blocking locks, as this attempt to update-block-rollback may occur several times for a single transaction, you will see several transaction rollback increments even though a single transaction actually occurred. If this statistic is high, then check for conflicting transactions that lock many rows.

Undo Change Vector Statistic

The undo change vector size statistic is a cumulative count in bytes of the size of undo records. By calculating a ratio of undo change vector size to user calls, you can get an idea of the amount of undo being generated per user call. If the undo being generated is large, then look for excessive write activity to the undo tablespace, as this could be slowing down your transactions and causing stress on the I/O subsystem. If your

application generates large amounts of undo by nature, consider moving the undo tablespace to high-speed disk or flash-based assets.

User Statistics

The user I/O wait time statistic is the cumulative I/O wait time. Use this statistic with user calls to determine the average I/O wait time per user call. This can also be used with the physical reads plus the physical writes to determine the average I/O wait time per I/O. If the I/O wait time becomes an issue, you can add physical storage (disks), increase the amount of disk caching, or place hot tables and indexes on flash-based storage.

Note

The user I/O wait time statistic has been removed from the report starting in Oracle Database 11.2.0.4. You will need to sum the I/O wait times in the service-related statistics to get this information in releases 11.2.0.4 and later.

The user commits and user rollbacks are used with other statistics to determine weighting. If the number of user rollbacks is high compared to user commits, look to ad-hoc SQL or improper session termination as the possible cause. Before the Oracle database supported the MERGE and UPSERT commands, programmers would process to an exception for INSERT and UPDATE decisions; in other words, they would attempt an INSERT and, if it failed giving an exception, then the transaction would roll back and the programmer would then trigger an update. This type of processing results in large amounts of undo activity and should be avoided.

Work Area Statistics

The work area statistics were already discussed in the section on sort statistics. Essentially, the most desired work area statistic is `workarea_executions-optimal`, as these were done within the `PGA_AGGREGATE_TARGET` settings in memory. Any of the other work area statistics indicate a sort to storage. If there seem to be excessive `workarea_executions-optimal`, then look to eliminate unneeded sorts such as the use of distinct from improper joins, sorts that could be eliminated using a multicolumn index or unneeded order by and group by operations.

Instance Activity Statistics—Absolute Values

The absolute values show the high-watermarks for various memory, login, and other cumulative statistics that cannot be diffed. Using ratios of logins to the various PGA and user global area (UGA) memory allocations can show possible settings for `PGA_AGGREGATE_TARGET` values or sort area values. Remember that these values are incremented each time a process is created and adds to its PGA or UGA areas. That contribution is not removed once the session logs out, so the actual totals may bear little resemblance to the real values. Use these statistics for information only.

Instance Activity Statistics—Thread Activity

The thread activity statistic shows the number of (projected) redo logs switched per hour. The old way was to have redo logs switch every 15 minutes. This rule of thumb to switch every 15 minutes may or may not apply to your database—it is best to tune redo log size according to the needs of your system. However, excessive redo log switching (such as once a minute with a 5 MB log) should be avoided, as this generates lots of I/O, latch, and CPU overhead.

Summary

In this chapter, we discussed the first half of the AWR report up through the Instance Activity Statistics. In the next chapter, we examine the rest of the AWR report and see how to use it to analyze our database performance issues.

10. Database Forensics and Tuning Using AWR Analysis: Part II

[Chapter 9](#) covered the review and analysis of the first half of the Automatic Workload Repository (AWR) report. In this chapter, we complete this analysis.

Tablespace I/O Statistics

The next section of the AWR report deals with the tablespace I/O statistics. There are two parts to the tablespace I/O statistics: the first part rolls up the I/O statistics by tablespace, and the second lists the statistics by datafile, as each tablespace may have multiple datafiles associated with it. [Listing 10.1](#) shows an example of this section of the report.

Listing 10.1 Tablespace I/O Statistics Section

[Click here to view code image](#)

```
Tablespace IO Stats                               DB/Inst:
AULTDB/aultdb1  Snaps: 91-92
-> ordered by IOs (Reads + Writes) desc
Tablespace
-----
          Av      Av      Av      Av      Buffer
Av Buf   Reads  Reads/s  Rd(ms) Blks/Rd   Writes  Writes/s    Waits
Wt (ms)
-----
-----
```

| | Av | Av | Av | Av | Buffer | | |
|----------|---------|---------|--------|---------|--------|----------|-------|
| Av Buf | Reads | Reads/s | Rd(ms) | Blks/Rd | Writes | Writes/s | Waits |
| Wt (ms) | | | | | | | |
| DATA | 512,639 | 142 | 11.8 | 6.4 | 0 | 0 | 6 |
| INDEXES | 32,625 | 9 | 11.3 | 16.7 | 0 | 0 | 37 |
| TEMP | 4,024 | 1 | 17.6 | 30.9 | 4,014 | 1 | 0 |
| SYSAUX | 571 | 0 | 29.3 | 1.4 | 698 | 0 | 0 |
| SYSTEM | 471 | 0 | 5.3 | 1.8 | 56 | 0 | 0 |
| UNDOTBS1 | 9 | 0 | 10.0 | 1.0 | 215 | 0 | 1 |
| USERS | 1 | 0 | 10.0 | 1.0 | 0 | 0 | 0 |

```
File IO Stats                               DB/Inst:
```

| AULTDB/aultdb1 Snaps: 91-92 | | -> ordered by Tablespace, File | | | | | |
|-----------------------------|---------|--------------------------------|--|-------|--------|----------|--------|
| Tablespace | | Filename | | | | | |
| Av Buf | | Av | Av | Av | | Av | Buffer |
| Reads | Reads/s | Rd(ms) | Blks/Rd | | Writes | Writes/s | Waits |
| Wt (ms) | | | | | | | |
| DATA | | | +DATA/aultdb/datafile/data.257.660765277 | | | | |
| 501,566 | 139 | 10.8 | 5.2 | 0 | 0 | 0 | 6 |
| DATA | | | +DATA/aultdb/datafile/data.259.660843403 | | | | |
| 11,073 | 3 | 56.9 | 64.5 | 0 | 0 | 0 | 0 |
| INDEXES | | | +DATA/aultdb/datafile/indexes.258.66076543 | | | | |
| 32,625 | 9 | 11.3 | 16.7 | 0 | 0 | 37 | |
| SYSAUX | | | +DATA2/aultdb/datafile/sysaux.257.66075592 | | | | |
| 571 | 0 | 29.3 | 1.4 | 698 | 0 | 0 | |
| SYSTEM | | | +DATA2/aultdb/datafile/system.256.66075592 | | | | |
| 471 | 0 | | | | | | |
| 5.3 | 1.8 | | 56 | 0 | 0 | 0.0 | |
| TEMP | | | +DATA2/aultdb/tempfile/temp.266.660756117 | | | | |
| 1,340 | 0 | 17.5 | 31.0 | 1,338 | 0 | 0 | 0 |
| TEMP | | | +DATA2/aultdb/tempfile/temp.272.660915285 | | | | |
| 2 | 0 | 10.0 | 1.0 | 0 | 0 | 0 | |
| TEMP | | | +DATA2/aultdb/tempfile/temp.273.660930207 | | | | |
| 2 | 0 | 10.0 | 1.0 | 0 | 0 | 0 | |
| TEMP | | | +DATA2/aultdb/tempfile/temp.274.660989059 | | | | |
| 1,340 | 0 | 17.7 | 31.0 | 1,338 | 0 | 0 | 0 |
| TEMP | | | +DATA2/aultdb/tempfile/temp.275.661003761 | | | | |
| 1,340 | 0 | 17.6 | 31.0 | 1,338 | 0 | 0 | |
| UNDOTBS1 | | | +DATA2/aultdb/datafile/undotbs1.258.660755 | | | | |
| 9 | 0 | 10.0 | 1.0 | 215 | 0 | 1 | |
| USERS | | | +DATA2/aultdb/datafile/users.259.660755929 | | | | |
| 1 | 0 | 10.0 | 1.0 | 0 | 0 | 0 | |

The tablespace I/O section of the AWR report can run to several pages if you have many tablespaces, each with multiple datafiles. In situations where partitions are being used and each is given its own tablespace or datafile, this is often the case.

Note

Notice that the timing for writes is not a part of the report; this is because Oracle doesn't stress write tuning, since it uses delayed block cleanout for most writes and writes the blocks back to the tablespaces only when needed. However, redo writes, undo writes, and temporary tablespace writes fall outside the normal

writes in that they are done immediately.

When reviewing this section of the AWR report, watch for the tablespaces that are showing high numbers of reads and writes and are showing high average read milliseconds and high numbers of buffer waits. High values for read millisecond and buffer waits indicates I/O stress and possible memory starvation for the instance. If buffer waits are not indicated, but there is still a high value for read milliseconds, then the I/O subsystem is being stressed.

For tablespaces or datafiles that are exhibiting I/O stress, make sure there is adequate memory. Then try tuning the SQL that accesses objects within them, while considering expanding disk resources or the use of flash technology. Of course, it may be more expensive to fix the application if it is large or from a third-party vendor. In that case, expanding existing disk assets or use of flash technology may be the only way to correct excessive read and write times.

Review the I/O reports when the top five events are I/O related. Correlate the objects accessed with the top SQL statements for physical reads to the tablespace- and datafile-level statistics. By examining the average blocks per read, you can determine whether the access to the tablespace is predominantly index based (the value is closer to 1 block per read), the activity is predominantly full table or index scans (the value is close to BD file multiblock read count), or the access is direct (the value is higher than DB file multiblock read count).

Tip

If the temporary tablespace shows high levels of I/O in spite of sorts (disk) or work area executions single pass and multipass being zero, then look at the use of hash joins in the `v$sql_plan` table and also look for global temporary table and bitmap usage, as these may be causing the temporary tablespace activity.

Buffer Pool Statistics

The next section of the report deals with how the buffer pools are being used. In the example used in this section, shown in [Listing 10.2](#), only one buffer pool, the default one, is shown. However, in your database there could be a keep, recycle, 2 K, 4 K, 8 K, 16 K, or 32 K (on 64 bit) in addition to your default block size pool (note that you cannot use the special `blocksize` designation parameter for the block size that is the same as your default pool).

Listing 10.2 Buffer Pool Statistics Section

[Click here to view code image](#)

-> Standard block size Pools D: default, K: keep, R: recycle
-> Default Pools for other block sizes: 2k, 4k, 8k, 16k, 32k

| | | | | | | Free |
|-------|----------------|------|---------|----------|---------------|------|
| Write | Buffer | | | | | |
| Comp | Number of Pool | Busy | Buffer | Physical | Physical Buff | |
| P | Buffers | Hit% | Gets | Reads | Writes | Wait |
| Wait | Waits | | | | | |
| D | 159,244 | 91 | | | | |
| | 6,287,434 | | 572,581 | 2,143 | 0 | 44 |

Instance Recovery Stats DB/Inst:

AULTDB/aultdb1 Snaps: 91-92

-> B: Begin snapshot, E: End snapshot

| Targt | Estd | | | | | | Log | File | Log |
|----------|--------|--------|------|-----------|-----------|-----------|------|------|------|
| Ckpt | Log | Ckpt | | | | | | | |
| MTTR | MTTR | | | | | | | | |
| Recovery | Actual | Target | Size | Timeout | Interval | | | | |
| (s) | (s) | Estd | IOs | Redo Blks | Redo Blks | Redo Blks | Blks | Blks | Redo |
| Blks | Redo | Blks | | | | | | | |
| B | 0 | 0 | 250 | 974 | 1015 | 92160 | | | 1015 |
| E | 0 | 0 | 292 | 1192 | 2751 | 92160 | | | 2751 |

Buffer Pool Advisory DB/Inst:

AULTDB/aultdb1 Snap: 91-92

-> Only rows with estimated physical reads >0 are displayed

-> ordered by Block Size, Buffers For Estimate

| P | Size for Est (M) | Size Factor | Buffers for Estimate | Read Factor | Estimated Physical Reads | |
|---|------------------|-------------|----------------------|-------------|--------------------------|-----------|
| | | | | | Est | Phys |
| D | 128 | .1 | 15,536 | 3.4 | | 1,988,649 |
| D | 256 | .2 | 31,072 | 2.5 | | 1,497,186 |
| D | 384 | .3 | 46,608 | 2.0 | | 1,196,087 |
| D | 512 | .4 | 62,144 | 1.6 | | 959,386 |
| D | 640 | .5 | 77,680 | 1.3 | | 787,130 |
| D | 768 | .6 | 93,216 | 1.1 | | 674,908 |
| D | 896 | .7 | 108,752 | 1.0 | | 620,121 |

| | | | | | |
|---|-------|-----|---------|-----|---------|
| D | 1,024 | .8 | 124,288 | 1.0 | 599,692 |
| D | 1,152 | .9 | 139,824 | 1.0 | 593,191 |
| D | 1,280 | 1.0 | 155,360 | 1.0 | 592,402 |
| D | 1,312 | 1.0 | 159,244 | 1.0 | 592,402 |
| D | 1,408 | 1.1 | 170,896 | 1.0 | 592,356 |
| D | 1,536 | 1.2 | 186,432 | 1.0 | 591,798 |
| D | 1,664 | 1.3 | 201,968 | 1.0 | 591,798 |
| D | 1,792 | 1.4 | 217,504 | 1.0 | 591,798 |
| D | 1,920 | 1.5 | 233,040 | 1.0 | 591,798 |
| D | 2,048 | 1.6 | 248,576 | 1.0 | 591,798 |
| D | 2,176 | 1.7 | 264,112 | 1.0 | 591,798 |
| D | 2,304 | 1.8 | 279,648 | 1.0 | 591,798 |
| D | 2,432 | 1.9 | 295,184 | 1.0 | 591,798 |
| D | 2,560 | 2.0 | 310,720 | 1.0 | 591,798 |

This report section has three parts: Buffer Pool Statistics, Instance Recovery Stats, and Buffer Pool Advisory.

Buffer Pool Statistics

The buffer pool statistics give the gross performance indicators for the buffer pool. The number of buffers, cache hit ratio, buffer gets, physical reads, physical writes, free buffer waits, write completion waits, and buffer busy waits are shown here.

Of the statistics shown, the most important are those dealing with waits. The free buffer waits statistic is a good indicator if you need more buffers in the pool where it shows a value greater than zero. The write complete waits occur if database writer (DBWR) cannot keep up with the writing of dirty blocks. A block that has been put on the write list cannot be reused until it has been written; when the buffer activity is such that DBWR can't write the blocks fast enough after they are on the write list, then write complete waits are generated. If you see write complete waits, try boosting the priority of DBWR and log writer (LGWR) processes or adding DBWR processes.

Buffer busy waits are indicative of an overloaded buffer cache where processes aren't releasing buffers fast enough; this can happen because of interested transaction lists (ITL) waits, locks, and other issues that prevent a session from taking ownership of a block in use by another session. Sometimes, increasing the number of buffers can help with buffer busy waits, but it can also be a signal of application locking issues or too large a block size. With too large a block size in later versions of Oracle, you may also see the *wait for other processes* wait, which is actually more descriptive of what is going on. Placing tables and indexes that are experiencing these types of waits into a tablespace with a smaller block size can help. Look in the sections on ITL waits later in the report to help pin down which objects are causing the problems.

Instance Recovery Statistics

The instance recovery statistics show how many blocks would need to be recovered if an instance crash were to occur. Essentially, any block that is not current at the time of the crash would need to be evaluated for roll-forward then rollback operations. Use this section to tune the various fast start and recovery initialization parameters.

Buffer Pool Advisory Section

The buffer pool advisory attempts to show you numerically what would happen if you increase or decrease the number of buffers in your buffer pool. It is a good practice to graph the size factor or the buffers for estimate against the estimated physical reads saved. [Figure 10.1](#) shows an example graph from the data provided in [Listing 10.2](#). Start with the 0.7 read factor value or higher or you may see odd results.

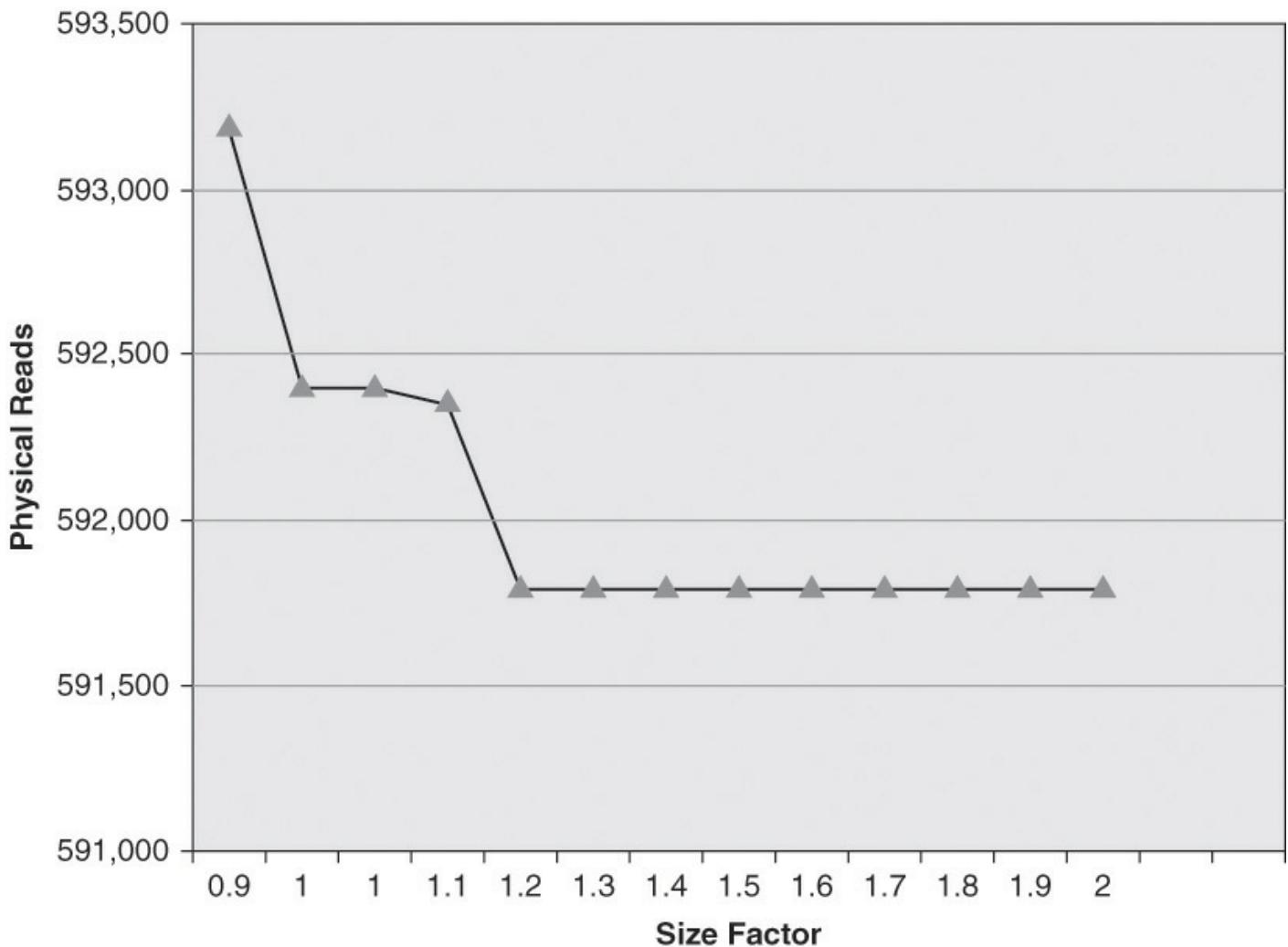


Figure 10.1 Example advisory plot

As you can see, we started our plot at 0.9. At 1.2 size factor, we see a drop in expected physical reads, and then the physical reads is expected to plateau out to past twice our current setting. Therefore, based on the advisory, increasing our DB cache size to 1.2 times the current setting should provide a benefit. There is another small decrease at 1.4, but the gain probably wouldn't be noticeable between 1.2 and 1.4.

PGA Statistics

The next section of the report deals with the program global area (PGA) and its settings (PGA\_AGGREGATE\_TARGET). You would look at tuning the PGA if you were seeing sorts to disk, work area executions in the single or multipass statistics, or excessive I/O to the temporary tablespace. [Listing 10.3](#) shows this report section.

Listing 10.3 PGA Statistics Section

[Click here to view code image](#)

| PGA Aggr Summary | | DB/Inst: AULTDB/aultdb1 Snaps: 91-92 | | | |
|---|------------------|--------------------------------------|---------------|--------------|--------------|
| -> PGA cache hit % - percentage of W/A (WorkArea) data processed only in-memory | | | | | |
| PGA Cache Hit % | W/A MB Processed | Extra W/A MB | Read/Written | | |
| 54.8 | 2,843 | | 2,345 | | |
| ----- | ----- | ----- | ----- | ----- | |
| PGA Aggr Target Stats | | DB/Inst: | | | |
| AULTDB/aultdb1 | Snaps: 91-92 | | | | |
| No data exists for this section of the report. | | | | | |
| ----- | ----- | ----- | ----- | ----- | |
| PGA Aggr Target Histogram | | DB/Inst: | | | |
| AULTDB/aultdb1 | Snaps: 91-92 | | | | |
| -> Optimal Executions are purely in-memory operations | | | | | |
| Low | High | Total Execs | Optimal Execs | 1-Pass Execs | M-Pass Execs |
| Optimal Execs | | | | | |
| ----- | ----- | ----- | ----- | ----- | ----- |
| 2K | 4K | 1,833 | 1,833 | 0 | |
| 64K | 128K | 5 | 5 | 0 | |
| 128K | 256K | 1 | 1 | 0 | |
| 256K | 512K | 6 | 6 | 0 | |
| 512K | 1024K | 439 | 439 | 0 | |
| 1M | 2M | 6 | 6 | 0 | |
| 2M | 4M | 6 | 6 | 0 | |
| 4M | 8M | 14 | 14 | 0 | |
| 8M | 16M | 6 | 6 | 0 | |
| 16M | 32M | 4 | 4 | 0 | |
| 64M | 128M | 3 | 3 | 0 | |

| | | | | |
|------|------|---|---|---|
| 256M | 512M | 6 | 0 | 6 |
|------|------|---|---|---|

PGA Memory Advisory DB/Inst:
AULTDB/aultdb1 Snap: 92
-> When using Auto Memory Mgmt, minimally choose a
pga\_aggregate\_target value where Estd PGA
Overallloc Count is 0

| PGA Target
Overalllo
Est %
Count | Size
Estd
Factr
Time | W/A
Processed | Estd
W/A
MB | Extra
Read/
Written | Estd
Cache | P | Estd
Hit | PGA |
|---|-------------------------------|------------------|-------------------|---------------------------|---------------|---|-------------|-----|
| 64
1.6E+05 | 0.1 | 3,388.1 | | 6,390.6 | 35.0 | | 22 | |
| 128
1.5E+05 | 0.3 | 3,388.1 | | 5,795.7 | 37.0 | | 2 | |
| 256
1.3E+05 | 0.5 | 3,388.1 | | 4,885.0 | 41.0 | | 1 | |
| 384 | 0.8 | 3,388.1 | | 1,172.5 | 74.0 | 0 | 7 | |
| 512 | 1.0 | 3,388.1 | | 1,172.5 | 74.0 | 0 | 7 | |
| 614 | 1.2 | 3,388.1 | | 1,172.5 | 74.0 | 0 | 7 | |
| 717 | 1.4 | 3,388.1 | | 1,172.5 | 74.0 | 0 | 7 | |
| 819 | 1.6 | 3,388.1 | | 1,172.5 | 74.0 | 0 | 7 | |
| 922 | 1.8 | 3,388.1 | | 1,172.5 | 74.0 | 0 | 7 | |
| 1,024 | 2.0 | 3,388.1 | | 1,172.5 | 74.0 | 0 | 7 | |
| 1,536 | 3.0 | 3,388.1 | | 1,172.5 | 74.0 | 0 | 7 | |
| 2,048 | 4.0 | 3,388.1 | | 1,172.5 | 74.0 | 0 | 7 | |
| 3,072 | 6.0 | 3,388.1 | | 1,172.5 | 74.0 | 0 | 7 | |
| 4,096 | 8.0 | 3,388.1 | | 1,172.5 | 74.0 | 0 | 7 | |

There are four parts to the PGA statistics section of the AWR reports: PGA Aggregate Summary, PGA Aggregate Target Status, PGA Aggregate Target Histogram, and PGA Memory Advisor.

PGA Aggregate Summary

The PGA Aggregate Summary section of the AWR report shows the rolled-up usage data for the PGA area. It shows what it calls the PGA cache hit percentage and then the components that make up that percentage: the work area megabytes processed and the extra work area megabyte read/written. The PGA cache hit percentage is the total number of bytes processed in the PGA versus the total number of bytes processed plus extra bytes read/written in extra passes. In our example report, we see that we got only

54 percent as we processed 2 GB to disk in six single passes. This measure indicates that if this were a typical transaction profile that was captured in the AWR report, we would need to add memory to our `PGA_AGGREGATE_TARGET` to eliminate disk-based sorts.

PGA Aggregate Target Statistics

The example report didn't include a section on PGA aggregate target statistics, so a section from another report follows:

[Click here to view code image](#)

| | PGA Aggr W/A | Auto %Auto | PGA %Man | PGA Global Mem | W/A Alloc (M) | PGA Used (M) | PGA % Mem | W/A Mem | PGA W/A |
|---------------|--------------|------------|-----------|----------------|---------------|--------------|-----------|---------|---------|
| Target (M) | Target (M) | Target (M) | Alloc (M) | Used (M) | % Mem | W/A Mem | W/A | | |
| Mem Bound (K) | B 1,628 | 1,434 | 425.37 | 284.23 | 66.82 | 100.00 | 0.00 | | |

| | | | | | | | | | |
|---|-------|-------|--------|--------|-------|--------|------|--|--|
| E | 1,628 | 1,424 | 315.79 | 177.43 | 56.19 | 100.00 | 0.00 | | |
|---|-------|-------|--------|--------|-------|--------|------|--|--|

In this section of the PGA reports, the differences between the start and end AWR statistics collections are shown. This section is used to determine whether the PGA aggregate settings are adequate or should be increased. If the ending statistics show increases, then the system had to adjust the parameters on the fly (or they were manually adjusted). You need to review this section to be sure that someone didn't reduce or otherwise change the PGA profile during the test.

PGA Aggregate Target Histogram

The PGA aggregate target histogram is probably the most useful part of the PGA section of the AWR report. In this section, the histogram shows you the various memory sizes that were used during the time period measured. By examining where single or multipass executions occurred, you can decide what your `PGA_AGGREGATE_TARGET` setpoint needs to be.

A single process gets a maximum of 5 percent of the `PGA_AGGREGATE_TARGET` setting, up to the value of the undocumented parameter `_pga_max_size`. The largest assigned sort segment would be a hash segment, which by default is two times the normal sort segment. Therefore, you can simply multiply by 40 the high boundary for the sorts you want to eliminate to get the needed setpoint. In our case, the upper range is 512 MB, so 40 times 512 MB would be 20 GB. Unfortunately, in Oracle Database 11g, the setting for `_pga_max_size` is 500 MB, so we can't totally guarantee we could eliminate all the sorts. [Listing 10.4](#) shows the PGA histogram report section.

Note

In Oracle Database 11g, the maximum setting is 32 GB for `PGA_AGGREGATE_TARGET`. However, you can set this parameter to more than the physical memory that is on the machine; just be sure you don't have so many processes wanting to do sorts at the same time that it would cause swapping!

Listing 10.4 PGA Aggregate Target Histogram Section

[Click here to view code image](#)

| PGA Aggr Target Histogram | | DB/Inst: AULTDB/aultdb1 | Snaps: |
|---------------------------|---|-------------------------|---------------|
| 91-92 | -> Optimal Executions are purely in-memory operations | | |
| <hr/> | | | |
| Low | High | Total Execs | Optimal Execs |
| Optimal Execs | Optimal Execs | 1-Pass Execs | M-Pass Execs |
| <hr/> | | | |
| 2K | 4K | 224,226 | 224,226 |
| 64K | 128K | 1,560 | 1,560 |
| 128K | 256K | 1,150 | 1,150 |
| 256K | 512K | 4,298 | 4,298 |
| 512K | 1024K | 7,200,201 | 7,200,201 |
| 1M | 2M | 11,248 | 11,248 |
| 2M | 4M | 793 | 641 |
| 4M | 8M | 255 | 125 |
| 8M | 16M | 32 | 32 |
| 16M | 32M | 16 | 16 |
| 32M | 64M | 51 | 41 |
| 64M | 128M | 26 | 26 |
| 128M | 256M | 4 | 1 |
| 256M | 512M | | 3 |
| 512M | 1024M | 10 | 0 |
| 1G | 2G | 7 | 7 |
| <hr/> | | | |

One thing that you need to watch for is what is called *out-of-band-sorts* (OOBS). OOBS are sorts or temporary segments of less than 512 MB written to disk. In the PGA sort and temporary segment algorithm, the undocumented parameter `_pga_max_size` determines the amount of the `PGA_AGGREGATE_TARGET` assigned to each process. By default, `_pga_max_size` is set to 512 MB, so in theory, unless you have too many processes vying for temporary segments at one time, all sorts and temporary activities should be able to be handled if they are less than 512 MB.

Tip

If you are seeing sorts or temporary activity to disk with segment sizes smaller than 500 MB, something is causing sorts and temps to bypass the PGA automatic sort algorithm. The usual suspects for this are parallel processes. One major cause is having the `DISPATCHERS` parameter or `SHARED_SERVERS` parameter set when it is not needed. Having `DISPATCHERS` or `SHARED_SERVERS` set results in any parallel process using

`SORT_AREA_SIZE` to determine sort and temporary segment sizes. Since `SORT_AREA_SIZE` defaults to 64 KB, the system will drive these activities to physical I/O targets. Setting `SORT_AREA_SIZE` to the size of segments that dominate the histogram (or larger if possible) may help mitigate this problem.

PGA Memory Advisor

In theory, a tool that estimates the effect of increasing or decreasing the `PGA_AGGREGATE_TARGET` sounds like a good idea, but in practice, this part of the PGA section of the AWR report has never worked properly. For example, it recommends that we could reduce the size of the `PGA_AGGREGATE_TARGET` to 80 percent of what it is right now with no ill effects.

Tip

If the estimated PGA overallocation count column shows values, then that setting is too low. If it doesn't, and the estimated time column shows no decrease, then that is the good setting.

Yet, when we do the calculations to eliminate sorts to disk, we get a number much larger than what we currently have set. However, it is showing that even at the largest setting it thinks is available, we would see only a 74 percent PGA cache hit and would still go to disk. Trust your own calculations based on the histogram. [Listing 10.5](#) shows an example PGA Memory Advisory report.

Listing 10.5 PGA Memory Advisor Report

[Click here to view code image](#)

| PGA Memory Advisory | | | DB/Inst: | |
|---|----------|---------------|--------------|-----------------|
| AULTDB/aultdb1 Snap: 92 | | | | |
| -> When using Auto Memory Mgmt, minimally choose a pga_aggregate_target value where Estd PGA Overalloc Count is 0 | | | | |
| PGA | Estd PGA | | Estd Extra | Estd |
| PGA Target | Size | | W/A MB | W/A MB |
| Read/ | Cache | Overalloc | | |
| Est (%) | Factr | | Processed | Written to Disk |
| Count | | | | Hit |
| 625 | 0.1 | 915,596,365.5 | 25,373,976.7 | 97.0 |
| 1,250 | 0.3 | 915,596,365.5 | 12,295,085.9 | 99.0 |
| 2,500 | 0.5 | 915,596,365.5 | 10,692,609.6 | 99.0 |
| 3,750 | 0.8 | 915,596,365.5 | 10,131,856.2 | 99.0 |

| | | | | |
|--------|-----|---------------|-------------|-------|
| 5,000 | 1.0 | 915,596,365.5 | 9,588,475.5 | 99.0 |
| 6,000 | 1.2 | 915,596,365.5 | 3,147,080.7 | 100.0 |
| 7,000 | 1.4 | 915,596,365.5 | 3,113,640.4 | 100.0 |
| 8,000 | 1.6 | 915,596,365.5 | 3,103,539.7 | 100.0 |
| 9,000 | 1.8 | 915,596,365.5 | 3,106,428.3 | 100.0 |
| 10,000 | 2.0 | 915,596,365.5 | 3,088,330.4 | 100.0 |
| 15,000 | 3.0 | 915,596,365.5 | 3,088,330.4 | 100.0 |
| 20,000 | 4.0 | 915,596,365.5 | 3,088,330.4 | 100.0 |
| 30,000 | 6.0 | 915,596,365.5 | 3,088,330.4 | 100.0 |
| 40,000 | 8.0 | 915,596,365.5 | 3,088,330.4 | 100.0 |

Shared Pool Statistics

The shared pool section of the AWR report is another section that is often useless; there seem to be numerous problems with the algorithms that populate the advisories. [Listing 10.6](#) shows an example excerpt from the AWR report.

Listing 10.6 Shared Pool Statistics Section

[Click here to view code image](#)

```
Shared Pool Advisory                               DB/Inst:
AULTDB/aultdb1  Snap: 92
-> SP: Shared Pool      Est LC: Estimated Library Cache   Factr:
Factor
-> Note there is often a 1:Many correlation between a single
logical object in the Library Cache, and the physical number of
memory objects
Associated with it. Therefore comparing the number of Lib Cache
objects (e.g. in v$librarycache), with the number of Lib Cache
Memory Objects is invalid.
```

| Shared | SP | Est | Est LC | | Est LC | | Est LC | | | |
|--------|-------|-------|--------|-------|--------|-------|----------|-------|-------|-------|
| | | | LC | Time | Time | Load | Load | Est | LC | |
| LC | Saved | Saved | Time | Time | Mem | Obj | Size (M) | Factr | (M) | Mem |
| Obj | (s) | Factr | (s) | Factr | | Hits | (K) | | | |
| ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |
| 192 | .9 | 3 | 495 | 4,555 | 1.0 | 41 | 1.0 | | | |
| 224 | 1.0 | 33 | 4,350 | 4,555 | 1.0 | 41 | 1.0 | | | |
| 256 | 1.1 | 45 | 6,645 | 4,557 | 1.0 | 39 | 1.0 | | | |
| 288 | 1.3 | 45 | 6,645 | 4,558 | 1.0 | 38 | .9 | | | |
| 320 | 1.4 | 45 | 6,645 | 4,558 | 1.0 | 38 | .9 | | | |
| 352 | 1.6 | 45 | 6,645 | 4,558 | 1.0 | 38 | .9 | | | |

| | | | | | | | |
|-----|-----|----|-------|-------|-----|----|----|
| 384 | 1.7 | 45 | 6,645 | 4,558 | 1.0 | 38 | .9 |
| 416 | 1.9 | 45 | 6,645 | 4,558 | 1.0 | 38 | .9 |
| 448 | 2.0 | 45 | 6,645 | 4,558 | 1.0 | 38 | .9 |

Other than recommending that we increase the shared pool size by 10 percent, not much else of value is shown. However, if the header data about shared SQL shows problems, this section might help you reset some sizes. Using SGA\_MAX\_SIZE and SGA\_TARGET in Oracle Database 10g and higher can help Oracle adjust shared pool size. In Oracle11g, use either the aforementioned parameters or the MEMORY\_MAX\_SIZE and MEMORY\_TARGET parameters to allow automatic shared pool size adjustment.

Other Advisories

The AWR report contains three other advisories: SGA target, streams pool, and Java pool. An excerpt from the AWR report for the advisories is shown in [Listing 10.7](#).

Listing 10.7 AWR Advisories Section

[Click here to view code image](#)

| SGA Target Advisory | | | DB/Inst: | |
|---------------------|-----------------|-----------------|--------------|-------|
| SGA Target Size (M) | SGA Size Factor | Est DB Time (s) | Est Physical | Reads |
| <hr/> | | | | |
| 396 | 0.3 | 8,538 | 592,206 | |
| 792 | 0.5 | 8,536 | 592,206 | |
| 1,188 | 0.8 | 8,536 | 592,206 | |
| 1,584 | 1.0 | 8,536 | 592,206 | |
| 1,980 | 1.3 | 8,536 | 592,206 | |
| 2,376 | 1.5 | 8,542 | 592,206 | |
| 2,772 | 1.8 | 8,542 | 592,206 | |
| 3,168 | 2.0 | 8,542 | 592,206 | |

| Streams Pool Advisory | | | DB/Inst: | |
|--------------------------|-------------|--------------------|-------------------|----------------------|
| Size for Est Spill Count | Size Factor | Est Spill Time (s) | Est Unspill Count | Est Unspill Time (s) |
| <hr/> | | | | |
| 32 | 0.13 | 0 | 0 | 0 |
| 64 | 0.25 | 0 | 0 | 0 |

| | | | | | |
|-----|------|---|---|---|---|
| 96 | 0.38 | 0 | 0 | 0 | 0 |
| 128 | 0.50 | 0 | 0 | 0 | 0 |
| 160 | 0.63 | 0 | 0 | 0 | 0 |
| 192 | 0.75 | 0 | 0 | 0 | 0 |
| 224 | 0.88 | 0 | 0 | 0 | 0 |
| 256 | 1.00 | 0 | 0 | 0 | 0 |
| 288 | 1.13 | 0 | 0 | 0 | 0 |
| 320 | 1.25 | 0 | 0 | 0 | 0 |
| 352 | 1.38 | 0 | 0 | 0 | 0 |
| 384 | 1.50 | 0 | 0 | 0 | 0 |
| 416 | 1.63 | 0 | 0 | 0 | 0 |
| 448 | 1.75 | 0 | 0 | 0 | 0 |
| 480 | 1.88 | 0 | 0 | 0 | 0 |
| 512 | 2.00 | 0 | 0 | 0 | 0 |
| 544 | 2.13 | 0 | 0 | 0 | 0 |
| 576 | 2.25 | 0 | 0 | 0 | 0 |
| 608 | 2.38 | 0 | 0 | 0 | 0 |
| 640 | 2.50 | 0 | 0 | 0 | 0 |

Java Pool Advisory
AULTDB/aultdb1 Snap: 92

| LC | Java | JP | | Est | | Est | | LC | | Est | | LC | |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-----|--|
| | | Time | Size | Time | Load | Load | Est | LC | Pool | Size | Est | Mem | |
| LC | Saved | Saved | Time | Time | | | | | | | | | |
| Obj | (M) | Factr | (M) | Mem | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- | | |
| 16 | 1.0 | 2 | 79 | 7 | 1.0 | 41 | 1.0 | | | | | | |
| 32 | 2.0 | 4 | 163 | 7 | 1.0 | 41 | 1.0 | | | | | | |

SGA Target Advisory

If you have the SGA\_TARGET parameter set, the AWR report shows the SGA\_TARGET advisory. This advisory shows the effects from changing the size of the SGA\_TARGET parameter on your system. In the case of the example report, it is indicating that if we increased our SGA\_TARGET by 50 percent, we would see a marginal increase in efficiency of the memory management; however, the increase is so small, it probably isn't worth the effort unless other factors confirm the recommendation. Considering the other sections we have looked at, there is no real reason to increase the setting.

Streams Pool Advisory

When the STREAMS\_POOL\_SIZE parameter is set, the streams pool advisory is populated in the AWR report. Streams uses the streams pool to buffer the messages it sends to and receives from other systems. If the streams pool size is insufficient, these messages are queued to disk (spilled). Excessive disk spills by the streams processes result in poor performance of the streams processes. The advisory shows the effects of increasing or decreasing the current stream pool by showing the increase or decrease in spillage and the effect on performance with projected time in seconds for either performance losses (Est Spill Time) or gains (Est Unspill Time).

Java Pool Advisory

There was a time when the Java pool was rarely if ever used; it was set to 16 MB and ignored. Now Oracle does more and more of its work, especially with export and import and other utilities, using Java in the kernel, making the Java pool setting an important one to periodically review. The Java pool advisory shows the effects of increasing the pool size. In the example report, even if we doubled the pool size, there would be no net gain (other than in objects able to be stored). If we were seeing Java pool-related errors or our Java was running slow, then this report might help us determine whether a Java pool issue existed.

Buffer Waits Statistics

If we see that the buffer busy waits event is causing issues with performance, the next section of the AWR report would be where we look to see the class of objects causing the problem. [Listing 10.8](#) shows the Buffer Waits section of our example report.

Listing 10.8 Buffer Waits Statistics Section

[Click here to view code image](#)

| Buffer Wait Statistics | | DB/Inst: |
|--|-------|---------------------|
| AULTDB/aultdb1 Snaps: 91-92 | | |
| -> ordered by wait time desc, waits desc | | |
| <hr/> | | |
| Class | Waits | Total Wait Time (s) |
| Avg Time (ms) | | |
| ----- | ----- | ----- |
| data block | 43 | 4 |
| undo header | 1 | 0 |
| ----- | ----- | ----- |

In our example, we see only data block and undo header waits. However, there are several other types of buffer waits possible:

- File header block

- First-level bitmap block (bmb)
 - Second-level bmb
 - Segment header

Each of these buffer waits points us to a different type of issue.

The data block type of wait usually indicates an issue with block sharing between processes. Essentially, the block may be too big for its own britches, meaning that it has too many rows and too many users want it at the same time. Usually, this means someone was reading in the block when someone else requested it. Try reducing rows per block.

The undo header buffer waits may indicate an insufficient number of undo segments. If you are using automatic undo management, try reducing the `transactions_per_rollback_segment` parameter to bring more undo segments online.

File header block waits usually mean free list or free list group problems. You can find the segments causing the issues in the [Segments with ITL Waits](#) section of the AWR report, which appears after the latch sections. Try using Automatic Segment Space Management (ASSM) to relieve this type of contention.

The first- and second-level bmb waits indicate issues with the bitmap blocks used in ASSM to manage free space in tables (the bmb blocks take the place of traditional free lists and free list groups). This could be caused by too large a block size or extreme internal “whitespace” inside tables caused by deletions.

The segment header buffer wait is usually caused by an insufficient number of free lists or free list groups which causes serialization of access and buffer waits. If you are using ASSM, switching these back to manual management may help.

Enqueue Statistics

Enqueues are serialization mechanisms in Oracle. Oracle uses them to prevent multiple updates from happening against the same record at the same time, and they work with locking and latching to achieve this end. When the enqueue deadlocks, enqueue waits, or enqueue timeouts point to an enqueue issue, you need to check the Enqueue Statistics section of the AWR report. The Enqueue Statistics section of our example report is shown in Listing 10.9.

Listing 10.9 Enqueue Statistics Section

[Click here to view code image](#)

Enqueue Activity DB/Inst:
AULTDB/aultdb1 Snaps: 91-92
-> only enqueues with waits are shown
-> Enqueue stats gathered prior to 10g should not be compared with
10g
-> ordered by Wait Time desc, Waits desc

Enqueue Type (Request Reason)

| Requests
Time (ms) | Succ Gets
9 | Failed Gets
0 | Waits
9 | Wt Time (s)
0 | Av Wt
0 |
|--|----------------|------------------|------------|------------------|------------|
| <hr/> | | | | | |
| BF-BLOOM FILTER (allocation contention)
2,618 | 2,611 | 7 | 14 | 14 | |
| <hr/> | | | | | |
| TD-KTF map table enqueue (KTF dump entries)
9 | 9 | 0 | 9 | 0 | |
| <hr/> | | | | | |
| PS-PX Process Reservation
648 | 616 | 32 | 208 | 0 | |
| <hr/> | | | | | |
| CF-Controlfile Transaction
7,661 | 7,660 | 1 | 118 | 0 | |
| <hr/> | | | | | |
| TM-DML
5,559 | 5,559 | 0 | 16 | 0 | |
| <hr/> | | | | | |
| XL-ASM Extent Fault Lock (fault extent map)
14 | 14 | 0 | 1 | 0 | |
| <hr/> | | | | | |
| TT-Tablespace
1,125 | 1,125 | 0 | 30 | 0 | |
| <hr/> | | | | | |
| HW-Segment High Water Mark
76 | 76 | 0 | 12 | 0 | |
| <hr/> | | | | | |
| WF-AWR Flush
19 | 19 | 0 | 11 | 0 | |
| <hr/> | | | | | |
| TA-Instance Undo
12 | 12 | 0 | 8 | 0 | |
| <hr/> | | | | | |
| KO-Multiple Object Checkpoint (fast object checkpoint)
81 | 81 | 0 | 27 | 0 | |
| <hr/> | | | | | |
| FB-Format Block
8 | 8 | 0 | 8 | 0 | |
| <hr/> | | | | | |
| JQ-Job Queue
60 | 60 | 0 | 2 | 0 | |
| <hr/> | | | | | |
| PG-Global Parameter
4 | 4 | 0 | 2 | 0 | |
| <hr/> | | | | | |
| AF-Advisor Framework (task serialization)
7 | 7 | 0 | 1 | 0 | |
| <hr/> | | | | | |
| PI-Remote PX Process Spawn Status
18 | 18 | 0 | 8 | 0 | |
| <hr/> | | | | | |
| RS-Reclaimable Space (prevent aging list update)
4 | 4 | 0 | 4 | 0 | |
| <hr/> | | | | | |
| DR-Distributed Recovery
2 | 2 | 0 | 2 | 0 | |
| <hr/> | | | | | |
| PE-Parameter
8 | 8 | 0 | 2 | 0 | |
| <hr/> | | | | | |
| JS-Job Scheduler (job run lock - synchronize)
2 | 2 | 0 | 1 | 0 | |
| <hr/> | | | | | |
| UL-User-defined
2 | 2 | 0 | 1 | 0 | |

US-Undo Segment

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 |
|---|---|---|---|---|

The enqueue statistics tell you more about what is happening under the hood in the user processes and SQL processing. In the excerpt in [Listing 10.9](#), we see that the BF-BLOOM FILTER enqueue is our predominant enqueue for this time period. This BF enqueue is present only when parallel query is used, and it signifies that the database used a bloom filter mechanism to filter the results from one or more of the parallel query slaves during the parallel query operations. The BF type enqueue has been available only since Oracle 10g when the bloom filter was added to Oracle's repertoire.

The biggest issue with determining what enqueues are telling us is that they aren't always well documented and may involve a web search or search of www.oracle.com or metalink.oracle.com to resolve. In the V\$SESSION\_WAIT and V\$LOCK dynamic performance views, you will find more details about enqueue issues by looking at the P1 and P2 values listed and knowing the type of enqueue. To see what the various P1 and P2 values mean for a specific enqueue, run the following query in your instance:

[Click here to view code image](#)

```
column parameter1 format a15
column parameter2 format a15
column parameter3 format a15
column lock format a8

Select
    substr(name,1,7) as "lock",parameter1,parameter2,parameter3
from v$event_name
where name like 'eng%'
```

The list for Oracle 11g has 247 entries. For the BF enqueue, the additional information we could get would be the node number, parallelizer number, and bloom number.

Tip

You should concentrate on the enqueues that show a wait time, since if there is no time penalty (at least one that can be measured in whole milliseconds) associated with an enqueue, you don't really care how many times a process or group of processes waited on it.

Undo Segment Statistics

Most DBAs use automated undo management. Automated undo management essentially uses the process count and the TRANSACTIONS\_PER\_ROLLBACK\_SEGMENT parameter to determine when segments should be brought online. Initially, Oracle brings 10 segments online, then waits for the ratio of processes to

TRANSACTIONS\_PER\_ROLLBACK\_SEGMENT to exceed 10 to bring on a new one, and each time the ratio increments after that, a new segment is brought online.

Unfortunately, most of these segments just sit there taking up space, for, as we all know, a process doesn't translate directly into a new transaction. Systems have been tracked that have a hundred undo segments, of which only five are actually being used and the rest are sitting there idle and offline just taking up space. An example of the undo statistics section from our AWR report is shown in [Listing 10.10](#).

Note

In the beginning, there was “redo and rollback.” Then, in late Oracle 8, the terminology was changed to “redo and undo.” If you hear someone talking about rollback segments, they mean undo segments, and vice versa.

Listing 10.10 Undo Statistics

[Click here to view code image](#)

```
Undo Segment Summary           DB/Inst: AULTDB/aultdb1  Snaps:  
91-92  
-> Min/Max TR (mins) - Min and Max Tuned Retention (minutes)  
-> STO - Snapshot Too Old count, OOS - Out of Space count  
-> Undo segment block stats:  
-> uS - unexpired Stolen, uR - unexpired Released, uU - unexpired  
reUsed  
-> eS - expired Stolen, eR - expired Released, eU -  
expired reUsed
```

| Undo | Num | Undo | Number of | Max | Qry | Max | Tx |
|---------|-------------|--------------|-----------|-------|-----|----------|-----------|
| Min/Max | STO | | uS/uR/uU/ | | | | |
| TS# | Blocks (K) | Transactions | | Len | (s) | Concurcy | TR (mins) |
| OOS | eS/eR/eU | | | | | | |
| 2 | .1 | | 1,090 | 1,725 | | 3 | 18.8/42.8 |
| 0/0 | 0/0/0/0/0/0 | | | | | | |

| Undo | Segment | Stats | DB/Inst: |
|----------------|---------|-------|----------|
| AULTDB/aultdb1 | Snaps: | 91-92 | |

-> Most recent 35 Undostat rows, ordered by Time desc

| STO/ | End | Time | Num | Undo | Number of | Max | Qry | Max | Tx | Tun | Ret |
|----------|-----|------|-----------|--------|--------------|-----|-----|-------|--------|-----|-----|
| OOS | | | uS/uR/uU/ | Blocks | Transactions | Len | (s) | Concy | (mins) | | |
| eS/eR/eU | | | | | | | | | | | |
| | | | | | | | | | | | |

| | | | | | |
|-----------------|----|-----|-------|---|----|
| 04-Aug 12:56 | 14 | 167 | 890 | 3 | 29 |
| 0/0 0/0/0/0/0/0 | | | | | |
| 04-Aug 12:46 | 10 | 141 | 289 | 3 | 19 |
| 0/0 0/0/0/0/0/0 | | | | | |
| 04-Aug 12:36 | 10 | 163 | 1,725 | 2 | 43 |
| 0/0 0/0/0/0/0/0 | | | | | |
| 04-Aug 12:26 | 18 | 240 | 1,124 | 3 | 33 |
| 0/0 0/0/0/0/0/0 | | | | | |
| 04-Aug 12:16 | 9 | 133 | 901 | 2 | 29 |
| 0/0 0/0/0/0/0/0 | | | | | |
| 04-Aug 12:06 | 88 | 246 | 300 | 3 | 19 |
| 0/0 0/0/0/0/0/0 | | | | | |

Undo statistics tell us how efficiently the undo segments are being handled. Unfortunately, there is not much that can be done to tune them if we are using automatic undo management. You control three aspects of the undo segments when you use automatic management: size of the undo tablespace, placement of the undo tablespace datafiles, and the value of the parameter

`TRANSACTIONS_PER_ROLLBACK_SEGMENT`. You can use the undo advisor in Oracle Enterprise Manager (OEM), Grid Control, or Database Control to determine what Oracle suggests for the size of the undo tablespace based on historical AWR data (it defaults to 7 days' worth, the maximum retained by default).

By tweaking the `TRANSACTIONS_PER_ROLLBACK_SEGMENT`, you can also reduce the STO numbers (if you get them). `STO` stands for “snapshot too old,” or ORA-01555. The OOS column lists out-of-space errors, which don’t occur often and usually mean that you ran out of space in your tablespace or file system. By using the Undo Advisor and playing with the undo retention numbers, you can derive a tablespace size to prevent both STO and OOS errors. The `dba_rollback_segments` view provides detailed information on the undo segments if you want to see more.

Latch Statistics

The next part of the AWR report is the Latch Statistics area. Like the instance activity statistics, latch activity statistics provide a plethora of information contained. Unfortunately, most of it is not useful for tuning efforts and should be filtered out of the results set. However, Oracle must use some of it for internal tuning efforts, so we must use our own filters to remove the nonessential data. A reduced version of the full latch section (filtered to show the latches of concern in this environment) is shown in [Listing 10.11](#).

Listing 10.11 Latch Section of the AWR Report

[Click here to view code image](#)

Latch Activity DB/Inst:
AULTDB/aultdb1 Snaps: 91-92
-> "Get Requests", "Pct Get Miss" and "Avg Slps/Miss" are statistics for willing-to-wait latch get requests
-> "NoWait Requests", "Pct NoWait Miss" are for no-wait latch get requests
-> "Pct Misses" for both should be very close to 0.0

| NoWait Latch Name | Requests | Pct Get | Avg Slps | Wait Time | NoWait Requests | Mis: |
|-----------------------|----------|-----------|-----------|-----------|-----------------|-------|
| KJC message pool free | | | | | | |
| li | 14,582 | 0.3 | 0.0 | 0 | 15,463 | 0.1 |
| gc | | | | | | |
| element | | | 2,414,376 | 0.0 | 0.3 | 2 |
| gcs resource | | | | | | 7,880 |
| hash | | 1,861,484 | 0.0 | 0.5 | 1 | 6 |
| virtual circuit | | | | | | |
| queues | 1 | 0.0 | | 0 | 0 | N/A |

Latch Sleep Breakdown DB/Inst: AULTDB/aultdb1 Snaps: 91-92
-> ordered by misses desc

| Get Latch Name | Requests | Misses | Sleeps | Get: |
|-------------------|-----------|-----------|--------|--------|
| cache buffers | | | | |
| chains | 8,492,860 | 21,037 | 3 | 21,034 |
| simulator lru | | | | |
| latch | 1,823,879 | 12,065 | 311 | 11,774 |
| cache buffers lru | | | | |
| chain | 1,190,948 | 6,096 | 352 | 5,799 |
| gc | | | | |
| element | | 2,414,376 | 767 | 213 |
| KJCT flow control | | | | |
| latch | 443,643 | 735 | 11 | 725 |

Latch Miss Sources DB/Inst: AULTDB/aultdb1 Snaps:

91-92
-> only latches with sleeps are shown
-> ordered by name, sleeps desc

| NoWait | | | | |
|-------------------|---------------|--------|--------|---------|
| Name | Where | Misses | Sleeps | Sleeps: |
| cache buffers lru | | | | |
| chain kcbzgws_1 | | 0 | 248 | 272 |
| gc | | | | |
| element | kclnfnfndnewm | | 0 | 112 |
| gcs resource | | | | |
| hash | kjbassume | 0 | 88 | 0 |

Mutex Sleep Summary DB/Inst: AULTDB/aultdb1 Snaps: 91-92

No data exists for this section of the report.

Parent Latch Statistics DB/Inst: AULTDB/aultdb1 Snaps: 91-92

No data exists for this section of the report.

Child Latch Statistics DB/Inst: AULTDB/aultdb1 Snaps: 91-92

No data exists for this section of the report.

The proper use of latches in the Oracle environment can determine whether or not your application can scale.

Note

If the latch-related events, such as latch free, are not in the top five wait events, then latching is not an issue for your database, and your efforts should be spent elsewhere.

Analysis of the latch section of the report is critical if scalability has become a concern in your database. The latch section of the AWR report has six sections:

- **Latch Activity:** This section provides overall latch statistics.
- **Latch Sleep Breakdown:** This section lists the latches that have sleeps.
- **Latch Miss Sources:** This section lists the latches that were the source of most misses.
- **Mutex Sleep Summary:** In Oracle 10.2.0.1, some latches were switched to lighter weight mutexes. This section shows the waits related to these objects.
- **Parent Latch Statistics:** This section shows parent latches.
- **Child Latch Statistics:** This section shows the latches that are children latches to parent latches in the previous section.

Some of these sections are useful; it should be noted, however, that they contain lots of extraneous information. As was stated previously, this example section was paired down to just those that showed statistics of concern; the actual section is three pages long!

Latch Activity

In the Latch Activity section are two statistics that deal with percentages: Pct Get Misses and Pct NoWait Misses. These statistics show the latches you should be concerned with right now. If either or both show a value other than N/A or 0.0, then that latch is showing some stress. Now, should you be concerned with fractional percentages? Probably not.

As you can see in the section of the report shown in [Listing 10.11](#), the highest percentage in the example report is 0.3 percent for Pct Get Misses for the KJC message pool free list latch (the full name is KJC message pool free list), which is a RAC-related latch dealing with the Kernel Job Control message pool (Global Enqueue Services). Since the percentage is less than 1 percent, it is of no real concern. Probably the best source of information on these latches is Oracle Metalink, but it may take some digging to find anything.

Note

Many experts believe that the use of miss percentages in tuning latches may lead you to tune latches that are really not important in the scheme of things. You should be looking at the next section of the report on latch sleeps instead.

Latch Sleep Breakdown

When a process tries to get a latch, if there is no latch available, the process spins on the CPU waiting for the latch to become available. This is called a *sleep*, and latches with high sleeps may be having contention issues. Generally speaking, look at the latch with the highest percentage of sleeps as related to total sleeps— $(\text{sleeps}/\text{sum(sleeps)}) * 100$ from v\$Latch—if you have a high latch free situation, as this will be the latch most affecting performance.

The best way to tune latching issues is to ensure the database has a proper level of resources (generally, memory in the cache and shared pool areas), and be sure to use

bind variables and eliminate hot block issues.

Latches and Spin Count

Many experts point to the value of the undocumented parameter `_spin_count` as a possible source for latch spin issues. The spin count tells Oracle how many CPU cycles to wait for a latch before incrementing spin count. This value may be set as low as 2000 for many systems, but with higher-speed CPUs, this value is too low and can result in high latch sleeps when there really isn't a problem. The argument for adjusting this undocumented parameter is that the value is really dependent on the CPU speed, and the value of 2000 for the default was based on CPU speeds available when Oracle 7.0 was released!

Obviously, we have seen a huge increase in CPU speeds since Oracle 7 with no increase in the setting of `_spin_count`. However, spin should be adjusted only if the value of your runqueue either is due to CPU and not I/O or is less than the number of CPUs (assuming you are on Linux, UNIX, AIX, or HP-UX). Runqueue tells how many processes are waiting to execute and can be incremented by processes waiting on either I/O or CPU. If runqueue is greater than CPUs and CPUs show idle time while I/O wait is high, then the runqueue is due to I/O wait; and if runqueue is greater than CPUs and CPU utilization is near 100 percent with low or no I/O wait, then it is due to CPU.

In situations where there is a high runqueue, you can sometimes correct it if it is CPU or I/O related with an Oracle system by renicing the LGWR, DBWR, and, if using RAC, the Global Enqueue Service Monitor (LMON) processes. *Renicing* means to increase those processes' priorities.

So, by increasing the `_spin_count` parameter, sometimes to as high as 10,000 or more, improvements in throughput and reductions in overall wait times have been seen. However, this value will need to be tested for a proper setting in your environment. For a good paper describing latch tuning, see “Resolving Oracle Latch Contention” by Guy Harrison at

[http://guyharrison.squarespace.com/storage/Resolving\\_Oracle\\_Latch\\_Contention.pdf](http://guyharrison.squarespace.com/storage/Resolving_Oracle_Latch_Contention.pdf).

Latch Miss Sources

The Latch Miss Sources section shows which latches are missed during an attempted get operation. In this section, the important consideration is, once again, the latch with the highest level of sleeps. See the earlier section on latch sleeps for tuning suggestions.

Mutex Sleep Summary

As with latches, mutexes, which replace some latches in Oracle Database 10.2.0.1 and later releases, will sleep and spin. The mutex with the highest level of sleeps should be investigated for tuning.

Parent and Child Latches

The sections on parent and child latches are used to help isolate which latches and their children are causing issues. By determining which latch or child latch is sleeping or waiting, you can determine whether the problem is related to memory, bind variables, or hot blocks.

Segment Access Areas

The next several sections are similar to the SQL sections except they deal with segments. They are similar to the SQL sections because they slice and dice the various segment access issues and reveal which segments are showing specific forms of contention. [Listing 10.12](#) shows these sections of the report.

Listing 10.12 Segments Sections

[Click here to view code image](#)

```

Segments by Logical Reads           DB/Inst:
AULTDB/aultdb1  Snaps: 91-92
-> Total Logical Reads:      22,791,070
-> Captured Segments account for 41.8% of Total

Tablespace          Subobject Obj.       Logical
Owner      Name   Object
Name       Name   Type    Reads %Total
-----  -----  -----  -----
-----  -----  -----  -----
TPCH
INDEXES      H_ORDERS_IDX1          INDEX  4,294,720  18.8
TPCH        DATA      H_LINEITEM      TABLE  2,117,568
TPCH        DATA      H_ORDER        TABLE  1,017,136
TPCH      INDEXES      SUPPLIER_IDX1
                  INDEX      626,848   2.75
TPCH        DATA      H_SUPPLIER      TABLE  620,432
-----  -----  -----  -----
-----  -----  -----  -----


Segments by Physical Reads         DB/Inst:
AULTDB/aultdb1  Snaps: 91-92
-> Total Physical Reads:      9,773,380
-> Captured Segments account for 39.3% of Total

Tablespace          Subobject Obj.       Physical
Owner      Name   Object
Name       Name   Type    Reads %Total
-----  -----  -----  -----
-----  -----  -----  -----
TPCH        DATA      H_LINEITEM      TABLE  2,107,980
TPCH        DATA      H_ORDER        TABLE  894,131
** UNAVAIL ** UNAVAIL ** UNAVAI **  AILABLE **

```

| | | | | |
|-------|---------|------------|-------|---------|
| UNDEF | 511,994 | 5.24 | | |
| TPCH | DATA | H_PART | TABLE | 123,676 |
| TPCH | DATA | H_PARTSUPP | TABLE | 117,400 |

Segments by Row Lock Waits DB/Inst:
AULTDB/aultdb1 Snaps: 91-92

No data exists for this section of the report.

Segments by ITL Waits DB/Inst:
AULTDB/aultdb1 Snaps: 91-92

No data exists for this section of the report.

Segments by Buffer Busy Waits DB/Inst:
AULTDB/aultdb1 Snaps: 91-92

No data exists for this section of the report.

Segments by Global Cache Buffer Busy DB/Inst:
AULTDB/aultdb1 Snaps: 91-92
-> % of Capture shows % of GC Buffer Busy for each top segment compared
-> with GC Buffer Busy for all segments captured by the Snapshot

| of | Tablespace | Subobject | Obj. | GC |
|---------|------------|-------------|------|------|
| Owner | Name | Object Name | Name | Type |
| Capture | | | | Busy |

\*\* UNAVAIL \*\* UNAVAIL \*\* UNAVA \*\* AILABLE \*\*

| | | | | |
|-------|---------|---------------|-------|---|
| UNDEF | 9 | 81.82 | | |
| TPCH | INDEXES | H_ORDERS_IDX1 | INDEX | 1 |
| TPCH | INDEXES | PARTSUPP_IDX1 | INDEX | 1 |

Segments by CR Blocks Received DB/Inst: AULTDB/aultdb1 Snaps:
91-92
-> Total CR Blocks Received: 361

-> Captured Segments account for 35.5% of Total

| Owner | Name | Object | Subobject | CR | |
|--------|--------|----------------------|-----------|------|----------|
| | | | | Name | Type |
| | | | | | |
| <hr/> | | | | | |
| SYS | SYSTEM | JOB\$ | | | TABLE 22 |
| SYS | SYSAUX | SMON_SCN_TIME | | | TABLE 21 |
| SYSMAN | SYSAUX | MGMT_SYSTEM_PERFORMA | | | TABLE 12 |
| SYSMAN | SYSAUX | MGMT_SYSTEM_PERF_LOG | | | INDEX 12 |
| SYSMAN | SYSAUX | MGMT_TASK_QTABLE | | | TABLE 12 |
| <hr/> | | | | | |
| <hr/> | | | | | |

Segments by Current Blocks Received DB/Inst:

AULTDB/aultdb1 Snaps: 91-92

-> Total Current Blocks Received: 95,445

-> Captured Segments account for 99.9% of Total

| Owner | Name | Object | Subobject | Current | |
|------------|------------|---------------|--------------|---------|--------------|
| | | | | Name | Type |
| | | | | | |
| <hr/> | | | | | |
| TPCH | INDEXES | H_ORDERS_IDX1 | | | INDEX 65,524 |
| TPCH | DATA | H_ORDER | | | TABLE 24,149 |
| TPCH | DATA | H_SUPPLIER | | | TABLE 2,232 |
| SYS | SYSTEM | TAB\$ | | | TABLE 996 |
| ** UNAVAIL | ** UNAVAIL | ** UNAVA ** | AVAILABLE ** | | |
| UNDEF | 776 | .81 | | | |
| <hr/> | | | | | |
| <hr/> | | | | | |

The segments sections of the AWR report allow you to pinpoint which segments and tablespaces are responsible for the various types of reads, waits, and other database-related statistics that are related to segments.

- **Segments by Logical Reads:** If you have issues with high logical reads, review these objects for possible partitioning, SQL issues, or index issues.
- **Segments by Physical Reads:** If you have physical read issues, review these objects for index issues or possible partitioning.
- **Segments by Row Lock Waits:** If you have high transaction lock (TX) type enqueue, look here for locking issues with objects. These will also show up in the RAC sections if the problem is present in a cluster database.
- **Segments by ITL Waits:** If you have large numbers of block waits, look here for

the causes.

- **Segments by Buffer Busy Waits:** These segments probably have too large a block size if in RAC. Otherwise, look at insufficient memory allocations—these segments are experiencing hot blocks.
- **Segments by Global Cache Buffer Busy:** Look to these segments for hot block issues in RAC. Usually, these are caused by too large a block size in RAC. If the segments are indexes, it may be helpful to convert them to reverse key indexes (however, this approach inhibits index scans).
- **Segments by CR Blocks Received:** This statistic shows consistent read issues; look at hot blocking and block size, and consider using PCTFREE or smaller blocks to reduce rows per block.
- **Segments by Current Blocks Received:** This statistic shows current blocks that are being transferred and indicates high numbers of transactions. Using small block sizes can help, as can using reverse key indexers.

Tip

Usually, if the object is contributing less than 10 percent of the total for a category, it is not a significant source of problems. However, when looking at partitioned objects, all of the partitions must be added together to get an objects contribution. Unfortunately, the report doesn't do this for you.

Library Cache Activity Sections

In Oracle 7, the tuning of the various dictionary caches was automated. The statistics in the next sections dealing with library cache activity show you, for example, whether you need to use more caching for sequences or if you should look at using automated segment management and other internal issues that are indicated through the library cache statistics. [Listing 10.13](#) shows an excerpt from our AWR report.

Listing 10.13 Library Cache Statistics

[Click here to view code image](#)

```
Dictionary Cache Stats          DB/Inst:  
AULTDB/aultdb1  Snaps: 91-92  
-> "Pct Misses"  should be very low (< 2% in most cases)  
-> "Final Usage" is the number of cache entries being used
```

| Cache | Get Requests | Pct Miss | Scan Reqs | Pct Miss | Mod Reqs |
|-------------------|--------------|----------|-----------|----------|----------|
| dc_awr_control | 63 | 3.2 | 0 | N/A | 0 |
| dc_database_links | 2 | 0.0 | 0 | N/A | 0 |

| | | | | | |
|----------------------|--------|------|------|-----|-----|
| dc_files | 8 | 0.0 | 0 | N/A | 0 |
| dc_global_oids | 2,826 | 0.2 | 0 | N/A | 0 |
| dc_histogram_data | 1,151 | 11.6 | 0 | N/A | 0 |
| dc_histogram_defs | 3,213 | 5.6 | 0 | N/A | 0 |
| dc_object_grants | 484 | 0.0 | 0 | N/A | 0 |
| dc_objects | 7,172 | 1.0 | 0 | N/A | 17 |
| dc_profiles | 64 | 0.0 | 0 | N/A | 0 |
| dc_rollback_segments | 850 | 0.0 | 0 | N/A | 0 |
| dc_segments | 1,020 | 5.9 | 0 | N/A | 4 |
| dc_sequences | 13 | 30.8 | 0 | N/A | 13 |
| dc_tablespaces | 9,757 | 0.0 | 0 | N/A | 0 |
| dc_users | 13,294 | 0.0 | 0 | N/A | 0 |
| global database | | | | | |
| name | 4,485 | 0.0 | 0 | N/A | 1 |
| outstanding_alerts | | 52 | 69.2 | 0 | N/A |
| | | | | | 2 |

Dictionary Cache Stats (RAC) DB/Inst: AULTDB/aultdb1 Snaps: 91-92

| Cache | GES | | GES | | GES | |
|--------------------|----------|-----------|----------|--|-----|--|
| | Requests | Conflicts | Releases | | | |
| dc_awr_control | 2 | 2 | 0 | | | |
| dc_global_oids | 5 | 0 | 0 | | | |
| dc_histogram_defs | 181 | 0 | 0 | | | |
| dc_objects | 71 | 0 | 0 | | | |
| dc_segments | 68 | 5 | 0 | | | |
| dc_sequences | 26 | 5 | 0 | | | |
| dc_tablespaces | 1 | 0 | 0 | | | |
| dc_users | 5 | 0 | 0 | | | |
| outstanding_alerts | 100 | 36 | 0 | | | |

--

Library Cache Activity DB/Inst:
AULTDB/aultdb1 Snaps: 91-92
-> "Pct Misses" should be very low

| Namespace | Get | | Pin | | Pct | | I1 |
|-----------|----------|-----|----------|-----|------|---------|----|
| | Requests | Pct | Requests | Pct | Miss | Reloads | |
| BODY | 1,514 | 0.0 | 1,858 | 0.2 | | | 4 |
| CLUSTER | 44 | 0.0 | 16 | 0.0 | | | 0 |
| INDEX | 2 | 0.0 | 2 | 0.0 | | | 0 |
| JAVA | | | | | | | |
| DATA | 2 | 0.0 | 0 | N/A | 0 | | 0 |
| SQL | | | | | | | |

| | | | | | |
|-----------------|--------|-----|--------|-----|-----|
| AREA | 2,246 | 1.5 | 17,091 | 2.5 | 121 |
| TABLE/PROCEDURE | 12,745 | 0.1 | 16,155 | 1.4 | 166 |
| TRIGGER | 376 | 0.0 | 423 | 0.0 | 0 |

Library Cache Activity (RAC) DB/Inst: AULTDB/aultdb1 Snaps: 91-92

| Invali- | GES Lock Requests | GES Pin Requests | GES Pin Releases | GES Inval Requests | GES |
|-----------------|-------------------|------------------|------------------|--------------------|-----|
| Namespace | | | | | (|
| CLUSTER | 16 | 16 | 0 | 0 | |
| INDEX | 2 | 2 | 0 | 0 | |
| TABLE/PROCEDURE | 4,553 | 15,492 | 0 | 0 | |

One thing to remember when dealing with statistics is that you need to have a statistically relevant number of measurements before the statistics are valid. If you have only two occurrences, then it is rather hard to draw valid conclusions. On the other hand, if you have 1,000 or 10,000 occurrences, then you can draw more valid conclusions. If we eliminate the “invalid” statistics from the previous sections, we are left with the valid statistics in [Listing 10.14](#).

Listing 10.14 Valid Library Cache Statistics

[Click here to view code image](#)

Dictionary Cache Stats DB/Inst: AULTDB/aultdb1 Snaps: 91-92

-> "Pct Misses" should be very low (< 2% in most cases)
-> "Final Usage" is the number of cache entries being used

| Cache Usage | Get Requests | Pct Miss | Scan Reqs | Pct Miss | Mod Reqs |
|-------------------|--------------|----------|-----------|----------|----------|
| dc_global_oids | 2,826 | 0.2 | 0 | N/A | 0 |
| dc_histogram_data | 1,151 | 11.6 | 0 | N/A | 0 |
| dc_histogram_defs | 3,213 | 5.6 | 0 | N/A | 0 |
| dc_objects | 7,172 | 1.0 | 0 | N/A | 17 |
| dc_segments | 1,020 | 5.9 | 0 | N/A | 4 |
| dc tablespaces | 9,757 | 0.0 | 0 | N/A | 0 |
| dc_users | 13,294 | 0.0 | 0 | N/A | 0 |
| global database | | | | | |

| | | | | | | |
|------|-------|-----|---|-----|---|---|
| name | 4,485 | 0.0 | 0 | N/A | 0 | 1 |
|------|-------|-----|---|-----|---|---|

Dictionary Cache Stats (RAC) DB/Inst:
 AULTDB/aultdb1 Snaps: 91-92

| Cache | GES Requests | GES Conflicts | GES Releases |
|-------|--------------|---------------|--------------|
|-------|--------------|---------------|--------------|

Library Cache Activity DB/Inst:
 AULTDB/aultdb1 Snaps: 91-92
 -> "Pct Misses" should be very low

| Namespace | Get Requests | Pct Miss | Pin Requests | Pct Miss | Reloads |
|-----------|--------------|----------|--------------|----------|---------|
|-----------|--------------|----------|--------------|----------|---------|

| | | | | | |
|-----------------|--------|-----|--------|-----|-----|
| BODY | 1,514 | 0.0 | 1,858 | 0.2 | 4 |
| SQL | | | | | |
| AREA | 2,246 | 1.5 | 17,091 | 2.5 | 121 |
| TABLE/PROCEDURE | 12,745 | 0.1 | 16,155 | 1.4 | 166 |

Library Cache Activity (RAC) DB/Inst:
 AULTDB/aultdb1 Snaps: 91-92

| Invali-
Namespace | GES Lock Requests | GES Pin Requests | GES Pin Releases | GES Inval Requests | GES |
|----------------------|-------------------|------------------|------------------|--------------------|-----|
|----------------------|-------------------|------------------|------------------|--------------------|-----|

| | | | | | |
|-----------------|-------|--------|---|---|--|
| TABLE/PROCEDURE | 4,553 | 15,492 | 0 | 0 | |
|-----------------|-------|--------|---|---|--|

So, what should we be looking for in the valid library cache statistics? Generally, the miss percentages should be low. However, if the database is just starting up, then the miss percentages will be high until the cache fills. In cases where many temporary segments, undo segments, and other objects are accessed and released, we may see high miss percentages, but they aren't of major concern. If we see the miss percentages for a majority of areas, then we probably have a shared pool sizing issue and we may need to (if we are using automated memory management) manually establish a "floor" value by setting the `shared_pool_size` parameter. If we can't set the shared pool parameter, we would need to increase `sga_target` to closer to `sga_max_size` in Oracle 10g

or `memory_target` closer to `memory_max_target` in later releases. If you find that these pairs of parameters are set to the same value, you will need to raise the maximum size parameters first. In most cases, you should have a difference of several percentage points in the values for the target and maximum size parameters, usually anywhere from 200 MB to 1 GB depending on the amount of memory your system can give to Oracle.

Based on the value of the target parameters, Oracle will automatically allocate space to the various automatically controlled areas, such as the shared pool, default DB block cache, large pool, Java pool, and others. Within the shared pool, the library and dictionary caches will be set; if the dictionary cache areas are too small, then misses will result. If the SQL or Procedural Language/Structured Query Language (PL/SQL) areas are too small, then you will see reloads and invalidations. This can also drive *latch free* events.

Dynamic Memory Components Sections

If you are using the automatic memory management in Oracle 10g or Oracle 11g, then the AWR report will contain sections showing what components were resized or modified during the AWR report period. An excerpt of this section is shown in [Listing 10.15](#).

Listing 10.15 Dynamic Memory Sections

[Click here to view code image](#)

| Memory Dynamic Components | | DB/Inst: AULTDB/aultdb1 | | Snaps: 91-92 | | | | |
|---------------------------|-----------|-------------------------|------|--------------|------|------|------|-------|
| Last Op | Component | Begin | Snap | Current | Min | Max | Oper | |
| Typ/Mod | | Size | (Mb) | Size | (Mb) | Size | (Mb) | Count |
| <hr/> | | | | | | | | |
| ASM Buffer Cach STA/ | | .00 | | .00 | | .00 | | .00 |
| DEFAULT 16K buf STA/ | | .00 | | .00 | | .00 | | .00 |
| DEFAULT 2K buff STA/ | | .00 | | .00 | | .00 | | .00 |
| DEFAULT 32K buf STA/ | | .00 | | .00 | | .00 | | .00 |
| DEFAULT 4K buff STA/ | | .00 | | .00 | | .00 | | .00 |

| | | | | | |
|-----------------|----------|----------|----------|----------|---|
| DEFAULT 8K buff | .00 | .00 | .00 | .00 | 0 |
| STA/ | | | | | |
| DEFAULT buffer | 1,312.00 | 1,312.00 | 1,296.00 | 1,328.00 | 2 |
| GRO/DEF | | | | | |
| KEEP buffer cac | .00 | .00 | .00 | .00 | 0 |
| STA/ | | | | | |
| PGA Target | 512.00 | 512.00 | 512.00 | 512.00 | 0 |
| STA/ | | | | | |
| RECYCLE buffer | .00 | .00 | .00 | .00 | 0 |
| STA/ | | | | | |
| SGA Target | 1,584.00 | 1,584.00 | 1,584.00 | 1,584.00 | 0 |
| STA/ | | | | | |
| Shared IO Pool | .00 | .00 | .00 | .00 | 0 |
| STA/ | | | | | |
| java pool | 16.00 | 16.00 | 16.00 | 16.00 | 0 |
| STA/ | | | | | |
| large pool | 16.00 | 16.00 | 16.00 | 16.00 | 0 |
| STA/ | | | | | |
| shared pool | 224.00 | 224.00 | 208.00 | 240.00 | 2 |
| SHR/DEF | | | | | |
| streams pool | .00 | .00 | .00 | .00 | 0 |
| STA/ | | | | | |

Memory Resize Operations Summary DB/Inst:

AULTDB/aultdb1 Snaps: 91-92

- > Resizes, Grows, Shrinks - Operations captured by AWR
if there are operations on the same component for the same operation\_type, target\_size, and with the same start\_time
only one operation is captured
- > ordered by Component

| Component | Min Size (Mb) | Max Size (Mb) | Avg Size (Mb) | Re-Sizes | Grows |
|-----------|---------------|---------------|---------------|----------|-------|
| Shrink | | | | | |
| ----- | ----- | ----- | ----- | ----- | ----- |
| ----- | ----- | ----- | ----- | ----- | ----- |
| DEFAULT | | | | | |
| buffer | 1,296.00 | 1,312.00 | 1,304.00 | 2 | 1 |
| shared | | | | | |
| pool | 224.00 | 240.00 | 232.00 | 2 | 1 |
| ----- | ----- | ----- | ----- | ----- | ----- |
| ----- | ----- | ----- | ----- | ----- | ----- |

Memory Resize Ops DB/Inst:

AULTDB/aultdb1 Snaps: 91-92

-> Oper Types/Modes:

INITializing, GROW, SHRink, STAtic/IMMEDIATE, DEFerred
Delta : change in size of the component

Target Delta: displayed only if final size <> target\_size
 -> Status: COMplete/CANcelled/INActive/PENding/ERRor
 -> ordered by start\_time desc, component

| Start
(M) | Ela
(s) | Component
(M) Sta | Oper
Typ/Mod | Init
Size | Target | Fi: |
|----------------|------------|----------------------|-----------------|--------------|--------|-----|
| 08/04 12:39:06 | 0 | | | | | |
| bufcache | GRO/DEF | 1,296 | 16 | N/A | 1,312 | COM |
| 08/04 12:39:06 | 0 | | | | | |
| shared | SHR/DEF | 240 | -16 | N/A | 224 | COM |
| 08/04 12:02:59 | 2 | | | | | |
| bufcache | SHR/DEF | 1,312 | -16 | N/A | 1,296 | COM |
| 08/04 12:02:59 | 2 | | | | | |
| shared | GRO/DEF | 224 | 16 | N/A | 240 | COM |
| <hr/> | | | | | | |
| <hr/> | | | | | | |

The dynamic memory sections are used as a guide to setting the dynamic memory parameters `sga_target` and `sga_max_size` in Oracle10g and `memory_target` and `memory_max_target` in Oracle11g.

If you see large numbers of resize operations in the statistics in this section, pay attention to which component is doing which action. You can use this information to determine if the manual parameters (`shared_pool_size`, `db_cache_size`, `java_pool_size`, `large_pool_size`, and `streams_pool_size`) should be set to a floor value. For example, if we were to see in the Memory Resize Operations Summary section that the shared pool has a number of grow operations while the default cache is shrinking, we might want to set the parameters for their sizes to the values indicated in the Memory Resize Ops section. If we see that all of the actions are deferred and not completed, that usually indicates that the target and maximum size parameters are set to the same value. Many defers or shrinks followed by grows can also indicate that we need to increase the maximum size parameters, since it shows that we are robbing Peter to pay Paul within our memory structures.

If we have the proper resources, and the target and maximum size parameters are set correctly, we should see marginal grows in the various pools and few shrinks. However, this assumes that we have all the memory we need to satisfy all memory needs. In systems that are memory poor, there may be no way to prevent this give and take of memory and in fact that is by design.

If you have issues with cache buffer-related or library-related latches and mutexes, this section of the report will show you possible reasons.

Process Memory Sections

The process global area is usually controlled by the `pga_aggregate_target`

parameter since Oracle 9.0. The individual process is usually constrained to 5 percent of the setting of `pga_aggregate_target` up to the value of the undocumented parameter, `_pga_max_size`.

In Oracle 9.0, the size of `_pga_max_size` was constrained to 200 MB; in Oracle 10g, depending on the release, this parameter was upped to 500 MB; and in Oracle 11g, it seems to be (at least on Windows Vista and Red Hat 32-bit Linux in Oracle 11.0.1.0.6) back to 200 MB. On 64-bit implementations, this may be different.

[Listing 10.16](#) shows the example AWR report sections for the process memory statistics.

Listing 10.16 Process Memory Sections

[Click here to view code image](#)

| Process Memory Summary | | | | DB/Inst: | | | | | | | |
|---|------------|-----------|----------------|--------------------|----------------|---------------------|----------|--|--|--|--|
| AULTDB/aultdb1 Snaps: 91-92 | | | | | | | | | | | |
| <ul style="list-style-type: none"> -> B: Begin snap E: End snap -> All rows below contain absolute values (i.e. not diffed over the interval) -> Max Alloc is Maximum PGA Allocation size at snapshot time -> Hist Max Alloc is the Historical Max Allocation for still-connected processes -> ordered by Begin/End snapshot, Alloc (MB) desc | | | | | | | | | | | |
| Category | Alloc (MB) | Used (MB) | Avg Alloc (MB) | Std Dev Alloc (MB) | Max Alloc (MB) | Hist Max Alloc (MB) | Num Proc | | | | |
| <hr/> | | | | | | | | | | | |
| <hr/> | | | | | | | | | | | |
| B | | | | | | | | | | | |
| Other | 153.1 | N/A | 3.5 | 4.9 | 24 | 24 | 44 | | | | |
| Freeable | 13.3 | .0 | .9 | .9 | 4 | N/A | 14 | | | | |
| JAVA | 1.6 | 1.6 | .8 | 1.2 | 2 | 2 | 2 | | | | |
| SQL | 1.2 | .5 | .1 | .1 | 0 | 3 | 20 | | | | |
| PL/SQL | .2 | .1 | .0 | .0 | 0 | 0 | 42 | | | | |
| E | | | | | | | | | | | |
| Other | 175.3 | N/A | 3.5 | 4.6 | 24 | 24 | 50 | | | | |
| Freeable | 101.5 | .0 | 5.3 | 10.1 | 28 | N/A | 19 | | | | |
| SQL | 23.2 | 22.5 | .9 | 1.7 | 5 | 166 | 26 | | | | |
| JAVA | 1.6 | 1.6 | .8 | 1.2 | 2 | 2 | 2 | | | | |
| PL/SQL | .2 | .1 | .0 | .0 | 0 | 0 | 48 | | | | |
| <hr/> | | | | | | | | | | | |
| <hr/> | | | | | | | | | | | |
| SGA Memory Summary | | | | DB/Inst: | | | | | | | |
| AULTDB/aultdb1 Snaps: 91-92 | | | | | | | | | | | |

| (Bytes) | Begin Size (Bytes) | End Size (Bytes) |
|------------------------|--------------------|------------------|
| SGA regions different) | | (if |
| ----- | ----- | ----- |
| --- | | |
| Database Buffers | 1,375,731,712 | |
| Fixed Size | 1,300,968 | |
| Redo Buffers | 10,858,496 | |
| Variable Size | 671,090,200 | |
| ----- | ----- | ----- |
| sum | 2,058,981,376 | |
| ----- | ----- | ----- |

SGA breakdown difference DB/Inst:
 AULTDB/aultdb1 Snaps: 91-92
 -> ordered by Pool, Name
 -> N/A value for Begin MB or End MB indicates the size of that Pool/Name was insignificant, or zero in that snapshot

| Pool Name | Begin MB | End |
|--------------------------|----------|------------|
| MB % Diff | | |
| ----- | ----- | ----- |
| java free | | |
| memory | 7.7 | 7.7 0.00 |
| java joxlod exec | | |
| hp | 8.1 | 8.1 0.00 |
| java joxs | | |
| heap | .3 | .3 0.00 |
| large ASM map operations | | |
| hashta | .2 | .2 0.00 |
| large PX msg | | |
| pool | .3 | .4 41.67 |
| large free | | |
| memory | 15.5 | 15.4 -0.79 |
| shared ASH | | |
| buffers | 4.0 | 4.0 0.00 |
| shared | | |
| CCursor | | |
| shared Heap0: | | |
| KGL | 3.4 | 4.0 15.27 |
| shared KCB Table Scan | | |
| Buffer | 4.0 | 4.1 1.74 |
| shared KGL | | |
| buckets | 4.0 | 4.0 0.00 |
| shared KGL | | |
| handle | 3.0 | 3.0 0.00 |
| shared KGLS | | |

| | | | |
|--------------------------|---------|---------|--------|
| heap | 3.8 | 4.6 | 20.93 |
| shared KQR L | | | |
| PO | N/A | 2.3 | N/A |
| shared KQR M | | | |
| PO | 2.8 | 3.0 | 5.87 |
| shared KSFD SGA I/O | | | |
| b | 4.0 | 4.0 | 0.00 |
| shared | | | |
| PCursor | 2.6 | 2.6 | 1.61 |
| shared PL/SQL | | | |
| DIANA | 11.4 | 11.4 | 0.00 |
| shared PL/SQL | | | |
| MPCODE | 12.8 | 13.0 | 1.53 |
| shared | | | |
| db_block_hash_buckets | 5.9 | 5.9 | 0.00 |
| shared dbwriter coalesce | | | |
| buffer | 4.0 | 4.0 | 0.00 |
| shared free | | | |
| memory | 41.4 | 32.7 | -20.92 |
| shared gcs | | | |
| resources | 20.2 | 20.2 | 0.00 |
| shared gcs | | | |
| shadows | 15.7 | 15.7 | 0.00 |
| shared ges big msg | | | |
| buffers | 4.1 | 4.1 | 0.00 |
| shared | | | |
| quesblFilter_seg | N/A | 4.0 | N/A |
| shared row | | | |
| cache | 3.6 | 3.6 | 0.00 |
| shared sql | | | |
| area | 14.1 | 16.6 | 17.03 |
| shared type object | | | |
| de | 2.7 | 2.7 | 0.02 |
| buffer_cache | 1,312.0 | 1,312.0 | |
| fixed_sga | 1.2 | 1.2 | |
| log_buffer | 10.4 | 10.4 | |
| ----- | | | |
| ----- | | | |

Process Memory Summary

The Process Memory Summary section of the AWR report expands on the previous PGA sections that showed the use of the PGA for sort type operations. In these sections, we see the before and after snapshots of how the PGA areas changed from the beginning to the end AWR snapshot. By comparing the begin to the end snapshot for the different PGA sections, we can determine what might need to be tuned in our PGA environment. One thing to remember here is that these numbers are totals and are not averaged over the total number of processes except where it is explicitly stated that it is an average, sum,

or maximum value.

The B (begin) and E (end) sections allow us to compare how the usage changed over the course of the snapshot period. For example, in [Listing 10.16](#), the freeable memory jumped from 13.3 MB to 101.5 MB, or an average of 1.8 MB per process. Generally, freeable memory is memory that was used for sort or hash activities. We also saw increases in other memory and SQL memory usages.

SGA Memory Summary

The SGA Memory Summary section just shows gross changes to the main areas of the SGA. The results would be the same if you did a `show SGA` command in SQL\*Plus before and after your activity and noted and changes to the numbers shown. Generally, it is not of much use in tuning.

SGA Breakdown Difference

The SGA Breakdown Difference section shows the memory components whose allocations may have changed, how much they changed by percentage, and in what direction. This section allows us to analyze the component by component changes and also helps pinpoint possible memory leaks.

Tip

One set of the statistics shown in this section that bears watching is the free statistics. If the free statistics don't change, then memory may be overallocated to those components.

Streams Component Sections

If you are utilizing Oracle Streams, then the AWR will populate the various streams sections of the AWR report. Essentially, if you are seeing spills to disks from the streams pool or excessive time delays in the queues (usually, you will see both if you see one or the other), then you need to look at increasing the streams pool size (it should be handled automatically) or increase the number of queue processes for either the capture queues or apply queues. If you use resource limits, the resource limit section will show how the resource limits have been applied during this period.

[Listing 10.17](#) shows the streams component areas. Unfortunately, since our example database is not using streams, these sections are not populated with actual data from the example database but with data derived from other actual reports.

Listing 10.17 Streams AWR Report Sections

[Click here to view code image](#)

Streams CPU/IO Usage

DB/Inst:

AULTDB/aultdb1 Snaps: 91-92
 -> Streams processes ordered by CPU usage
 -> CPU and I/O Time in micro seconds

| Session Type | CPU Time | User I/O Time | Sys I/O |
|---------------|-----------|---------------|---------|
| Time | | | |
| <hr/> | | | |
| <hr/> | | | |
| STREAMS | | | |
| Capture | 2,128,000 | 0 | 0 |
| STREAMS Apply | | | |
| Reader | 609,945 | 2,050 | 1,341 |
| Logminer | | | |
| Builder | 185,835 | 0 | 0 |
| Logminer | | | |
| Preparer | 175,559 | 0 | 0 |
| QMON | | | |
| Slaves | 132,888 | 0 | 0 |
| Logminer | | | |
| Reader | 97,009 | 0 | 887,687 |
| STREAMS Apply | | | |
| Server | 35,580 | 0 | 0 |
| STREAMS Apply | | | |
| Coordinator | 747 | 0 | 0 |
| QMON | | | |
| Coordinator | 454 | 0 | 0 |
| Propagation | | | |
| Sender | 0 | 0 | 0 |
| <hr/> | | | |
| <hr/> | | | |

Streams Capture DB/Inst:
 AULTDB/aultdb1 Snaps: 91-92
 -> Lag Change should be small or negative (in seconds)

| Name | Captured | | Pct
Per | Pct
Per | Pct
Lag | Pct
RuleEval | Pct
Enqueue |
|------------|----------|----------|------------|------------|------------|-----------------|----------------|
| | Enqueued | RedoWait | | | | | |
| | Second | Pause | | | | | |
| Capture | | | | | | | |
| Name | Second | Second | Change | Time | Time | Time | Time |
| <hr/> | | | | | | | |
| <hr/> | | | | | | | |
| STREAM_CAP | 65 | 39 | 93 | 0 | 23 | 0 | 0 |
| <hr/> | | | | | | | |
| <hr/> | | | | | | | |

Streams Apply /Inst:
 AULTDB/aultdb1 Snaps: 91-92
 -> Pct DB is the percentage of all DB transactions that this apply

handled

- > WDEP is the wait for dependency
- > WCMT is the wait for commit
- > RBK is rollbacks
- > MPS is messages per second
- > TPM is time per message in milli-seconds
- > Lag Change should be small or negative (in seconds)

| Pct | Applied | Pct | Pct | Pct | Applied | Dequeue | Apply | Lag |
|------------|---------|-----|-----|------|---------|---------|-------|-----|
| Apply Name | | TPS | DB | WDEP | WCMT | | | |
| RBK | MPS | TPM | | TPM | Change | | | |
| <hr/> | | | | | | | | |
| STREAM_APP | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| <hr/> | | | | | | | | |
| <hr/> | | | | | | | | |

Buffered Queues /Inst: AULTDB/aultdb1 Snaps: 91-92
-> The Spill Rate should be very close to zero
-> The Diff in Percentage Spilled should be very close to zero or negative

| Queue Schema and Name | Incoming per second | Outgoing per second | Spilled per second | D: Pct Spilled |
|-----------------------|---------------------|---------------------|--------------------|----------------|
| <hr/> | | | | |
| STRMADMIN.REP_CAPTURE | 39 | 39 | 0 | 0 |
| STRMADMIN.REP_DEST_QU | 0 | 0 | 0 | 0 |
| <hr/> | | | | |
| <hr/> | | | | |

Buffered Subscribers /Inst: AULTDB/aultdb1 Snaps: 91-92
-> All Subscribers should have a zero spill rate

| Subscriber Name | Incoming per second | Outgoing per second | Spilled per second | |
|----------------------|---------------------|---------------------|--------------------|--|
| <hr/> | | | | |
| STREAM_APP | 793 | 793 | 6 | |
| STREAM_APP | 0 | 0 | 0 | |
| PROXY: "STRMADMIN"." | 391 | 391 | 0 | |
| PROXY: CATLIBRAR.ASU | 0 | 0 | 0 | |
| PROXY: CATLIBRAR.ASU | -793 | -793 | -6 | |
| <hr/> | | | | |
| <hr/> | | | | |

Rule Set /Inst: AULTDB/aultdb1 Snaps:

91-92

-> Rule Sets ordered by Evaluations

| Ruleset | Evals | Evals | Execs | SQL Time | CPU Time | Elap: |
|--------------------------|-------|-------|-------|----------|----------|-------|
| Name | | | | | | |
| STRMADMIN.RULESET\$ _ 10 | 3,174 | 0 | 3,174 | 1,565 | 2, | |
| SYS.ALERT_QUE_R | 2 | 0 | 0 | 0 | 0 | |
| STRMADMIN.RULESET\$ _ 6 | 0 | 0 | 0 | 0 | 0 | |
| STRMADMIN.RULESET\$ _ 3 | 0 | 0 | 0 | 0 | 0 | |
| STRMADMIN.RULESET\$ _ 8 | 0 | 0 | 0 | 0 | 0 | |

The statistics dealing with timing and spillage to disk are probably the most important of the streams sections. If any single queue is showing excessive times, then adding additional queues may help with that problem; however, the addition of queues may require changes on the receiving or sending instances in Advanced Queue (AQ) setup as well. In addition, there may be spillage numbers shown. If you are experiencing spillage, you have insufficient memory allocated to the streams pool, and you should increase the size of that memory component. The Streams Pool Advisory section will help you in deciding the needed changes to the streams pool memory allocation.

Resource Limits Statistics

The resource limits statistics show what resources have had to be modified by the Oracle kernel during the period covered by the report. If you see resources being continuously adjusted upward, it may indicate a need to revisit the settings of initialization parameters. [Listing 10.18](#) shows the Resource Limits section.

Listing 10.18 Resource Limits Statistics Section

[Click here to view code image](#)

| Resource Limit Stats | Inst: | | | | |
|---|------------|-------|---------------------|---------------------|---------|
| AULTDB/aultdb1 Snaps: 91-92 | | | | | |
| -> only rows with Current or Maximum Utilization > 80% of Limit are shown | | | | | |
| -> ordered by resource name | | | | | |
| Resource Name | Allocation | Limit | Current Utilization | Maximum Utilization | Initial |
| sessions | | | 51 | 65 | 150 |

Initialization Parameter Changes

Unless either some automated process or the DBA has made changes during the test period for which the AWR report has been run, there should be no changes to initialization parameter settings; therefore, if you see changes and you did not initiate them, you need to investigate the causes of the changes. [Listing 10.19](#) shows the initialization parameter section. Only parameters that have values different than the default or that have changed will be shown in the initialization parameter section.

Listing 10.19 Initialization Parameter Section

[Click here to view code image](#)

```
init.ora Parameters          DB/Inst: AULTDB/aultdb1  Snaps:  
91-92  
-> if IP/Public/Source at End snap is different a '*' is displayed  
  
End  
value  
Parameter Name      Begin value      (if  
different)  
-----  
-----  
audit_file_dest      /home/oracle/app/product/oracle/a  
audit_trail          DB  
cluster_database     TRUE  
cluster_database_instances 2  
compatible           11.1.0.0.0  
control_files        +DATA2/aultdb/controlfile/current  
db_block_size        8192  
db_create_file_dest +DATA2  
db_domain              
db_name              aultdb  
db_recovery_file_dest +DATA2  
db_recovery_file_dest_size 2147483648  
diagnostic_dest      /home/oracle/app/product/oracle  
dispatchers          (PROTOCOL=TCP) (SERVICE=aultdbXDB  
instance_number      1  
memory_target        2197815296  
open_cursors         300  
processes            150  
remote_listener      LISTENERS_AULTDB  
remote_login_passwordfile EXCLUSIVE  
spfile               +DATA/aultdb/spfileaultdb.ora  
star_transformation_enabled TRUE
```

```
thread          1
undo_tablespace UNDOTBS1
```

It is always a good idea to verify the initialization parameter settings are correct when analyzing AWR reports. A misplaced power of 10 or a botched kilobyte to megabyte to gigabyte calculation can make a world of difference in the efficiency of an SGA.

Tip

You should also look for odd undocumented parameter settings and track down the source of the suggested values shown if you are not familiar with the settings yourself. Many times, a DBA will carry settings from one version of Oracle to another during upgrades without considering whether or not special settings are still appropriate.

If we weren't using RAC, this would be the last section of the report; however, the use of RAC adds some additional global enqueue statistics sections to the AWR report, which are covered in the rest of this chapter.

Global Enqueue and Other RAC Sections

With RAC come many new statistics and areas to monitor. The AWR report has been expanded to provide detailed RAC sections to help you troubleshoot and diagnose RAC issues. [Listing 10.20](#) shows the additional RAC sections pruned of statistics that generally won't be of use.

Listing 10.20 Global Enqueue and RAC Statistics

[Click here to view code image](#)

| Global Enqueue Statistics | | DB/Inst: | AULTDB/aultdb1 | Snaps: |
|---------------------------|-------|----------|----------------|--------|
| 91-92 | | | | |
| Statistic | Trans | Total | per Second | per |
| acks for commit | | | | |
| broadcast(actual) | 216 | 0.1 | 0.2 | |
| acks for commit | | | | |
| broadcast(logical) | 242 | 0.1 | 0.2 | |
| broadcast msgs on | | | | |
| commit(actual) | 700 | 0.2 | 0.6 | |
| broadcast msgs on | | | | |
| commit(logical) | 701 | 0.2 | 0.6 | |

| | | | | |
|-------------------------------|---------|-------|-------|--|
| broadcast msgs on | | | | |
| commit(wasted) | 53 | 0.0 | 0.0 | |
| gcs assume no | | | | |
| cvt | 45,685 | 12.6 | 38.9 | |
| gcs blocked | | | | |
| converts | 205 | 0.1 | 0.2 | |
| gcs blocked cr | | | | |
| converts | 238 | 0.1 | 0.2 | |
| gcs compatible cr basts | | | | |
| (local) | 93,290 | 25.8 | 79.4 | |
| gcs dbwr flush pi | | | | |
| msgs | 36 | 0.0 | 0.0 | |
| gcs dbwr write request | | | | |
| msgs | 141 | 0.0 | 0.1 | |
| gcs immediate (compatible) | | | | |
| conver | 87 | 0.0 | 0.1 | |
| gcs immediate (null) | | | | |
| converts | 324 | 0.1 | 0.3 | |
| gcs immediate cr (compatible) | | | | |
| con | 11,512 | 3.2 | 9.8 | |
| gcs immediate cr (null) | | | | |
| converts | 213,759 | 59.2 | 181.9 | |
| gcs indirect | | | | |
| ast | 42,995 | 11.9 | 36.6 | |
| gcs indirect fg | | | | |
| ast | 42,995 | 11.9 | 36.6 | |
| gcs msgs process | | | | |
| time(ms) | 15,443 | 4.3 | 13.1 | |
| gcs msgs | | | | |
| received | 549,546 | 152.1 | 467.7 | |
| gcs new served by | | | | |
| master | 102 | 0.0 | 0.1 | |
| gcs pings | | | | |
| refused | 8 | 0.0 | 0.0 | |
| gcs retry convert | | | | |
| request | 16,809 | 4.7 | 14.3 | |
| gcs side channel msgs | | | | |
| actual | 5,444 | 1.5 | 4.6 | |
| gcs side channel msgs | | | | |
| logical | 146,320 | 40.5 | 124.5 | |
| ges msgs process | | | | |
| time(ms) | 344 | 0.1 | 0.3 | |
| ges msgs | | | | |
| received | 15,248 | 4.2 | 13.0 | |
| global posts queue | | | | |
| time | 318 | 0.1 | 0.3 | |
| global posts | | | | |
| queued | 59 | 0.0 | 0.1 | |
| global posts | | | | |
| requested | 63 | 0.0 | 0.1 | |

| | | | | |
|------------------------------|---------|-------|-------|--|
| global posts | | | | |
| sent | 59 | 0.0 | 0.1 | |
| implicit batch messages | | | | |
| received | 26,591 | 7.4 | 22.6 | |
| implicit batch messages | | | | |
| sent | 28,441 | 7.9 | 24.2 | |
| messages flow | | | | |
| controlled | 2,047 | 0.6 | 1.7 | |
| messages queue sent | | | | |
| actual | 68,909 | 19.1 | 58.6 | |
| messages queue sent | | | | |
| logical | 142,750 | 39.5 | 121.5 | |
| messages received | | | | |
| actual | 236,086 | 65.3 | 200.9 | |
| messages received | | | | |
| logical | 564,794 | 156.3 | 480.7 | |
| messages sent | | | | |
| directly | 134,160 | 37.1 | 114.2 | |
| messages sent | | | | |
| indirectly | 245,700 | 68.0 | 209.1 | |
| messages sent not implicit | | | | |
| batche | 40,468 | 11.2 | 34.4 | |
| messages sent | | | | |
| pbatched | 306,413 | 84.8 | 260.8 | |
| msgs causing lmd to send | | | | |
| msgs | 5,845 | 1.6 | 5.0 | |
| msgs causing lms(s) to send | | | | |
| msgs | 19,392 | 5.4 | 16.5 | |
| msgs received queue time | | | | |
| (ms) | 41,000 | 11.3 | 34.9 | |
| msgs received | | | | |
| queued | 564,808 | 156.3 | 480.7 | |
| msgs sent queue time | | | | |
| (ms) | 146,886 | 40.7 | 125.0 | |
| msgs sent queue time on ksxp | | | | |
| (ms) | 322,477 | 89.2 | 274.4 | |
| msgs sent | | | | |
| queued | 139,264 | 38.5 | 118.5 | |
| msgs sent queued on | | | | |
| ksxp | 243,401 | 67.4 | 207.1 | |
| process batch messages | | | | |
| received | 50,530 | 14.0 | 43.0 | |
| process batch messages | | | | |
| sent | 50,185 | 13.9 | 42.7 | |

| Statistic | Total |
|------------------------|-------|
| CR Block Requests | 486 |
| CURRENT Block Requests | 36 |
| Data Block Requests | 486 |
| Undo Block Requests | 0 |
| TX Block Requests | 7 |
| Current Results | 522 |
| Fairness Down Converts | 78 |
| Fairness Clears | 10 |
| Flushes | 4 |

| Statistic | Total | % <1ms | % <10ms | % <100ms | % <1s | % <10s |
|-----------|--------|--------|---------|----------|-------|--------|
| Pins | 93,484 | 99.95 | 0.00 | 0.05 | 0.00 | 0.00 |
| Flushes | 1 | 0.00 | 0.00 | 100.00 | 0.00 | 0.00 |
| Writes | 43 | 0.00 | 4.65 | 74.42 | 18.60 | 2.31 |

| Global Cache Transfer Stats | DB/Inst: |
|---|----------|
| AULTDB/aultdb1 Snaps: 91-92 | |
| -> Immediate(Immed)-Block Transfer NOT impacted by Remote Processing Delays | |
| -> Busy (Busy) - Block Transfer impacted by Remote Contention | |
| -> Congested (Congst) - Block Transfer impacted by Remote System Load | |
| -> ordered by CR + Current Blocks Received desc | |

| Inst Block | Blocks | CR | | Current | |
|------------|----------|-------|------|---------|----------|
| | | % | % | % | % |
| No Class | Received | Immed | Busy | Congst | Received |
| Congst | | | | | Immed |
| | | | | | Busy |
| 2 data | | | | | |

| | | | | | | | | |
|--------|-----|-------|-------|----|--------|-------|-------|----|
| block | 150 | 98.0 | 2.0 | .0 | 95,402 | 99.8 | .0 | .2 |
| 2 undo | | | | | | | | |
| header | 200 | 100.0 | .0 | .0 | 3 | 100.0 | .0 | .0 |
| 2 | | | | | | | | |
| Others | | 10 | 100.0 | .0 | .0 | 24 | 100.0 | .0 |

Global Cache Transfer Times (ms) DB/Inst:
AULTDB/aultdb1 Snaps: 91-92
-> Avg Time - average time of all blocks (Immed,Busy,Congst) in ms
-> Immed, Busy, Congst - Average times in ms
-> ordered by CR + Current Blocks Received desc

| | CR Avg Time (ms) | | | | Current Avg Time | | | |
|----------------|------------------|-------|--------|-------|------------------|-------|--------|-------|
| (ms) | Immed | Busy | Congst | All | Immed | Busy | Congst | |
| -----
----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |
| Inst Block | No Class | All | | | | | | |
| 2 data | blo | 2.5 | 1.9 | 27.6 | N/A | 1.8 | 1.7 | N/A |
| 2 undo | hea | 1.4 | 1.4 | N/A | N/A | 1.6 | 1.6 | N/A |
| 2 | others | 1.4 | 1.4 | N/A | N/A | 1.4 | 1.4 | N/A |
| 2 undo | blo | N/A | N/A | N/A | N/A | N/A | N/A | N/A |

Global Cache Transfer (Immediate) DB/Inst:
AULTDB/aultdb1 Snaps: 91-92
-> Immediate(Immed)-Block Transfer NOT impacted by Remote Processing Delays
-> % of Blocks Received requiring 2 or 3 hops
-> ordered by CR + Current Blocks Received desc

| | CR | | | | Current | | | |
|----------------|--------|------------|-------|-------|----------|-------|-------|-------|
| Src Block | Blocks | Immed Blks | % | % | Immed | | | |
| Blks | % | % | | | | | | |
| -----
----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |
| Inst | | | | | | | | |
| Class | Lost | Received | 2hop | 3hop | Received | 2hop | 3hop | |

| | | | | | | |
|------------|--------|-------|-----|-----|---|-----|
| 2 data blo | 0 | 147 | | | | |
| 100.0 0.0 | 95,204 | 100.0 | 0.0 | | | |
| 2 undo hea | 0 | 200 | | | | |
| 100.0 0.0 | 3 | 100.0 | 0.0 | | | |
| 2 others | 0 | 10 | | | | |
| 100.0 0.0 | 24 | 100.0 | 0.0 | | | |
| 2 undo | | | | | | |
| blo | 0 | 0 | N/A | N/A | 0 | N/A |
| | | | | | | |

Global Cache Times (Immediate) DB/Inst:
AULTDB/aultdb1 Snaps: 91-92
-> Blocks Lost, 2-hop and 3-hop Average times in (ms)
-> ordered by CR + Current Blocks Received desc

| Time (ms) | CR Avg Time (ms) | | | | Current Avg | | |
|-----------|------------------|------|-------|------|-------------|-------|------|
| | Src Block | Lost | Immed | 2hop | 3hop | Immed | 2hop |
| blo | 1.9 | 1.9 | N/A | | 1.7 | 1.7 | N/A |
| 2 undo | 1.4 | 1.4 | N/A | | 1.6 | 1.6 | N/A |
| hea | | | | | | | |
| 2 | | | | | | | |
| others | 1.4 | 1.4 | N/A | | 1.4 | 1.4 | N/A |
| 2 undo | N/A | N/A | N/A | | N/A | N/A | N/A |
| blo | | | | | | | |

Interconnect Ping Latency Stats DB/Inst:
AULTDB/aultdb1 Snaps: 91-92
-> Ping latency of the roundtrip of a message from this instance to
-> target in
-> The target instance is identified by an instance number.
-> Average and standard deviation of ping latency is given in
milliseconds
-> for message sizes of 500 bytes and 8K.
-> Note that latency of a message from the instance to itself is
used as
-> control, since message latency can include wait for CPU

| Target 500B | Pin Avg Latency | Stddev | 8K Ping Avg | Stddev |
|-------------|-----------------|--------|-------------|--------|
| Latency | | | | |

| Instance
msg | Count
8K msg | 500B msg | 500B msg | Count | 8K |
|-----------------|-----------------|----------|----------|-------|------|
| <hr/> | | | | | |
| 1 | 360 | .64 | 1.04 | 360 | .63 |
| 2 | 360 | 1.39 | 2.43 | 360 | 2.16 |
| <hr/> | | | | | |

Interconnect Throughput by Client DB/Inst:

AULTDB/aultdb1 Snaps: 91-92

-> Throughput of interconnect usage by major consumers.

-> All throughput numbers are megabytes per second

| Used By | Send
Mbytes/sec | Receive
Mbytes/sec |
|----------------|--------------------|-----------------------|
| <hr/> | | |
| Global Cache | .20 | .21 |
| Parallel Query | .35 | .39 |
| DB Locks | .03 | .03 |
| DB Streams | .00 | .00 |
| Other | .00 | .00 |
| <hr/> | | |

Interconnect Device Statistics DB/Inst:

AULTDB/aultdb1 Snaps: 91-92

-> Throughput and errors of interconnect devices (at OS level).

-> All throughput numbers are megabytes per second

| Device Name | IP Address | Public Source | | |
|-----------------------|-------------------|--------------------|------------------------------|----------------------------|
| <hr/> | | | | |
| Send
Mbytes/sec | Send
Errors | Send
Dropped | Send
Buffer
Overrun | Send
Carrier
Lost |
| <hr/> | | | | |
| Receive
Mbytes/sec | Receive
Errors | Receive
Dropped | Receive
Buffer
Overrun | Receive
Frame
Errors |
| <hr/> | | | | |
| eth0 | 11.1.1.1 | NO | Oracle Cluster Repository | |
| .77 | 0 | 0 | 0 | 0 |
| .82 | 1 | 0 | 0 | 1 |
| <hr/> | | | | |

Global Enqueue Statistics

Most of the global enqueue statistics really aren't useful unless you are trying to track

down a particular type of event, such as null to s conversion rates or other statistics that deal with how locks are converted throughout the system. An entire book could be written on RAC tuning, so it is a bit beyond this chapter to delve deeply into all the ins and outs of the various Global Enqueue Service statistics.

Global CR Served Statistics

Global consistent read statistics show you how often blocks are being requested and sent due to consistent read activity. A high number indicates hot blocks and the possibility that your block size is too large or you need to adjust rows per block.

Global Current Served Statistics

Current blocks being transferred shows that multiple transactions are requiring the same blocks to make changes. This usually indicates a very busy system and may show a need to reduce block size or rows per block.

Usually, you want to be sure that the transfer times histograms show that a majority of transfers occurred at less time than your average read/write latency to the disks.

Tip

If your transfer times are consistently greater than your disk latency times, then your interconnect is becoming a bottleneck, and you need to look at tuning it or replacing it with a higher-bandwidth, lower-latency technology such as InfiniBand.

Global Cache Transfer Statistics

Watching the number of blocks transferred is rather like watching waits; without the time component, the number is useless. If 10,000 blocks are transferred and it takes 1 second, you are happy. If 100 blocks are transferred and it takes 10 seconds, you are worried (or should be). So, while it's good to know the number of blocks and types of blocks, later sections that give you the time breakdowns for transfers are more important.

Global Cache Transfer Times

The global cache transfer times are the more critical of the statistics. Notice in this section that all of the statistics, save one, are less than 2 ms. For data blocks (busy), however, we see that it is taking 26.7 ms to transfer a data block. In this case, it is probably a transaction-related locking issue, since none of the other timings are bad.

Global Cache Transfer (Immediate)

Immediate mode transfers are detailed in this section for both consistent reads and current blocks. Again, this shows a breakdown of numbers of blocks by data type, which can help isolate which segments to examine for issues. However, always pair this

information with the next section before panicking over the number of blocks of a particular type shown to be whisking their way across the interconnect. Always look at timing data along with block counts.

Global Cache Times (Immediate)

From looking at the times for the various types of consistent reads and current clocks, we see that in no case were the transfer times excessive, so even if we did transfer 95,204 current blocks via two hops, we did it at 1.7 ms per block, almost three times faster than the best disk latency (5 ms). When transfer times become excessive, look at tuning the interconnect, reducing the number of blocks transferred by tuning SQL, or replacing the interconnect with a better technology.

Interconnect Ping Latency Statistics

By doing periodic pings, the system can get a rough idea of what transfer speeds should be. By looking at these ping statistics, you can see for different message sizes how well the interconnect is transferring data. If the times are excessive for various ping sizes, then look to tuning the underlying TPC/UDP buffers. If that doesn't help, replace the interconnect.

Interconnect Throughput by Client

You should examine the throughput by client when you are seeing excessive latency in the interconnect statistics. By examining the amount of data being transferred across the interconnect, you can determine if the interconnect is being overloaded and thus causing the long latencies.

Tip

Usually an overloaded interconnect will start dropping blocks, and if you see the dropped blocks parameters for the various instances above zero, then it may indicate the interconnect is seeing a lot of stress.

Interconnect Device Statistics

In this section, each of the NICs or interconnect devices is shown along with statistics on throughput and error counts. If you see nonzero dropped statistics (one or two isn't a worry but dozens are an alarm), then either you have a buffer issue or the interconnect is overloaded.

Summary

This chapter took us to the end of the AWR report. The AWR report is the front-line tool for tuning your Oracle database. We have looked at the various statistics dealing with tablespace I/O, memory, and Oracle internals such as latches, locks, and enqueue. We have reviewed how to utilize statistics to calculate ratios and other useful summations

for use in troubleshooting and tuning.

You should remember that the AWR is a licensed tool requiring the diagnostics and tuning pack licenses; however, Statspack, which contains almost all of the same information, is free and is still available even in Oracle Database 11g. So if you don't want to pay the extra license fees for the diagnostic and tuning packs, which are required for using AWR, then use Statspack.

11. Troubleshooting Problematic Scenarios in RAC

Oracle Real Application Clusters (RAC) is the parallelized cluster version of the Oracle database server family and plays a very significant role in everyday Oracle database server operations, particularly Exadata. RAC was born in Oracle version 6 in the form of Oracle Parallel Server (OPS), which was later rebranded as Oracle Real Application Clusters in version *9i*.

In addition to having the ability to scale horizontally without any downtime, RAC gives the Oracle database server the ability to load balance Oracle database workloads among multiple database nodes. The load-balancing feature of RAC is achieved by fusing the memory areas of the various cluster nodes by a mechanism known as *Global Cache Fusion*. This mechanism uses various underlying RAC daemons and processes, notably the Lock Management Server (LMS) process, which formulates and constitutes the backbone of Global Cache Fusion.

Oracle RAC is implemented by setting up and configuring Oracle Grid Infrastructure (Oracle Clusterware and Automatic Storage Management [ASM]). RAC runs on shared storage on a private cluster network interconnect to present a single logical source of truth to the database end user. Shared storage is a basic requirement for Oracle RAC cluster nodes. This shared storage is provided by implementing and utilizing ASM.

The private cluster interconnect is the networking component that powers Global Cache Fusion, the technology that gives RAC its high availability (HA) and load-balancing capabilities. The private cluster interconnect requires a very high-speed redundant low-latency fabric.

RAC provides horizontal scalability, high availability, and load balancing at the Oracle database compute tier. It is these features and characteristics that RAC provides that are fundamental to building elastic and self-serviceable Oracle Database Clouds.

This chapter focuses on how to identify, troubleshoot, and fix problematic RAC issues. Best practices, tips, and techniques on how to configure a well-oiled RAC ecosystem are also covered.

Troubleshooting and Tuning RAC

This section gives high-level useful tips and pointers on how to troubleshoot and performance-tune RAC.

Start with ORAchk

ORAchk is the new, comprehensive health-check tool for the RAC/Exadata stack and can be utilized for scanning prevalent issues within RAC. Prior to OraChk, EXAchk was the recommended health-check tool for RAC/Exadata—it can be downloaded from MOS

Note 1070954.1, “Oracle Exadata Database Machine Exachk or HealthCheck.” Exachk can be a great starting point in giving overall directions and pointers toward potential problems, issues, and pain points. ORAchk is the latest tool incorporating the functionality of EXAchk and RACcheck tools. ORAchk can be downloaded from MOS Note 1268927.2, “ORAchk—Health checks for the Oracle stack.”

Employ the TFA Collector Utility

Released with Oracle Database 11.2.0.4, the Transparent File Analyzer (TFA) Collector utility is the new, all-encompassing utility that simplifies collection of RAC diagnostic information. TFA greatly simplifies diagnostic data collection, upload, and troubleshooting analysis by Oracle support. TFA can be downloaded from MOS Note 1513912.1, “TFA collector—Tool for enhanced diagnostic gathering.”

Utilize the Automatic Diagnostic Repository

The Automatic Diagnostic Repository (ADR) is a comprehensive and consolidated platform for storing diagnostic RAC logging data in a file-based repository. The ADRCI command-line utility can be utilized to inspect and view diagnostic information across all RAC instances, including incident-related data, alert and trace files and dumps, and so on. ADRCI can also be used to package and compress all related diagnostic data into .ZIP files for transmission to Oracle Support for further analysis. However, ADR is subsumed by TFA, as it collects ADR data as well—if you use TFA, then it already collects ADR data.

Check the Alert and Trace Log Files

Each RAC database instance has its own alert log file. This file is located in the directory specified by the DIAGNOSTIC\_DEST init.ora parameter. This file can point to further trace files that are linked to specific problems and incidents. Parsing these alert and trace log files is absolutely critical to identifying and troubleshooting problems with RAC. If TFA is used to collect and analyze diagnostic data, then the RAC database alert and trace data is already collected within it.

Switch to the \$GRID\_HOME/log/host\_name folder to access log files of the Clusterware. The first folder that should be accessed in case of failure or for the purpose of investigation is \$GRID\_HOME/log/host\_name/crsd, which contains Cluster Ready Services Daemon (CRSD) log files. Next, cssd and ohasd folders might also be used to check the alerts of Cluster Synchronization Services and Oracle High Availability Services Daemon processes.

Employ the Three A’s

Employ the three A’s of performance tuning and troubleshooting for performance-related and other general issues:

- The Automatic Workload Repository (AWR) report enables you to conduct

forensics on the Oracle database server family. To get more detailed information about AWR, refer to [Chapters 9](#) and [10](#), “[Database Forensics and Tuning Using AWR Analysis](#),” parts I and II.

- The Automatic Database Diagnostic Monitor report crunches the data in the AWR report and spits out findings and recommendations in human-readable form.
- The Active Session History (ASH) report enables you to perform tracing and forensics on individual database sessions.

Check the Private Cluster Interconnect

The private cluster interconnect is the central network backbone across which all internode cache fusion occurs. The private cluster interconnect should be inspected for potential issues. In addition to leveraging Oracle logs, system and network administrators should be engaged for this endeavor.

Enable Tracing and Inspect the Trace Logs

Tracing can be enabled on Java-based RAC utilities, such as Database Configuration Assistant (DBCA), Database Upgrade Assistant (DBUA), Cluster Verification Utility, and Server Control Utility (SRVCTL). These logs can be found in their respective directories (for example, \$ORACLE\_HOME/cfgtoollogs/dbca) and should be inspected in case of troubleshooting problems and issues.

Utilize the Cluster Health Monitor

Cluster Health Monitor (CHM) is a very useful tool for troubleshooting problematic scenarios in RAC. CHM metrics, which are housed in the Grid Infrastructure Management Repository, can be analyzed for troubleshooting on a RAC cluster. CHM data can be garnered by running the diagcollection.pl script, which is present in the \$GRIDHOME/bin directory. However, diagcollection.pl is obsolete in favor of TFA.

Miscellaneous Tools and Utilities

In addition to the tools previously mentioned, following are some useful tools and utilities that can be employed for diagnosing and troubleshooting RAC issues:

- RAC Configuration Audit Tool is used to audit different configuration settings in CRS, Grid Infrastructure, RAC, and ASM environments (RACcheck) (DOC ID 1268927.1).
- The ProcWatcher script is used to monitor and examine Oracle Database and Clusterware processes (DOC ID 459694.1).
- OS Watcher Black Box (OSWBB) is used to capture performance metrics of the operating system (DOC ID 1531223.1).
- The ORATOP utility is used for near real-time monitoring of databases, RAC, and single instances (DOC ID 1500864.1).

Useful My Oracle Support Resources

Following are some very useful My Oracle Support (MOS) resources for troubleshooting RAC environments:

- **810394.1:** “RAC and Oracle Clusterware best practices and starter kit (platform independent)”
- **1053147.1:** “11gR2 Clusterware and Grid Home—What you need to know”
- **169706.1:** “Oracle Database (RDBMS) on Unix AIX, HP-UX, Linux, Mac OS X, Solaris, Tru64 Unix operating systems installation and configuration requirements quick reference (8.0.5 to 11.2)”
- **1366558.1:** “Top 11gR2 Grid Infrastructure upgrade issues”

A Well-Oiled RAC Ecosystem

This section covers best practices, tips, and techniques to ensure that your RAC environment is operating as a well-oiled machine, is mitigating problematic scenarios, and is scalable and sustainable, giving you the best bang for your buck. These tips are briefly touched on in this section—some of them are discussed in detail in subsequent chapters of this book.

Maximum Availability Architecture

Implement the Oracle Maximum Availability Architecture (MAA) set of guidelines and best practices to eliminate single points of failure (SPOFs) and achieve high availability.

Following are the key components of Oracle MAA:

- Oracle RAC
- Oracle ASM
- No SPOF at every component and layer of the hardware and software stack

Following are highly recommended components of Oracle MAA for RAC that should be implemented to achieve high availability in the Oracle stack:

- Oracle Active Data Guard should be implemented. It provides replication of Oracle RAC databases infrastructures while allowing you to open your standby (replicated) databases in read-only mode for reporting, analytics, and so on.
- Set up and configure normal or high redundancy for ASM disk groups (double-mirroring or triple-mirroring respectively).
- Implement EtherChannel (bonded) network interfaces or equivalent technologies for RAC interconnect interfaces: public, backup network channels, and so on.
- Set up, configure, and test an effective backup strategy utilizing Oracle Recovery Manager (RMAN).
- Set up and configure database block-checking parameters (DB\_BLOCK\_CHECKING, DB\_BLOCK\_CHECKSUM, and

- Set `DB_LOST_WRITE_PROTECT` to protect against data block corruption.
- For all production databases within RAC, turn on `ARCHIVELOG` and `FORCE LOGGING` modes to make recovery operations stable and effective.
- Use Oracle Flashback options to protect data against logical corruptions. These options also provide the ability to rewind the RAC databases very easily to a specific point in time.
- Set a minimum or required level of `UNDO_RETENTION` to protect undo operations as well as Flashback operations.
- Set up and configure the applications connecting to RAC to seamlessly failover to the surviving nodes of the cluster as well as the Data Guard standby database in case of failure.

Optimal and Efficient Databases in RAC

The following options, tips, and techniques provide great aid in ensuring efficient and optimal consolidation of Oracle databases in RAC. Categorizing and consolidating Oracle databases and schemas into similar groups is a common theme in achieving this goal.

Current Versions

Stay current with the latest releases and versions of the RAC software stack. Potential bugs pose an existential threat to RAC security and stability and are routinely fixed in new patches, releases, and versions.

ORAck and Exachk

Run Exachk (the recommended health-check tool from Oracle) periodically to check the health of your overall Exadata/RAC ecosystem. Exachk can be downloaded from MOS Note 1070954.1, “Oracle Exadata Database Machine Exachk or HealthCheck.” ORAck (MOS Note 1268927.2), the new health-check tool for the Oracle Stack, contains the capabilities of Exachk and should be periodically run to proactively and reactively diagnose issues with the system as a whole. ORAck is discussed later in this chapter.

Effective Resource Management

Configuring and leveraging resource management options in Oracle at the database, compute, and storage tiers to efficiently manage the various resources in RAC is crucial to having a balanced and well-tuned system. The two main technologies in this realm to set up are Oracle Database Resource Manager (DBRM) and IO Resource Manager (IORM—applicable for Exadata).

Resource management technologies put caps on system resources on groups of database users as well as efficiently manage system resources. Using resource management technologies prevents Oracle programs, queries, and users from running away with your system resources, leading to systemwide failures.

CPU and Memory Management

Having adequate amounts of CPU and memory, along with implementing resource limits on them, is a sound and time-tested strategy in setting up RAC-clustered databases. This ensures application workloads run in a stable manner while preventing fatal problems such as node evictions and split-brain scenarios in RAC.

Instance caging (CPU) is a mechanism that is implemented by the `CPU_COUNT` parameter. It limits the amount of CPU available to an Oracle RAC database. On the other hand, incorrect or insufficient allocation of `CPU_COUNT` and instance caging can lead to `RESMGR:CPU Quantum` wait events. These wait events should be minimized in order to ensure adequate amounts of CPU horsepower to the Oracle RAC database instances.

Automatic Memory Management (AMM) should be set up and configured in a manner that ensures adequate levels of memory for the RAC database instances while putting resource limits on the amount of memory consumption in a multi-tenant RAC environment.

HugePages are not compatible with AMM in Linux. This incompatibility must be taken into account when using AMM. Automatic Shared Memory Management (ASMM) should be used instead in conjunction with HugePages. The following MOS notes cover this subject in greater detail:

- **749851.1:** “HugePages and Oracle Database 11g Automatic Memory Management (AMM) on Linux”
- **1134002.1:** “ASMM versus AMM and LINUX x86-64 HugePages Support”

Antivirus Software

Antivirus software should not be operational on RAC nodes. It can have a fatal impact on the internal workings of Oracle RAC processes, eventually resulting in node evictions.

Third-Party Monitoring Tools and Utilities

Third-party monitoring tools and utilities should be leveraged/implemented with caution because they have the potential and tendency to interfere with the internal workings of RAC.

Optimal Tuning of RAC Parameters

Care must be taken when configuring the initialization parameter of the RAC database. For example, overallocation of the `LMS_PROCESSES` parameter can lead to excessive CPU consumption leading to fatal consequences from a cluster stability standpoint. Refer to MOS Note 1392248.1, “Auto Adjustment of LMS Process Priority in Oracle RAC with version 11.2.0.3 and Later.”

Partitioning and Parallelization

Following the divide-and-conquer approach for achieving maximized performance in

RAC cluster environments, the golden themes of partitioning and parallelization go hand in hand to achieve this goal.

Setting up automatic degree of parallelism (`PARALLEL_DEGREE_POLICY=AUTO`) is an excellent approach to enabling parallelism with minimal required effort while enabling automatic parallelism (multithreading) for statement processing in Oracle.

This is a powerful new feature with Oracle 11.2 and later. However, care must be exercised in the correct and optimal setup and configuration of the `PARALLEL_` parameters to ensure that automatic degree of parallelism does not have a negative effect on the overall performance of the system. For example, setting `PARALLEL_MAX_SERVERS` to a very high value can potentially starve system memory resulting in nodes being unresponsive and eventually being evicted from the cluster. This is used in conjunction with the Resource Manager so that sessions don't overconsume the parallel slaves.

Partitioning of large database objects is absolutely crucial to achieving the necessary required performance in RAC environments. Partitioning and parallelism often complement each other by reducing overall CPU and memory consumption, achieving optimal performance, and ensuring system stability.

Troubleshooting RAC with OEM 12c

Oracle Enterprise Manager (OEM) 12c is the end-to-end management, monitoring, administration, and support tool of choice for RAC environments. With intuitive, easy-to-use functionality, OEM 12c is the standard and best platform for monitoring, managing, and troubleshooting problematic scenarios in Oracle RAC environments.

The following tools, utilities, reports, and pages can prove helpful in identifying, tracking, and fixing problem scenarios for RAC in OEM 12c:

- Performance home page
- Cluster cache coherency
- Real-time ADDM report
- AWR, Automatic Database Diagnostic Monitor (ADDM), and Active Session History (ASH) reports
- Top activity
- SQL monitoring
- Compare period ADDM
- Compare period (AWR) reports
- Blocking sessions
- SQL tuning, access, and performance advisors
- Emergency monitoring

Utilities and Commands for Troubleshooting

This section browses through *some* of the common utilities and commands used to inspect, view, and administer RAC and its underlying components. These commands are broken down into three main categories, each related to its corresponding command-line tool or utility:

- **Oracle Clusterware Control Utility (CRSCTL):** CRSCTL gives you the ability to perform administrative tasks on Oracle RAC Clusterware and its underlying componentry—for example, start, stop, check, enable, disable, health-check, and so on. CRSCTL commands can be run from any node in the cluster. Its use should be restricted to Oracle RAC Clusterware critical operations only—most of the resource management can be performed using SRVCTL.
- **Server Control Utility (SRVCTL):** SRVCTL is used to manage the resources and entities managed by Oracle Clusterware—for example, start, stop, add, remove, enable, and disable the services, databases and instances, SCAN listeners, NodeApps, ASM, and so on.
- **Miscellaneous:** This category includes other command-line utilities and tools to query, inspect, and manage RAC, Clusterware, and so on.

The `crsctl check` commands in [Listing 11.1](#) check the status of the Clusterware componentry installed on RAC.

Listing 11.1 The `crsctl check` Commands

[Click here to view code image](#)

```
$ crsctl check has
CRS-4638: Oracle High Availability Services is online

$ crsctl check crs
CRS-4638: Oracle High Availability Services is online
CRS-4537: Cluster Ready Services is online
CRS-4529: Cluster Synchronization Services is online
CRS-4533: Event Manager is online
```

The `crsctl query` commands in [Listing 11.2](#) show the information related to Oracle RAC Clusterware.

Listing 11.2 The `crsctl query` Commands

[Click here to view code image](#)

```
$ crsctl query crs activeversion
Oracle Clusterware active version on the cluster is [11.2.0.4.0]

$ crsctl query crs releaseversion
Oracle High Availability Services release version on the local node
is [11.2.0.4.0]
```

```
$ crsctl query crs softwareversion  
Oracle Clusterware version on node [oe01db01] is [11.2.0.4.0]
```

```
$ crsctl query crs softwareversion -all
Oracle Clusterware version on node [oe01db01] is [11.2.0.4.0]
Oracle Clusterware version on node [oe01db02] is [11.2.0.4.0]
Oracle Clusterware version on node [oe01db03] is [11.2.0.4.0]
Oracle Clusterware version on node [oe01db04] is [11.2.0.4.0]
Oracle Clusterware version on node [oe01db05] is [11.2.0.4.0]
Oracle Clusterware version on node [oe01db06] is [11.2.0.4.0]
Oracle Clusterware version on node [oe01db07] is [11.2.0.4.0]
Oracle Clusterware version on node [oe01db08] is [11.2.0.4.0]
```

```
$ crsctl query css votedisk
```

```
## STATE File Universal Id           File
Name                               Disk group
-----
-----  
1. ONLINE 7cb8478916a84f10bf7dbb336ca68601
(o/192.168.10.5/DBFS_DG_CD_02_oe01cel01) [DBFS_DG]
2. ONLINE 8eb8c6e255534f84bfb1da0194b845bb
(o/192.168.10.6/DBFS_DG_CD_02_oe01cel02) [DBFS_DG]
3. ONLINE 7f378e868c094fb0bfc38af465fc64f4
(o/192.168.10.7/DBFS_DG_CD_02_oe01cel03) [DBFS_DG]
Located 3 voting disk(s).
```

```
$ crsctl query crs administrator  
CRS Administrator List: *
```

The `crsctl status` and `crsctl get` commands in Listing 11.3 show the status information related to the Oracle RAC Clusterware componentry.

Listing 11.3 The crsctl status and crsctl get Commands

[Click here to view code image](#)

```
$ crsctl status serverpool -p
NAME=Free
IMPORTANCE=0
MIN_SIZE=0
MAX_SIZE=-1
SERVER_NAMES=
PARENT_POOLS=
EXCLUSIVE_POOLS=
ACL=owner:oracle:rwx,pgrp:oinstall:rwx,other::r-x

NAME=Generic
```

```

IMPORTANCE=0
MIN_SIZE=0
MAX_SIZE=-1
SERVER_NAMES=oe01db01 oe01db02
PARENT_POOLS=
EXCLUSIVE_POOLS=
ACL=owner:oracle:r-x,pgrp:oinstall:r-x,other::r-x

NAME=ora.dbm
IMPORTANCE=1
MIN_SIZE=0
MAX_SIZE=-1
SERVER_NAMES=oe01db01 oe01db02
PARENT_POOLS=Generic
EXCLUSIVE_POOLS=
ACL=owner:oracle:rwx,pgrp:oinstall:rwx,other::r--

NAME=ora.sri
IMPORTANCE=1
MIN_SIZE=0
MAX_SIZE=-1
SERVER_NAMES=oe01db01 oe01db02
PARENT_POOLS=Generic
EXCLUSIVE_POOLS=
ACL=owner:oracle:rwx,pgrp:oinstall:rwx,other::r--

$ crsctl status serverpool ora.dbm -p
NAME=ora.dbm
IMPORTANCE=1
MIN_SIZE=0
MAX_SIZE=-1
SERVER_NAMES=oe01db01 oe01db02
PARENT_POOLS=Generic
EXCLUSIVE_POOLS=
ACL=owner:oracle:rwx,pgrp:oinstall:rwx,other::r--

$ crsctl get cluster mode status
Cluster is running in "standard" mode

$ crsctl get node role config
Node 'exa1db01' configured role is 'hub'

```

The `srvctl status` commands in [Listing 11.4](#) show the status information related to the resources that Oracle RAC Clusterware manages.

Listing 11.4 The `srvctl status` Commands

[Click here to view code image](#)

```

$ srvctl status server -n oe01db01,oe01db02,oe01db03,oe01db04
Server name: oe01db01
Server state: ONLINE
Server name: oe01db02
Server state: ONLINE
Server name: oe01db03
Server state: ONLINE
Server name: oe01db04
Server state: ONLINE

$ srvctl status database -d dbm
Instance dbm1 is running on node oe01db01
Instance dbm2 is running on node oe01db02

$ srvctl status instance -d dbm -i dbm1
Instance dbm1 is running on node oe01db01

$ srvctl status nodeapps
VIP oe01db01-vip is enabled
VIP oe01db01-vip is running on node: oe01db01
VIP oe01db02-vip is enabled
VIP oe01db02-vip is running on node: oe01db02
VIP oe01db03-vip is enabled
VIP oe01db03-vip is running on node: oe01db03
VIP oe01db04-vip is enabled
VIP oe01db04-vip is running on node: oe01db04
VIP oe01db05-vip is enabled
Network is enabled
Network is running on node: oe01db01
Network is running on node: oe01db02
Network is running on node: oe01db03
Network is running on node: oe01db04
GSD is disabled
GSD is not running on node: oe01db01
GSD is not running on node: oe01db02
GSD is not running on node: oe01db03
GSD is not running on node: oe01db04
ONS is enabled
ONS daemon is running on node: oe01db01
ONS daemon is running on node: oe01db02
ONS daemon is running on node: oe01db03
ONS daemon is running on node: oe01db04

$ srvctl status nodeapps -n oe01db01
VIP oe01db01-vip is enabled
VIP oe01db01-vip is running on node: oe01db01
Network is enabled
Network is running on node: oe01db01
GSD is disabled
GSD is not running on node: oe01db01

```

```
ONS is enabled
ONS daemon is running on node: oe01db01

$ srvctl status asm
ASM is running on oe01db01, oe01db02, oe01db03, oe01db04

$ srvctl status diskgroup -g DATA1
Disk Group DATA1 is running on oe01db01, oe01db02,
oe01db03, oe01db04

$ srvctl status listener
Listener LISTENER is enabled
Listener LISTENER is running on node(s): oe01db01, oe01db02,
oe01db03, oe01db04

$ srvctl status listener -n oe01db01
Listener LISTENER is enabled on node(s): oe01db01
Listener LISTENER is running on node(s): oe01db01

$ srvctl status scan
SCAN VIP scan1 is enabled
SCAN VIP scan1 is running on node oe01db02
SCAN VIP scan2 is enabled
SCAN VIP scan2 is running on node oe01db01
SCAN VIP scan3 is enabled
SCAN VIP scan3 is running on node oe01db03

$ srvctl status scan -i 1
SCAN VIP scan1 is enabled
SCAN VIP scan1 is running on node oe01db02

$ srvctl status scan_listener
SCAN Listener LISTENER_SCAN1 is enabled
SCAN listener LISTENER_SCAN1 is running on node oe01db02
SCAN Listener LISTENER_SCAN2 is enabled
SCAN listener LISTENER_SCAN2 is running on node oe01db01
SCAN Listener LISTENER_SCAN3 is enabled
SCAN listener LISTENER_SCAN3 is running on node oe01db03

$ srvctl status scan_listener -i 1
SCAN Listener LISTENER_SCAN1 is enabled
SCAN listener LISTENER_SCAN1 is running on node oe01db02

$ srvctl status vip -n oe01db01
VIP oe0101-vip is enabled
VIP oe0101-vip is running on node: oe01db01

$ srvctl status vip -i oe0101-vip-vip
PRKO-2167 : VIP oe0101-vip-vip does not exist.
```

The `srvctl config` commands in [Listing 11.5](#) show the configuration information related to the resources that Oracle RAC Clusterware manages.

Listing 11.5 The `srvctl config` Commands

[Click here to view code image](#)

```
$ srvctl config database -d dbm
Database unique name: dbm
Database name: dbm
Oracle home: /u01/app/oracle/product/11.2.0.4/dbhome_1
Oracle user: oracle
Spfile: +DATA_OE01/dbm/spfiledbm.ora
Domain: at-rockside.lab
Start options: open
Stop options: immediate
Database role: PRIMARY
Management policy: AUTOMATIC
Server pools: dbm
Database instances: dbm1, dbm2
Disk Groups: DATA_OE01, RECO_OE01
Mount point paths:
Services:
Type: RAC
Database is administrator managed

$ srvctl config nodeapps -n oe01db01
-n <node_name> option has been deprecated.
Network exists: 1/174.17.40.0/255.255.255.0/bondeth0, type static
VIP exists: /oe0101-
vip/174.17.40.4/174.17.40.0/255.255.255.0/bondeth0,
           hosting node oe01db01
GSD exists
ONS exists: Local port 6100, remote port 6200, EM port 2016

$ srvctl config listener -l LISTENER -a
Name: LISTENER
Network: 1, Owner: oracle
Home: <CRS home>
      /u01/app/11.2.0.4/grid on node(s) oe01db02, oe01db01
End points: TCP:1521
```

The commands in [Listing 11.6](#) show miscellaneous information related to RAC componentry.

Listing 11.6 Miscellaneous RAC Componentry Information

[Click here to view code image](#)

```

$ ocrconfig -showbackup

oe01db01      2014/09/08
14:41:23      /u01/app/11.2.0.3/grid/cdata/oe01-cluster/backup00.ocr

oe01db01      2014/09/08
10:41:23      /u01/app/11.2.0.3/grid/cdata/oe01-cluster/backup01.ocr

oe01db01      2014/09/08
06:41:23      /u01/app/11.2.0.3/grid/cdata/oe01-cluster/backup02.ocr

oe01db01      2014/09/07
02:41:21      /u01/app/11.2.0.3/grid/cdata/oe01-cluster/day.ocr

oe01db01      2014/08/28
14:41:06      /u01/app/11.2.0.4/grid/cdata/oe01-cluster/week.ocr

oe01db01      2013/02/26
17:20:00      /u01/app/11.2.0.4/grid/cdata/oe01-
cluster/backup_20130226_172000.ocr

$ olsnodes -s
oe01db01 Active
oe01db02 Active

$ olsnodes -n
oe01db01 1
oe01db02 2

$ ocrcheck
Status of Oracle Cluster Registry is as follows :
Version          :      3
Total space (kbytes)   :    262120
Used space (kbytes)    :     3036
Available space (kbytes) :    259084
ID                : 1278623030
Device/File Name    : +DBFS_DG
                      Device/File integrity check succeeded
Device/File not configured
                      Device/File not configured
                      Device/File not configured
                      Device/File not configured
Cluster registry integrity check succeeded
ocrcheck          Logical corruption check succeeded


```

```

$ ocrcheck -local
Status of Oracle Local Registry is as follows :
Version          :      3
Total space (kbytes)   :    262120
Used space (kbytes)    :     2684

```

```
Available space (kbytes) : 259436
ID : 957270436
Device/File Name :
/u01/app/11.2.0.4/grid/cdata/oe01db01.olr
```

```
Device/File integrity check succeeded
Local registry integrity check succeeded
Logical corruption check succeeded
```

Summary

RAC is *the* software component that provides load balancing and high availability at the Oracle database tier by eliminating a single point of failure at the database compute tier. This chapter started with a quick primer about RAC followed by the various methodologies, utilities, and tools for troubleshooting and fixing RAC issues. Best practices for configuration, setup, and efficient operations along with common, everyday commands and performance-tuning tips were also covered to aid in troubleshooting and tuning RAC.

12. Leveraging SQL Advisors to Analyze and Fix SQL Problems

SQL issues formulate the major chunk of an Oracle professional's troubleshooting activity. Hardly a day goes by in the professional life of an Oracle DBA that SQL issues don't present a hardy challenge. With the sophistication of technology and the rapid advancement of subsequent versions loaded with new features, an advanced-level Swiss-knife toolset is required to address these issues. Oracle has come a long way in responding to the complex challenge of analyzing, troubleshooting, and fixing various SQL issues with a broad suite of SQL Advisors to address a wide spectrum of problem scenarios.

This chapter's pointed focus is on the SQL Tuning Advisor—the standard tool of choice for identifying, analyzing, and fixing SQL issues. It also presents a primer and general guidelines on how to leverage the SQL Access Advisor and SQL Repair Advisor. This chapter is a good starting point for intuitive and rapid fixing of SQL issues by covering the SQL tuning, access, and repair advisors. How-to steps to run the SQL Advisors in Oracle Enterprise Manager (OEM) 12c are illustrated throughout this chapter.

Alternative command-line interface steps for running the SQL Tuning Advisor and SQL Access Advisor are also covered in this chapter. A brief overview of the various available SQL performance analyzers is also presented.

OEM 12c—SQL Advisors Home

The SQL Advisors Home is a central place in OEM 12c providing easy access to all of the powerful SQL Advisors, shown in [Figures 12.1](#) and [12.2](#), which can be accessed in OEM 12c by the following menu hierarchy:

OEM 12c → Select Database → Main Menu → Performance → SQL → Advisors Home

The screenshot shows the 'Advisor Central' section of the OEM 12c interface. At the top, there are tabs for 'Advisors' (which is selected) and 'Checkers'. Below this, under the 'Advisors' heading, there is a list of advisors: ADDM, Maximum Availability Architecture (MAA) Advisor, Segment Advisor, Streams Performance Advisor, Automatic Undo Management, Memory Advisors, and SQL Advisors. The 'SQL Advisors' link is highlighted with a red box. To the right, under 'Data Recovery Advisor', are links for MTTR Advisor and SQL Performance Analyzer Home, also highlighted with red boxes. At the bottom left is a 'Search' field, and at the very bottom is a 'Page Footer' with the number 360.

Figure 12.1 A representation of the SQL Advisor links in Advisor Central

The screenshot shows the Oracle Database Advisor Central interface, specifically the SQL Advisors section. At the top, there's a navigation bar with links like Oracle Database, Performance, Availability, Security, Schema, Administration, and a Logged In status. Below the navigation, the title "SQL Advisors" is displayed. A brief description states: "The SQL Advisors address several important use cases having to do with SQL: identify physical structures optimizing a SQL workload, tune individual statements with heavy execution plans, identify and correct result set divergence, build test cases for failed SQL." Under this, there are three main sections: "SQL Access Advisor", "SQL Tuning Advisor", and "SQL Repair Advisor". Each section has a brief description and a link to its details or worksheet.

Figure 12.2 A representation of the SQL Advisors home page in Advisor Central

SQL Tuning Advisor

SQL Tuning Advisor is an advanced, sophisticated utility that performs comprehensive analysis and optimization of SQL statements. Generally, it is used for analyzing and optimizing top SQL statements that are performing poorly and are expensive to run in terms of system resources.

The SQL Tuning Advisor is a real time-saver of a tool that can dramatically cut down on tuning SQL by eliminating the need for lengthy, laborious, and complex manual tuning of SQL statements. It can be invoked automatically as part of the Automatic Maintenance Tasks job framework (AUTOTASK) or manually as and when the need arises.

SQL Tuning Advisor performs a deep-level analysis on the underlying SQL statement, the relevant statistics and execution plans, and so on, and generates a whole host of recommendations, including SQL profiles, SQL baselines, new index creation, identification of stale statistics, and rewriting SQL statements for optimization. The reasoning and proposed impact is also provided for each recommendation. Alternative SQL plans are listed along with cost benefits for each proposed recommendation.

SQL Tuning Advisor is the main tool used by AUTOTASK. However, SQL Tuning Advisor run from AUTOTASK does not operate on SQL statements that involve parallel queries, nonrepeatable ad-hoc SQL statements, recursive SQL, and SQL-profiled long-running queries. These types of SQL statements can be processed by running SQL Tuning Advisor manually on demand.

SQL Tuning Advisor can be run on individual SQL statements or SQL tuning sets. SQL Tuning Advisor requires a separate license as part of the Oracle Database Tuning Pack.

Note

The two most common and popular recommendation outcomes of running SQL Tuning Advisor on SQL statements are SQL profiles and indexes. SQL profiles enable the Optimizer to choose optimal execution plans by containing a set of ancillary statistics for the SQL statement. SQL profiles are advantageous from a performance-tuning perspective such that performance improvements can be achieved at the Oracle database server tier without changing code (SQL Hints).

Running SQL Tuning Advisor in OEM 12c

Oracle Enterprise Manager 12c Cloud Control is the preferred methodology for rapidly and easily running SQL Tuning Advisor. It can be accessed by following a few intuitive clicks:

OEM 12c → Select Database → Main Menu → Performance → SQL → SQL Tuning Advisor

[Figure 12.3](#) shows how to easily and intuitively run the SQL Tuning Advisor in OEM 12c on a SQL tuning set—a group of SQL statements. It can be run in either limited or comprehensive mode.

The screenshot shows the 'Schedule SQL Tuning Advisor' configuration page. At the top, there are fields for 'Name' (SQL\_TUNING\_3856472675875), 'Description' (Custom SQL Tuning Advisor Task), and 'SQL Tuning Set' (SQL\_Tuning\_Set123). Below these are sections for 'Scope' and 'Schedule'. In the 'Scope' section, 'Total Time Limit (minutes)' is set to 30, and 'Comprehensive' mode is selected. In the 'Schedule' section, the time zone is set to (UTC-05:00) US Eastern Time, and the run is scheduled 'Immediately'. To the right, there is an 'Overview' box with information about the SQL Tuning Advisor's function and a 'Top Activity' tab.

Figure 12.3 A representation of running SQL Tuning Advisor on a SQL tuning set (a set of SQL statements)

The time limit spent by the Advisor on each SQL statement can also be specified. Additionally, the schedule for running the SQL Tuning Advisor can be specified.

Alternatively, the SQL Tuning Advisor can be run from the Top Activity page and an individual SQL page in OEM, as shown in [Figures 12.4](#) and [12.5](#).

The screenshot shows the 'Top SQL' page with a table of active SQL statements. The columns are 'Select', 'Activity (%)', 'SQL ID', and 'SQL Type'. The data is as follows:

| Select | Activity (%) | SQL ID | SQL Type |
|-------------------------------------|--------------|---------------|----------------|
| <input type="checkbox"/> | 18.18 | 085tx0kjaww7h | CALL METHOD |
| <input type="checkbox"/> | 9.09 | 58ty8xga68qst | PL/SQL EXECUTE |
| <input type="checkbox"/> | 9.09 | 16x4ffvmkhtbb | PL/SQL EXECUTE |
| <input checked="" type="checkbox"/> | 9.09 | 16mtpnc196wr7 | SELECT |

Figure 12.4 Running SQL Tuning Advisor from the Top Activity page in OEM 12c

Top Activity > SQL Details: fx5sz971andba
SQL Details: fx5sz971andba

Switch to SQL ID: [] Go View Data Historical Refresh SQL Worksheet Actions SQL Tuning Advisor Go

> Text

```
SELECT D.TARGET_GUID, M.MARKER_TIMESTAMP, CM.MARKER_TIMESTAMP, C.START_COLLECTION_TIMESTAMP, D.TIMEZONE_REGION, D.OPT_CODE, C.CURRENT_STATUS,
D.CAN_CALCULATE, GP.GRACE_DELAY, MIN(CAST(FROM_TZ(CAST(MM.MARKER_TIMESTAMP AS TIMESTAMP), DD.EDEP_TIMEZONE_REGION)) AT TIME ZONE D.TIMEZONE_REGION AS DATE)), MAX(CAST(FROM_TZ(CAST(MM.MARKER_TIMESTAMP AS TIMESTAMP), DD.FDFP_TTMF70MF_RFGT0N) AT TIME ZONE D.TIMEZONE_REGION AS DATE)) AT TIME ZONE D.TIMEZONE_REGION AS DATE))
```

Figure 12.5 Running SQL Tuning Advisor from an individual SQL page in OEM

Figures 12.6, 12.7, 12.8, 12.9, and 12.10 show some of the sample recommendations along with an example comparison of explain plans. Implementing explain plans in OEM 12c is simply a matter of following the next click on the Implement button. Each recommendation's impact should be evaluated and measured for benefit analysis before finalizing the implementation.

Recommendations for SQL ID:16mtpnc196wr7

Only one recommendation should be implemented.

SQL Information

SQL Text: SELECT /\*+ CARDINALITYLockedtgt, 100 \*/ ENTITY\_GUID FROM EM\_MANAGEABLE\_ENTITIES T, TABLE(CAST(:B1 AS MGMT\_TARGET\_GUID\_ARRAY)) LOCKEDTGT WHERE REP\_SIDE\_AVAIL = 0 AND T.EMD\_URL = :B2 AND T.ENTITY\_GUID...

Select Recommendation

Original Explain Plan (Annotated)

Implement Validate with SPA

| Select | Type | Findings | Recommendations | Rationale | Benefit (%) | Other Statistics | New Explain Plan | Compare Explain Plans |
|----------------------------------|-------------------|---|--|---|-------------|------------------|------------------|-----------------------|
| <input checked="" type="radio"/> | Statistics | Optimizer statistics for table "SYSMAN"."MGMT_BLACKOUT_WINDOWS" are stale. | Consider collecting optimizer statistics for this table and its indices. | The optimizer requires up-to-date statistics for the table and its indices in order to select a good execution plan. | | | | |
| <input checked="" type="radio"/> | Alternative Plans | Some alternative execution plans for this statement were found by searching the system's real-time and historical performance data. | Consider creating a SQL plan baseline for the plan with the best average elapsed time. | Execution statistics from AWR show an average elapsed time of 0.0540s for the recommended plan, compared to 0.0505s for the original plan. Creating a plan baseline for the plan with the best elapsed time will prevent the Oracle optimizer from selecting a plan with worse performance. | 100 | 100 | 100 | |

Figure 12.6 A representation of SQL Tuning Advisor recommendations for stale statistics and creating a SQL plan baseline

Recommendations for SQL ID:fx5sz971andba

Only one recommendation should be implemented.

SQL Information

SQL Text: SELECT D.TARGET\_GUID, M.MARKER\_TIMESTAMP, CM.MARKER\_TIMESTAMP, C.START\_COLLECTION\_TIMESTAMP, D.TIMEZONE\_REGION, D.OPT\_CODE, C.CURRENT\_STATUS, D.CAN\_CALCULATE, GP.GRACE\_DELAY, MIN(CAST(FROM\_TZ(CAST(MM.MARKER\_TIMESTAMP AS TIMESTAMP), DD.EDEP\_TIMEZONE\_REGION)) AT TIME ZONE D.TIMEZONE\_REGION AS DATE)), MAX(CAST(FROM\_TZ(CAST(MM.MARKER\_TIMESTAMP AS TIMESTAMP), DD.FDFP\_TTMF70MF\_RFGT0N) AT TIME ZONE D.TIMEZONE\_REGION AS DATE)) AT TIME ZONE D.TIMEZONE\_REGION AS DATE))

Select Recommendation

Original Explain Plan (Annotated)

Implement Validate with SPA

| Select | Type | Findings | Recommendations | Rationale | Benefit (%) | Other Statistics | New Explain Plan | Compare Explain Plans |
|----------------------------------|-------------|---|--|-----------|-------------|------------------|------------------|-----------------------|
| <input checked="" type="radio"/> | SQL Profile | A potentially better execution plan was found for this statement. | Consider accepting the recommended SQL profile. No SQL profile currently exists for this recommendation. | | 57.78 | 100 | 100 | |

Figure 12.7 A representation of SQL Tuning Advisor recommendations for a SQL profile

Select Recommendation

Original Explain Plan (Annotated)

| Select | Type | Findings | Recommendations | Rationale | Benefit (%) | Other Statistics | New Explain Plan | Compare Explain Plans |
|----------------------------------|-------------------|---|--|--|-------------|------------------|------------------|-----------------------|
| <input checked="" type="radio"/> | Alternative Plans | Some alternative execution plans for this statement were found by searching the system's real-time and historical performance data. | Because no execution history for the Original Plan was found, the SQL Tuning Advisor could not determine if any of these execution plans are superior to it. However, if you know that one alternative plan is better than the Original Plan, you can create a SQL plan baseline for it. This will instruct the Oracle optimizer to pick it over any other choice in the future. | Creating a plan baseline for the plan with the best elapsed time will prevent the Oracle optimizer from selecting a plan with worse performance. | 100 | | | |
| <input checked="" type="radio"/> | Restructure SQL | An expensive cartesian product operation was found at line ID 14 of the execution plan. | Consider removing the disconnected table or view from this statement or add a join condition which refers to it. | A cartesian product should be avoided whenever possible because it is an expensive operation and might produce a large amount of data. | | | | |

Figure 12.8 A representation of SQL Tuning Advisor recommendations for a cartesian product operation

restructure SQL statement

| Select Recommendation | | | Recommendations | Rationale | Benefit (%) | Other Statistics | New Explain Plan | Compare Explain Plans |
|--|------|----------|--|-----------|-------------|------------------|------------------|-----------------------|
| Select | Type | Findings | | | | | | |
| ● SQL Profile 2 potentially better execution plans were found for this statement. Choose one of the following SQL profiles to implement. | | | Consider accepting the recommended SQL profile to use parallel execution for this statement. | | 99.99 | 0.00 | 0.00 | 0.00 |
| | | | Consider accepting the recommended SQL profile. | | 99.8 | 0.00 | 0.00 | 0.00 |

Figure 12.9 A representation of SQL Tuning Advisor recommendations for SQL profiles

| Compare Explain Plans | | | | | | | | | | |
|--|---------|--------|-------------|-------|-------------------|-------|-------------------------------------|-----------|-------------------------------|------------------------|
| Original Explain Plan (Annotated) | | | | | | | | | | |
| ● Indicates an adjustment from the original plan by the SQL Tuning Advisor | | | | | | | | | | |
| Plan Hash Value 534253291 | | | | | | | | | | |
| Expand All Collapse All | | | | | | | | | | |
| Operation | Line ID | Object | Object Type | Order | Rows | Bytes | Cost | Time | CPU Cost | I/O Cost |
| ▀ SELECT STATEMENT | 0 | | | 15 | | 0.128 | 24,881,414,144
0.128 | 2,990,555 | 8,612,740,868,341,760
0.00 | 24,422,076,416
0.00 |
| ▀ SORTAGGREGATE | 1 | | | 14 | | 0.128 | | | | |
| ▀ MERGEJOIN CARTESIAN | 2 | | | 13 | 7,935,818,240.000 | 0.128 | 24,881,414,144
7,935,818,240.000 | 2,990,555 | 8,612,740,868,341,760
0.00 | 24,422,076,416
0.00 |
| ▀ NESTED LOOPS | 3 | | | 10 | 273,830.812 | 0.128 | 7,619,559
273,830.812 | 916 | 70,140,379,136
0.00 | 7,615,818
0.00 |
| ▀ NESTED LOOPS | 4 | | | 8 | 254,095.266 | 0.128 | 5,092,192
254,095.266 | 613 | 47,350,857,728
0.00 | 5,089,667
0.00 |
| ▀ HASHJOIN | 5 | | | 5 | 209,690.266 | 0.128 | 37,658
209,690.266 | 5 | 5,485,237,760
0.00 | 37,365
0.00 |

| New Explain Plan With SQL Profile | | | | | | | | | | |
|---|---------|--------|-------------|-------|------|------------|---------------------|------|------------------------|-----------------|
| Plan Hash Value 3802554353 | | | | | | | | | | |
| Expand All Collapse All | | | | | | | | | | |
| Operation | Line ID | Object | Object Type | Order | Rows | Bytes | Cost | Time | CPU Cost | I/O Cost |
| ▀ SELECT STATEMENT | 0 | | | 38 | | 0.128 | 272,849
0.128 | 33 | 80,787,652,608
0.00 | 268,540
0.00 |
| ▀ SORTAGGREGATE | 1 | | | 37 | | 0.128 | | | | |
| ▀ PX COORDINATOR | 2 | | | 36 | | | | | | |
| ▀ PX SEND QC (RANDOM) | 3 | | | 35 | | 0.128 | | | | |
| ▀ SORTAGGREGATE | 4 | | | 34 | | 0.128 | | | | |
| ▀ HASHJOIN | 5 | | | 33 | | 21,236 | 272,849
21,236 | 33 | 80,787,652,608
0.00 | 268,540
0.00 |
| ▀ JOIN FILTER CREATE | 6 | | | 5 | | 68,069,672 | 2,009
68,069,672 | 1 | 229,767,760
0.00 | 1,997
0.00 |

Figure 12.10 A representation of SQL Tuning Advisor recommendations for a comparison of original and new explain plans by implementing a SQL profile

Running SQL Tuning Advisor Manually in SQL\*Plus

[Listing 12.1](#) illustrates the commands for creating, executing, and displaying the report of a SQL Tuning Advisor task in SQL\*Plus. The `USER_ADVISOR_TASKS` and `V$ADVISOR_PROGRESS` data dictionary views can be queried to monitor the progress of a SQL Tuning Advisor task.

Listing 12.1 Creating and Executing a SQL Tuning Advisor Task in SQL\*Plus

[Click here to view code image](#)

```
DECLARE
    sqlstmt    CLOB;
    taskname   VARCHAR2(30);
BEGIN
    sqlstmt :=      'select r.region,s.totalsales from sales s,
regions r where
                    s.region_no = r.region_no group by r.region_no';

    taskname :=     DBMS_SQLTUNE.CREATE_TUNING_TASK (
                    sql_text      => sqlstmt,
                    bind_list    => sql_binds(anydata.ConvertNumber(100)),
                    user_name    => 'tfm',
                    scope        => 'COMPREHENSIVE',
                    time_limit   => 60,
                    task_name    => 'totalsales_sql_sta_task',
                    description  => 'Individual Query - SQL Tuning Advisor
Task');
END;
/

BEGIN
    DBMS_SQLTUNE.EXECUTE_TUNING_TASK(task_name=>'totalsales_sql_sta_t
END;
/


SET LONGCHUNKSIZE 1500
SET LINESIZE 250
SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK( 'totalsales_sql_sta_task' )
FROM DUAL;
```

SQL Access Advisor

SQL Access Advisor is a powerful tuning tool that is used to analyze SQL workloads and provide performance improvement by suggesting recommendations for materialized views, partitioning, and indexes. SQL Access Advisor can be run on the following SQL workloads:

- Current and recent SQL activity
- SQL tuning set
- Hypothetical workload

SQL Access Advisor also provides recommendations for dropping unused indexes. In addition to covering full, refreshable materialized views, SQL Access Advisor can be leveraged to make materialized views fast-refreshable for an optimal implementation. SQL Access Advisor is part of the SQL Tuning Pack licensed option.

Note

All performance recommendations (materialized views, partitioning, and indexes) provided by the SQL Access Advisor must be measured for performance gains before solidifying the implementation of the resulting recommendations.

Running SQL Access Advisor in OEM 12c

The SQL Access Advisor can be launched from OEM 12c by the following menu hierarchy:

OEM 12c → Select Database → Main Menu → Performance → SQL → SQL Access Advisor

The SQL Access Advisor can be configured to validate and verify the usage of existing access structures *or* to recommend new access structures, including materialized views, indexes, and partitioning. This is illustrated in [Figure 12.11](#).



Figure 12.11 A representation of running options in SQL Access Advisor in OEM to verify the usage of existing access structures or recommend new access structures

[Figure 12.12](#) shows how to alternatively run the SQL Access Advisor on SQL tuning sets.

SQL Tuning Sets

A SQL Tuning Set is a collection of SQL Statements that can be used for tuning purposes.

| Search | Go | | |
|----------------------------------|-----------------------------|--------------------|------------------|
| Filter on a name or partial name | | | |
| Details | Drop | Copy To A Database | Export To A File |
| Schedule SQL Tuning Advisor | Schedule SQL Access Advisor | | |
| Select | Name | Schema | Description |
| <input checked="" type="radio"/> | STS_CUSTOM_9940 | SYS | 6913 |

Figure 12.12 Running SQL Access Advisor from the SQL Tuning Sets page in OEM

The workload source options can be specified for running the SQL Access Advisor, as illustrated in [Figure 12.13](#). These include Current and Recent Activity, Use an Existing SQL Tuning Set, and Create a Hypothetical Workload from specific schemas and tables.

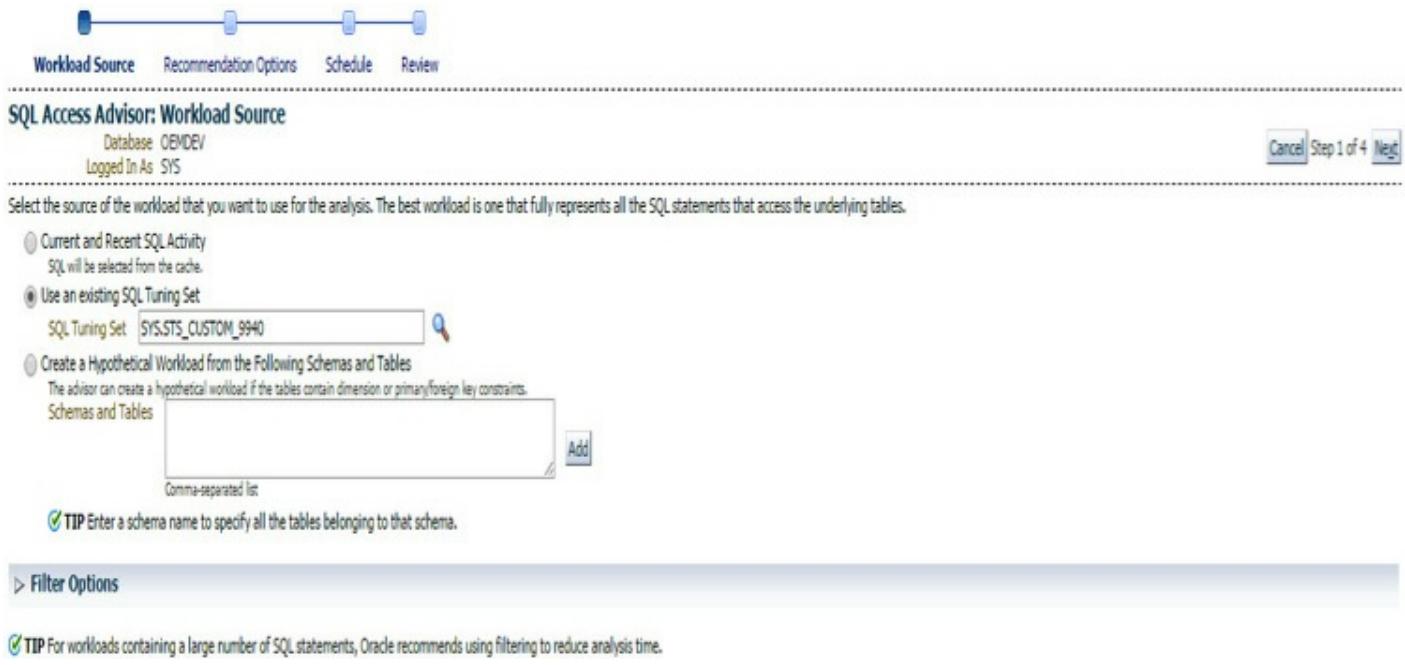


Figure 12.13 A representation of SQL Access Advisor wizard for workload source runtime options

Specific SQL Access structure recommendations can be specified for running the SQL Access Advisor, as illustrated in [Figure 12.14](#). These include materialized views, indexes, and partitioning. The Advisor Mode can also be specified as Limited or Comprehensive.



Figure 12.14 A representation of SQL Access Advisor wizard for structures runtime options

[Figure 12.18](#) shows the overall end results of running the SQL Access Advisor on a SQL tuning set gathered from current SQL cache activity along with a projected improvement in performance. [Figures 12.15](#), [12.16](#), [12.17](#), and [12.18](#) illustrate that drilling down into the individual recommendations gives you a detailed view for the specific recommendations, the projected potential for improvement of performance, and the exact steps for implementation.

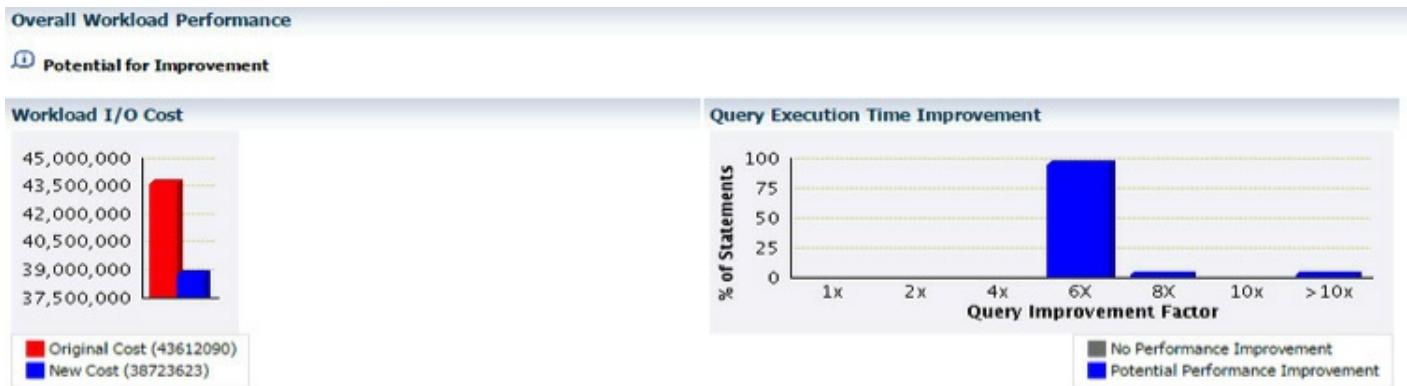


Figure 12.15 A representation of SQL Access Advisor results with potential for improvement

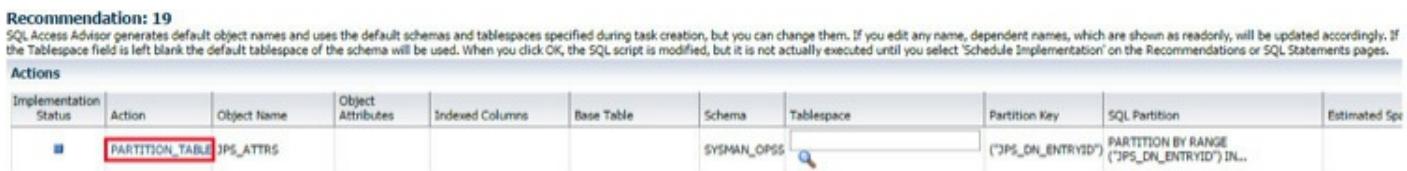


Figure 12.16 A representation of example recommendation in SQL Access Advisor for a partition table

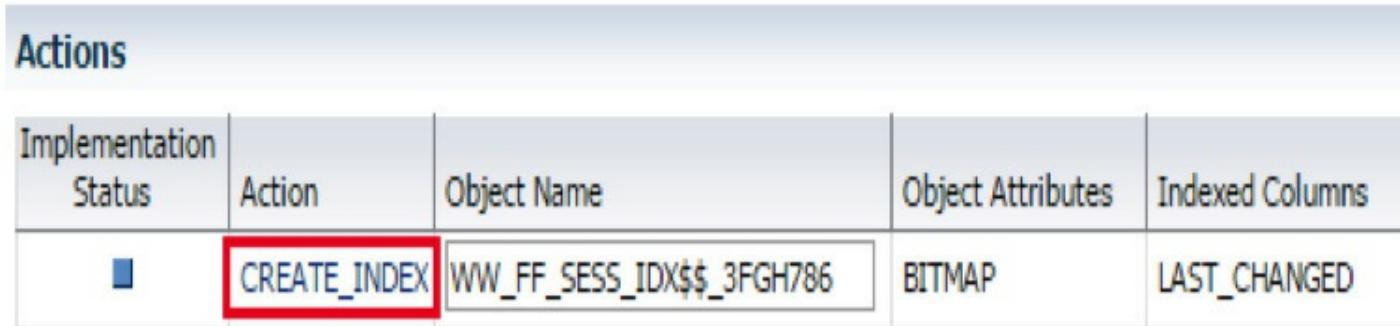


Figure 12.17 Example recommendation in SQL Access Advisor to create an index

Actions

Set Schema for All Actions Go

Set Tablespace for All Actions Go

| Implementation Status | Action | Object Name | Object Attributes | Indexed Columns |
|-----------------------|------------------------------|---------------------------------|-------------------|-----------------|
| ■ | CREATE_MATERIALIZED_VIEW_LOG | | | |
| ■ | CREATE_MATERIALIZED_VIEW | MV\$\$_2BDD0000 | General Match | |
| ■ | GATHER_TABLE_STATISTICS | MV\$\$_2BDD0000 | | |
| ■ | CREATE_INDEX | MV\$\$_2BDD0005_IDX\$\$_2BDD000 | BTREE | C4, C2, C1 |

Figure 12.18 Example recommendation in SQL Access Advisor to create a materialized view

Running SQL Access Advisor Manually in SQL\*Plus

[Listing 12.2](#) illustrates the commands for creating, configuring, and executing an example SQL Access Advisor task in SQL\*Plus. In this example, the code is generated from an OEM 12c SQL Access Advisor Task run. The *USER\_ADVISOR\_SQLA\_WK\_STMTS* and *USER\_ADVISOR\_ACTIONS* data dictionary views can be queried to view the recommendations generated by the SQL Access Advisor.

Listing 12.2 Creating and Executing a SQL Access Advisor Task in SQL\*Plus

[Click here to view code image](#)

```

DECLARE
    taskname varchar2(30)          := 'SQL_ACCESS_9940';
    task_desc varchar2(256)        := 'SQL Access Advisor';
    task_or_template varchar2(30)  := 'SQLACCESS_EMTASK';
    task_id number                 := 0;
    num_found number;
    sts_name varchar2(256)        := 'STS_CUSTOM_9940';
    sts_owner varchar2(30)        := 'SYS';

BEGIN
    Create the SQL Access Advisor Task
    dbms_advisor.create_task(DBMS_ADVISOR.SQLACCESS_ADVISOR,
    task_id,taskname,task_desc,task_or_template);

    -- Reset the SQL Access Advisor Task
    dbms_advisor.reset_task(taskname);

    -- Delete Previous STS Workload Task Link

```

```

select count(*) into num_found from user_advisor_sqla_wk_map where
task_name = taskname and workload_name = sts_name;
IF num_found > 0 THEN
dbms_advisor.delete_sts_ref(taskname, sts_owner, sts_name);
END IF;

-- Link STS Workload to Task
dbms_advisor.add_sts_ref(taskname, sts_owner, sts_name);

-- Set the STS Workload Parameters
dbms_advisor.set_task_parameter(taskname, 'VALID_ACTION_LIST', DBMS ADVISED
dbms_advisor.set_task_parameter(taskname, 'VALID_MODULE_LIST', DBMS ADVISED
dbms_advisor.set_task_parameter(taskname, 'SQL_LIMIT', DBMS ADVISED.ADVISOR
dbms_advisor.set_task_parameter(taskname, 'VALID_USERNAME_LIST', DBMS ADVISED
dbms_advisor.set_task_parameter(taskname, 'VALID_TABLE_LIST', DBMS ADVISED
dbms_advisor.set_task_parameter(taskname, 'INVALID_TABLE_LIST',
DBMS ADVISED.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname, 'INVALID_ACTION_LIST',
DBMS ADVISED.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname, 'INVALID_USERNAME_LIST',
DBMS ADVISED.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname, 'INVALID_MODULE_LIST',
DBMS ADVISED.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname, 'VALID_SQLSTRING_LIST',
DBMS ADVISED.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname, 'INVALID_SQLSTRING_LIST', '');

/* Set Task Parameters */
dbms_advisor.set_task_parameter(taskname, 'ANALYSIS_SCOPE', 'ALL');
dbms_advisor.set_task_parameter(taskname, 'RANKING_MEASURE', 'PRIORITY');
dbms_advisor.set_task_parameter(taskname, 'DEF_PARTITION_TABLESPACE',
DBMS ADVISED.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname, 'TIME_LIMIT', 10000);
dbms_advisor.set_task_parameter(taskname, 'MODE', 'COMPREHENSIVE');
dbms_advisor.set_task_parameter(taskname, 'STORAGE_CHANGE', DBMS ADVISED
dbms_advisor.set_task_parameter(taskname, 'DML_VOLATILITY', 'TRUE');
dbms_advisor.set_task_parameter(taskname, 'WORKLOAD_SCOPE', 'PARTIAL');
dbms_advisor.set_task_parameter(taskname, 'DEF_INDEX_TABLESPACE',
DBMS ADVISED.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname, 'DEF_INDEX_OWNER', DBMS ADVISED
dbms_advisor.set_task_parameter(taskname, 'DEF_MVIEW_TABLESPACE',
DBMS ADVISED.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname, 'DEF_MVIEW_OWNER', DBMS ADVISED
dbms_advisor.set_task_parameter(taskname, 'DEF_MVLOG_TABLESPACE',
DBMS ADVISED.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname, 'CREATION_COST', 'TRUE');
dbms_advisor.set_task_parameter(taskname, 'JOURNALING', '4');
dbms_advisor.set_task_parameter(taskname, 'DAYS_TO_EXPIRE', '30');

-- Execute the SQL Access Advisor Task
dbms_advisor.execute_task(taskname);

```

END;

SQL Repair Advisor

SQL Repair Advisor is leveraged for repairing SQL statements at the database server tier that end up in critical failure. SQL Repair Advisor performs a comprehensive analysis on the failed SQL statement and may provide a SQL patch recommendation that fixes the failure issue by instructing the Query Optimizer to select a different execution plan that avoids failure. Implemented SQL patches can be managed in the SQL Plan Control page in OEM 12c. Alternatively, the DBMS\_SQLDIAG package can be leveraged for running the SQL Repair Advisor from a command-line interface. SQL Repair Advisor is not covered in extensive detail in this chapter.

[Figure 12.19](#) illustrates how SQL Repair Advisor can be launched from the Support Workbench link from the SQL Advisor homepage in OEM 12c by the following menu hierarchy:

OEM 12c → Select Database → Main Menu → Performance → Advisors Home → SQL Advisors → SQL Repair Advisor

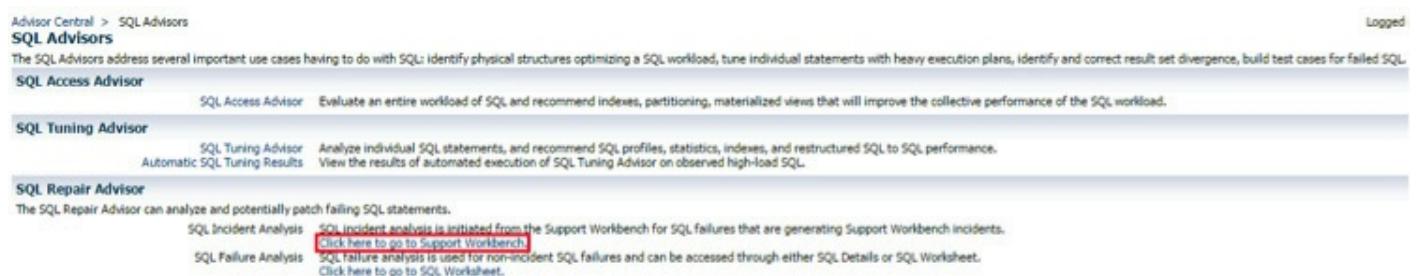


Figure 12.19 A representation of running SQL Repair Advisor from the Advisor Central page in OEM 12c

[Figure 12.20](#) shows how SQL Repair Advisor can be invoked from the Problem Details page from Support Workbench.



Figure 12.20 A representation of running SQL Repair Advisor from the Support Workbench Problem Details page in OEM 12c

SQL Performance Analyzer

In addition to the SQL Advisors covered in the previous sections, the SQL Performance Analyzer is a sophisticated and advanced tool that enables an Oracle DBA to analyze the impact of major changes at the Oracle database system level—for example, version upgrades, optimizer statistics, initialization parameter changes, and migration to Exadata. SQL Performance Analyzer is not covered in extensive detail in this chapter.

SQL Performance Analyzer can prove to be a very effective tool for proactively predicting, forecasting, and avoiding SQL performance issues. SQL Performance Analyzer is part of the Real Application Testing license option.

[Figure 12.21](#) shows the launch link in Advisor Central in OEM 12c. [Figure 12.22](#) shows the various workflow options available for the SQL Performance Analyzer in OEM 12c.

The screenshot shows the 'Advisor Central' interface. At the top, there are tabs for 'Advisors' (which is selected) and 'Checkers'. Below this, under the 'Advisors' section, there is a list of advisors: ADDM, Maximum Availability Architecture (MAA) Advisor, Segment Advisor, Streams Performance Advisor, Automatic Undo Management, Memory Advisors, SQL Advisors, Data Recovery Advisor, MTRR Advisor, and SQL Performance Analyzer Home. The 'SQL Performance Analyzer Home' link is highlighted with a red box.

Figure 12.21 A representation of running SQL Performance Analyzer from the Advisor Central page in OEM 12c

The screenshot shows the 'SQL Performance Analyzer Home' page. At the top, it says 'SQL Performance Analyzer Home' and 'Page Refreshed'. Below this, a section titled 'SQL Performance Analyzer Workflows' is shown. It contains a list of workflow options: 'Upgrade from 9i or 10.1', 'Upgrade from 10.2 or 11g', 'Parameter Change', 'Optimizer Statistics', 'Exadata Simulation', and 'Guided Workflow'. These first four items are grouped together and enclosed in a red box. To the right of each option is a brief description of what it does. At the bottom right of the page, there is a link 'Create a SQL Performance Analyzer Task and execute custom experiments using manually created SQL trials.'

Figure 12.22 Workflow options in SQL Performance Analyzer

Summary

Fixing SQL issues is an integral part of an Oracle DBA's troubleshooting activities. This chapter acquainted you with the various SQL Advisors available in the Oracle database server—powerful and sophisticated tools that can save valuable time by automating the tuning process and eliminating manual diagnostic work. Key step-by-step instructions on how to run the various SQL Tuning Advisors were also covered in this chapter.

13. Extending Data Pump for Data and Object Migration

The Data Pump feature of the Oracle database is a high-speed, highly scalable mechanism for transferring data and objects from one database to another (using database links), from a database to a file (export), or from a file to a database (import). Data Pump uses direct-path loading and unloading technologies to improve performance.

Data Pump can be invoked using the following methods:

- Exclusive Data Pump export and import tools, using `expdp` and `impdp`
- PL/SQL code calling the `DBMS_DATAPUMP` package to move data and `DBMS_METADATA` package to extract metadata (objects)
- Oracle Enterprise Manager Cloud Control
- Oracle SQL Developer

Data Pump comes with various options and parameters, and when you consider the possibilities of having various combinations of parameters, the options are endless—which obviously makes providing examples for each combination difficult. This chapter shows specific situations and how Data Pump utilities can be used for day-to-day database administration activities.

Using Data Pump

Data Pump was introduced in Oracle Database 10g and has evolved over the versions with various features and performance improvements in moving data and objects. The legacy export/import (`exp/imp`) utility is still maintained in all versions of database, including Oracle 12c, for backward compatibility and to migrate legacy data to newer versions of the database. This chapter shows you how to extend Data Pump for your various data and object migration needs.

Before we start, let us review the basics behind the export and import tools:

- There are two sets of export/import tools available in Oracle: original export and import utilities (`exp/imp`) are the oldest and are maintained in the newer versions. New Oracle Data Pump export and import utilities (`expdp/impdp`) are available since Oracle 10g.
- Legacy `exp/imp` is a client-side tool, meaning the dump file is written to (and read from) a local directory on the machine where you run the tool. Data Pump is a server-side utility, meaning the dump file is written to (and read from) a directory on the database server. By default, the file is written to (or read from) the directory defined by the `DATA_PUMP_DIR` directory (in `dba_directories`) location.
- Data Pump export and import jobs can be paused, stopped, and restarted. Legacy `exp/imp` does not have this option.

- Data Pump provides a very powerful interactive command-line mode that allows you to monitor and control Data Pump export and import operations on a different terminal.
- Data Pump jobs can be initiated using the expdp/impdp tools or by using the Procedural Language/Structured Query Language (PL/SQL) package DBMS\_DATAPUMP. The expdp and impdp tools are part of the Oracle client install.

The legacy exp/imp is not discussed in this chapter, but if you are a hardcore exp/imp fan, from Oracle Database 11g Release 2 onward, you can still use the same parameters you use with exp/imp in expdp/impdp (imp or exp with `help=y` shows you the available options and parameters). Data Pump converts the legacy parameters to Data Pump-specific parameters internally and executes the job. This is useful for DBAs who resist change to the Data Pump tools. Although using Data Pump in legacy mode is less flexible and offers fewer options than using the full-featured product, it is a good start to using Data Pump tools.

Copying Objects

The most basic use of Data Pump is to copy objects or schemas from one database to another, regardless of whether they are using the same version or different versions. If the objects do not exist in the target, the copy job is quicker and easier because there is no preparation or cleanup required. Before starting any Data Pump export job, you must confirm a few things:

- Are you writing the export dump file to a directory? If yes, do you have enough space to save the dump file?
- Is there a directory object pointing to that directory? Or is your default DATA\_PUMP\_DIR directory location good enough to save the export dump file?
- Can you skip writing the dump file and copy the objects directly using the network link feature of Data Pump, utilizing a database link?
- What options or parameters should you use to do the export and import?
- Is the character set the same or compatible between the databases where export/import is performed?

If you need to copy data or objects from Oracle Database 9*i* or older, there is no Data Pump; you have to resort to the legacy exp/imp tools. Legacy exp/imp utilities are still available in Oracle Database 12*c*.

This chapter discusses a large number of options that can be used with Data Pump. Due to page constraints, we cannot show sample output for all code used in this chapter.

Tip

If you are not sure how much space is required for the export dump file, you can run expdp with the ESTIMATE\_ONLY=Y option, which gives you the expected export size of the tables in the dump file.

Data Pump Modes

Normally, we consider export/import (using exp/imp or expdp/impdp) when we have to

- Copy a table from one database to another. Use the table mode export with the TABLES= parameter along with other parameters.
- Copy a schema from one database to another, or duplicate schema objects under another user in the same database. Use the schema mode export with the SCHEMAS= parameter along with other parameters.
- Copy the entire database, usually performed for platform migration or to backup. Use the full mode export with the FULL=Y parameter along with other parameters.

Though table, schema, and full are the most commonly used modes, two more modes are available in Data Pump export/import:

- **Tablespace mode:** Using the TABLESPACES= parameter, you can specify the tablespace names, and only objects belonging to the tablespace are exported. The dependent objects, even if they belong to another tablespace, are exported. For example, when exporting a table, if the indexes belong to another tablespace, they will still be exported.
- **Transport tablespace mode:** When the TRANSPORT\_TABLESPACES= parameter is used, only the metadata for the tables and their dependent objects within a specified set of tablespaces is exported. This mode is used to copy a tablespace physically (by copying the datafiles belonging to the tablespace) from one tablespace to another.

Oracle documentation and invoking expdp/impdp with HELP=Y provides you help with the modes of export. This chapter shows examples of not-so-common object and data migration requirements accomplished using Data Pump.

Though introduction to Data Pump is not an objective of this chapter, let us quickly review the common parameters used in expdp and impdp:

- **DIRECTORY:** This is a database directory location with permission for the Oracle database owner to read and write. The default value and location for this parameter is defined by the DATA\_PUMP\_DIR directory.
- **DUMPFILE:** This is the name of the dump file. The default value is expdat.dmp.
- **LOGFILE:** This is the name of the log file. The default value is export.log.
- **FULL/SCHEMAS/TABLES:** These are mutually exclusive parameters to specify the export/import mode.
- **CONTENT:** This specifies if the export/import should include metadata only, data only, or both metadata and data.
- **EXCLUDE/INCLUDE:** These are mutually exclusive parameters to refine the object types and names of objects to include or exclude in an export/import.

Working with Private and Public Objects

Public objects do not belong to any schema, and there is no PUBLIC schema per se for you to specify PUBLIC as the schema name while exporting to get all PUBLIC objects. Almost all objects that come under a schema can be considered private to the schema owner. One very special private object is a database link, as you cannot create a database link qualifying the link name with a schema name. Pretty much all other objects can be created by a DBA qualifying the object name with a schema name to create under a specific schema. Private database links are always created under the current user only. This section shows you how to export database links and synonyms.

Saving and Restoring Database Links

During database refreshes, the database links in the target database get wiped. When there are many private database links in the database, scripting and creating the database links become difficult because you cannot qualify the database link with a schema name to create the database link. You must login as the owner on the database link to create the database link. However, Data Pump can be used to make this process simple.

The EXCLUDE and INCLUDE parameters of Data Pump decides which objects to include or exclude. These parameters are very powerful and granular enough to export or import any type of object, including a database link.

The database links for the whole database can be saved to a dump file using

[Click here to view code image](#)

```
$ expdp dumpfile=uatdblinks.dmp directory=clone_save full=y  
include=db_link userid =\"/ as sysdba\"
```

The code exports public database links and all private database links. If you are interested only in the private database links belonging few schema names, you could export using

[Click here to view code image](#)

```
$ expdp dumpfile=uatdblinks.dmp directory=clone_save  
schemas=APPS,XX,XXL include=db_link userid =\"/ as sysdba\"
```

The code gets clumsy when you try to include complex parameters in the command line, especially with the addition of escape character \ before each ', ", and , special characters. It is better to use a parameter file so you do not have to worry about the escape characters. From now on, the examples in this chapter will show only the relevant parameters. Assume the inclusion of dumpfile, logfile, directory, and userid parameters with all examples.

Exporting Public Database Links and Synonyms

What if we need to save only the public database links in the export, or we want to follow a pattern matching for schema names? The EXCLUDE and INCLUDE parameters are very flexible and powerful, and they accept a subquery instead of hard-coded values.

It is easy to figure out the public synonyms or public database links by querying the DBA\_SYNONYMS or DBA\_DB\_LINKS views. Here is how you export only the public database links:

[Click here to view code image](#)

```
full=y
include=db_link:"IN (SELECT db_link
                      from dba_db_links Where Owner = 'PUBLIC')"
```

Since a query is used, you can imagine the possibilities are endless if you can write the appropriate query.

The next example shows how to export the public synonyms defined on the schema HR. This is useful when you have to copy a schema from one database to another. The schema level export only copies the database objects owned by the schemas you list in the SCHEMAS parameter. Public database links and public synonyms are included in the export only if full database export (FULL=Y) is performed.

[Click here to view code image](#)

```
full=y
include=DATABASE_EXPORT/SCHEMA/PUBLIC_SYNONYM/SYNONYM:"IN (SELECT
synonym_name from dba_synonyms
where Owner = 'PUBLIC' and table_owner = 'HR')"
```

Verifying Content of the Export Dump File

How do you verify if the expdp exported the database links (or, for that matter, any object) properly; in other words, how do you validate that the INCLUDE and EXCLUDE parameters you specified did what you expected them to do? You can use the impdp import with the SQLFILE parameter:

[Click here to view code image](#)

```
$ impdp dumpfile=pubdblinks.dmp directory=clone_save
  full=y sqlfile=testfile.txt
```

Instead of doing the actual import, the metadata from the dump file is written to the file name specified in the SQLFILE parameter. Reading the content of the output file shows you what is exported:

[Click here to view code image](#)

```
$ cat testfile.txt
-- CONNECT SYS
ALTER SESSION SET EVENTS '10150 TRACE NAME CONTEXT FOREVER, LEVEL
1';
ALTER SESSION SET EVENTS '10904 TRACE NAME CONTEXT FOREVER, LEVEL
1';
ALTER SESSION SET EVENTS '25475 TRACE NAME CONTEXT FOREVER, LEVEL
1';
ALTER SESSION SET EVENTS '10407 TRACE NAME CONTEXT FOREVER, LEVEL
```

```

1';
ALTER SESSION SET EVENTS '10851 TRACE NAME CONTEXT FOREVER, LEVEL
1';
ALTER SESSION SET EVENTS '22830 TRACE NAME CONTEXT FOREVER, LEVEL
192 ';
-- new object type path: DATABASE_EXPORT/SCHEMA/DB_LINK
CREATE PUBLIC DATABASE LINK "CZPRD.BT.NET"
    USING 'czprd';
CREATE PUBLIC DATABASE LINK "ERP_ARCHIVE_ONLY_LINK.BT.NET"
    CONNECT TO "AM_STR_HISTORY_READ" IDENTIFIED BY VALUES
'05EDC7F2F211FF3A79CD5A526EF812A0FCC4527A185146A206C88098BB1FF8F0B1
    USING 'ILMPRD1';

```

If the dump file is big with rows exported, and if you are not interested in the rows, make sure you specify the CONTENT=METADATA\_ONLY parameter along with SQLFILE.

Finding Valid INCLUDE and EXCLUDE Values

Metadata filtering is implemented through the EXCLUDE and INCLUDE parameters, and these parameters are mutually exclusive. In the previous examples, you saw the use of the EXCLUDE parameter. You can specify any valid database object in the EXCLUDE and INCLUDE parameters. So, for example, you can exclude constraints, but how do you know if you can exclude NOT NULL constraints from being exported? That is, how do you determine what are the valid values for EXCLUDE and INCLUDE parameters?

Depending on the type of export performed, you can query a database view to find the valid values:

- To find the valid object types that can be excluded and included during full database export/import, query the DATABASE\_EXPORT\_OBJECTS view.
- To find the valid object types that can be excluded and included during schema-level export/import, query the SCHEMA\_EXPORT\_OBJECTS view.
- Similarly, use the TABLE\_EXPORT\_OBJECTS view object types during table-level or tablespace-level export/import.

Following is an example query from DATABASE\_EXPORT\_OBJECTS, looking for valid options to exclude constraints:

[Click here to view code image](#)

```

SQL> SELECT object_path, comments
      FROM database_export_objects
     WHERE object_path like '%CONSTRAINT%'
       AND object_path not like '%/%';

```

| OBJECT_PATH | COMMENTS |
|-------------|---|
| CONSTRAINT | Constraints (including referential constraints) |

```
REF_CONSTRAINT  
Referential constraints
```

Output shows that it is possible to exclude and include all constraints, or just referential constraints (foreign key constraints). By the way, when you exclude or include constraints, it does not consider NOT NULL constraints.

The syntax of the EXCLUDE and INCLUDE parameters allows a name clause along with the object type. The name clause evaluates the expression. The following examples help you understand the options and power of EXCLUDE and INCLUDE:

- The following exports only the foreign key constraints belonging to HR schema:

```
schemas=HR  
include='REF_CONSTRAINT'
```

- The following exports a full database, except HR and SH schemas; notice the use of the IN expression:

[Click here to view code image](#)

```
full=y  
exclude=schema:"in ('HR', 'SH')"
```

- The following validates the full export did exclude HR and SH (it uses the impdp with SQLFILE). This is validated if there is a CREATE USER statement for HR and SH users. INCLUDE=USER only does the user creation; no schema objects are imported.

```
sqlfile=validate.txt  
include=USER
```

- The following exports all tables in the database that end with TEMP and do not belong to the SYSTEM and APPS schemas. Remember, we cannot use both INCLUDE and EXCLUDE at the same time.

[Click here to view code image](#)

```
full=y  
include=table:"IN (select table_name from dba_tables  
      where table_name like '%TEMP'  
      and owner not in ('SYSTEM', 'APPS'))"
```

Exporting Subsets of Data

Sometimes when copying data to a test instance, you are not interested in all of the rows in a huge table but need only current data or a sample of data. There are a couple of different methods to export a subset of data from a table instead of exporting all rows. The most common and easy method is to use the QUERY parameter. As with other parameters, it is easier to have this parameter in a parameter file so that you need not worry about using the escape characters for all quotes and parentheses when doing on

command line.

The WHERE clause you specify in QUERY is applied to all tables included in the export, unless you restrict the query to a single table by specifying the table name before the WHERE clause. For example, if you want to export the GL schema, but want only the last 6 months of data in the GL\_TRANSACTIONS and GL\_LINES table, you would do the following:

[Click here to view code image](#)

```
schemas=GL
query=GL_TRANSACTIONS:"where account_date > add_months(sysdate,
-6)"
query=GL.GL_LINES:" where transaction_date > add_months(sysdate,
-6)"
```

Notice the use of QUERY twice. This is required because we cannot specify more than one table before the WHERE clause used in QUERY.

That brings up a good question: What if you have many tables to which to apply this condition; do you have to repeat each table with the QUERY parameter? Yes, and you can. Or you can do two separate exports using the EXCLUDE or INCLUDE clause.

Let's check out another example. Expanding on the earlier example, assume we want to apply the query condition to all tables that end with TRANSACTION. You need to perform two exports. First, export with these parameters:

[Click here to view code image](#)

```
schemas=GL
include=table:"like '%TRANSACTION'"
query="where account_date > add_months(sysdate, -6)"
```

Perform the second export without the QUERY parameter:

[Click here to view code image](#)

```
schemas=GL
exclude=table:"like '%TRANSACTION'"
```

It is also possible to export a subset of data from a table using the SAMPLE parameter by specifying a percentage of rows to export. Similar to QUERY, you can limit the sample condition to just one table or to the entire export. Following is an example of using SAMPLE parameter:

```
schemas=GL
sample=GL_TRANSACTIONS:20
sample=GL.GL_LINES:30
```

All the tables in the GL schema will be exported. For GL\_TRANSACTIONS, only 20 percent of the rows are exported, and for GL\_LINES only 30 percent of the rows exported.

The QUERY and SAMPLE conditions operate on a single table during export. In some

circumstances, you want to export data that is a result of a complex join condition between multiple tables. You may be able to create a view, but export Data Pump until Oracle Database 11g Release 2 exports only the view definition, not the view data. From Oracle Database 12c onward, there is the VIEWS\_AS\_TABLES parameter to export the data in a view as a table. This is another good way to export a subset of data if the dataset is based on more than one table or complex joins.

In the following example, data from a view is exported. When you import, they go into the target database as a table.

Here is the information from the source database showing the columns, data type of EMP\_DEPT, and that it is a view:

[Click here to view code image](#)

```
SQL> desc emp_dept
Name           Null?    Type
-----
DEPTNO          NOT NULL NUMBER(2)
DNAME            VARCHAR2(14)
ENAME            VARCHAR2(10)

SQL> select object_type from user_objects where object_name =
'EMP_DEPT';

OBJECT_TYPE
-----
VIEW
```

The following parameter is used to do the export of the view data:

[Click here to view code image](#)

```
views_as_tables=scott.emp_dept
```

Here is output from export:

[Click here to view code image](#)

```
$ expdp parfile=p1.txt
Export: Release 12.1.0.2.0 - Production on Wed Apr 8 22:06:49 2015
Copyright (c) 1982, 2014, Oracle and/or its affiliates. All rights reserved.
Connected to: Oracle Database 12c Enterprise Edition Release
12.1.0.2.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real
Application Testing options
Starting "SYSTEM"."SYS_EXPORT_TABLE_01": system/********@orcl
parfile=p1.txt
Estimate in progress using BLOCKS method...
Processing object type TABLE_EXPORT/VIEWS_AS_TABLES/TABLE_DATA
Total estimation using BLOCKS method: 16 KB
Processing object type TABLE_EXPORT/VIEWS_AS_TABLES/TABLE
```

```

. . . exported "SCOTT"."EMP_DEPT"                                6.234 KB      14
rows
Master table "SYSTEM"."SYS_EXPORT_TABLE_01" successfully
loaded/unloaded
*****
Dump file set for SYSTEM.SYS_EXPORT_TABLE_01 is:
  /home/oracle/app/oracle/admin/cdb1/dpdump/x1.dmp
Job "SYSTEM"."SYS_EXPORT_TABLE_01" successfully completed at
                           Wed Apr 8 22:06:59 2015 elapsed 0
00:00:08

```

Now, the following parameter file imports the contents of the dump file to another database, under the MIKE schema:

```
remap_schema=scott:mike
```

The output from import shows the view is created as a table, and rows are imported:

[Click here to view code image](#)

```

$ impdp parfile=p1.txt
Import: Release 12.1.0.2.0 - Production on Wed Apr 8 22:09:42 2015
Copyright (c) 1982, 2014, Oracle and/or its affiliates. All rights
reserved.

Connected to: Oracle Database 12c Enterprise Edition Release
12.1.0.2.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real
Application Testing options
Master table "SYSTEM"."SYS_IMPORT_FULL_01" successfully
loaded/unloaded
Starting "SYSTEM"."SYS_IMPORT_FULL_01":  system/********@orcl
parfile=p1.txt
Processing object type TABLE_EXPORT/VIEWS_AS_TABLES/TABLE
Processing object type TABLE_EXPORT/VIEWS_AS_TABLES/TABLE_DATA
. . . imported "MIKE"."EMP_DEPT"                                6.234 KB      14
rows
Job "SYSTEM"."SYS_IMPORT_FULL_01" successfully completed at
                           Wed Apr 8 22:09:47 2015 elapsed 0
00:00:03

```

Finally, the object type and columns are validated using the same queries used before:

[Click here to view code image](#)

```

SQL> desc emp_dept
Name          Null?    Type
-----
DEPTNO        NUMBER(2)
DNAME          VARCHAR2(14)
ENAME          VARCHAR2(10)
SQL> select object_type from user_objects where object_name =
'EMP_DEPT';

```

```
OBJECT_TYPE
```

```
-----  
TABLE
```

Changing Object Properties

Data Pump import offers a very flexible way to change the properties of objects during import. This section uses examples to make you aware of the capabilities. These examples don't go into a lot of detail or provide syntax, but you can always review the documentation to learn more.

Importing Partitioned Tables as Nonpartitioned

This example uses Data Pump to export the partitioned table SALES\_DATA owned by SLS schema. During import, you need to create the table as non-partitioned under SLS\_HIST schema. The REMAP\_SCHEMA parameter changes the schema owner and the MERGE option for partitions creates the table as non-partitioned.

```
REMAP_SCHEMA=SLS:SLS_HIST  
TABLES=SLS.SALES_DATA  
PARTITIONS_OPTIONS=MERGE
```

Importing Table Partitions as Individual Tables

This example uses Data Pump to export the partitioned table SALES\_DATA owned by the SLS schema. During import, you need to create a separate nonpartitioned table for each partition of SALES\_DATA.

```
PARTITION_OPTIONS=DEPARTITION  
TABLES=SLS.SALES_DATA
```

Since the partition names in SALES\_DATA are SDQ1, SDQ2, SDQ3, and SDQ4, the new table names created would be SALES\_DATA\_SDQ1, SALES\_DATA\_SDQ2, SALES\_DATA\_SDQ3, SALES\_DATA\_SDQ4.

Masking Data

Often it is necessary to copy tables and schema from a production database to a nonproduction database where sensitive data needs to be scrambled or masked in the nonproduction database. Export and import in Data Pump provide a masking feature using the REMAP\_DATA parameter.

Employee table HR.ALL\_EMPLOYEES contains the social security numbers of employees along with other information. When copying this table from a production to a development database, you want to scramble the social security numbers, or mask them, or convert them to different values:

1. First, create a package and a function that returns the scrambled SSN when the production SSN is passed in as a parameter. Let's call this procedure HR\_UTILS.SSN\_SCRAMBLE.

2. During import, use the REMAP\_DATA parameter to change the data value:

[Click here to view code image](#)

```
TABLES=HR.ALL_EMPLOYEES  
REMAP_DATA=HR.ALL_EMPLOYEES.SSN:HR.HR_UTILS.SSN_SCRAMBLE
```

If you want to secure the dump file, it may be appropriate to use this function and parameter in the export itself. Thus, the dump file will not have SSNs but instead the scrambled numbers only.

Renaming Tables or Different Tablespaces

Sometimes you have to bring a table back from export backup but with a different name. In this example, the table DAILY\_SALES is imported to the database as DAILY\_SALES\_031515. The export dump is a SALES schema export. The original table is on the tablespace SALES\_DATA, and the restored table needs to go on the tablespace SALES\_BKUP.

[Click here to view code image](#)

```
TABLES=SALES.DAILY_SALES  
REMAP_TABLE=SALES.DAILY_SALES:DAILY_SALES_031515  
REMAP_TABLESPACE:SALES_DATA:SALES_BKUP
```

Using Default Storage Parameters

If you do not wish import to bring any of the attributes associated with a table or index segment, and you want to instead have the object take the defaults for the user and the destination database/tablespace, you can do so easily with a transform option. In this example, SALES schema objects are imported, but the tablespace, storage attributes, and logging clause all will default to what is defined for the user MIKE and his default tablespace:

[Click here to view code image](#)

```
SCHEMAS=SALES  
REMAP_SCHEMA=SALES:MIKE  
TRANSFORM=SEGMENT_ATTRIBUTES:N
```

In this example, if you would like to keep the original tablespace but only want to remove the storage attributes for the object type index, you could specify the optional object along with the TRANSFORM parameter. The valid object type values are TABLE or INDEX. In the following example, tables will use the default tablespace and storage characteristics, and indexes will go to the source index tablespace but will not use any of the source storage clauses:

[Click here to view code image](#)

```
SCHEMAS=SALES  
REMAP_SCHEMA=SALES:MIKE  
TRANSFORM=SEGMENT_ATTRIBUTES:N:TABLE
```

TRNASFORM=STORAGE :N :INDEX

Resizing Tablespaces during Import

Another unused but useful TRANSFORM option is the PCTSPACE parameter. When you specify a positive value for this parameter between 1 and 100, the value is used as a percentage multiplier for data files and extent allocations.

For example, the following creates each tablespace at 30 percent of its original size:

```
FULL=Y  
TRANSFORM=PCTSPACE:30
```

The percentage multiplier is also applied to the extent allocations.

Consolidating Multiple Tablespaces

User data warehouse (DWH) owns several tables and indexes and materialized views. The user DWH has objects on more than 40 tablespaces in the database. It's been a few years since DWH user objects have been reorganized, and there has been storage reduction and some performance gain when the objects are reorganized because most tables involve large amounts of INSERT and DELETE operations daily.

Say, for example, you are performing a planned reorg. To keep the tablespace management simple, you're going to consolidate all tablespaces into two tablespaces: DWH\_DATA for table and materialized views, and DWH\_INDEX for indexes. Since this database got upgraded from an older version, use of the LONG datatype is widespread and hence ALTER TABLE MOVE is not an appropriate option. Therefore, you'll use Data Pump as follows:

1. Export the DWH schema.
2. Use the LOGTIME parameter to add a timestamp to the output messages and log file. This is very useful for timing the different stages of export or import.
3. Use the METRICS parameter to show the time taken to export or import a category of objects:

```
SCHEMAS=DWH  
LOGTIME=ALL  
METRICS=YES  
PARALLEL=8  
DUMPFILE=dwh_exp_%U.dmp  
DIRECTORY=SCRATCH_AREA  
FILESIZE=4GB  
STATUS=180  
EXCLUDE=STATISTICS
```

4. Capture the grants given to the DWH schema by other users, including SYS/SYSTEM.
5. To be on the safe side, and to bring back any missing privileges or objects,

perform another full export of the database without any data.

6. Use CONTENT=METADATA\_ONLY to tell Data Pump to not export any data, but only the data definition language (DDL) statements:

```
FULL=Y  
CONTENT=METADATA_ONLY  
LOGTIME=ALL  
METRICS=YES  
DUMPFILE=full_exp.dmp  
DIRECTORY=SCRATCH_AREA  
STATUS=180
```

7. Drop the DWH schema and its objects, and drop all the tablespaces where DWH had objects. When dropping user DWH, the public synonyms will remain in the database, so no need to do anything with public synonyms.
8. Create new tablespaces DWH\_DATA and DWH\_INDEX. Create a DWH user, with default tablespace DWH\_DATA.
9. Grant DWH privileges on other user objects including SYS/SYSTEM.
10. Perform an import of the DWH schema. The default tablespace for the DWH user is DWH\_DATA, so by default objects will go to that tablespace. Do the import in two parts:

[Click here to view code image](#)

```
SCHEMAS=DWH  
LOGTIME=ALL  
METRICS=YES  
PARALLEL=8  
DUMPFILE=dwh_exp_%U.dmp  
DIRECTORY=SCRATCH_AREA  
STATUS=180  
EXCLUDE=INDEX  
TRANSFORM=SEGMENT_ATTRIBUTES:N
```

11. There are two methods to get the indexes in DWH\_INDEX tablespace. Use the SQLFILE option to generate the index creation statements, and replace the tablespace clause with DWH\_INDEX tablespace in the dwh\_index.sql file and run the file:

```
SCHEMAS=DWH  
DUMPFILE=dwh_exp_%U.dmp  
DIRECTORY=SCRATCH_AREA  
INCLUDE=INDEX  
TRANSFORM=STORAGE:N  
SQLFILE=dwh_index.sql
```

Another method is to set the default tablespace of the user DWH to DWH\_INDEX and perform the import as follows:

[Click here to view code image](#)

```
SCHEMAS=DWH
LOGTIME=ALL
METRICS=YES
PARALLEL=8
DUMPFILE=dwh_exp_%U.dmp
DIRECTORY=SCRATCH_AREA
INCLUDE=INDEX
TRANSFORM=SEGMENT_ATTRIBUTES:N
```

12. Recompile all invalid objects by using ?/rdbms/admin/utlrp.sql.

13. Gather statistics on the DWH schema:

[Click here to view code image](#)

```
DBMS_STATS.GATHER_SCHEMA_STATS('DWH');
```

Using PL/SQL API with Data Pump

So far in this chapter, we have seen only the Data Pump client tools expdp and impdp used. The discussion will be incomplete if we do not talk about using SQL\*Plus (PL/SQL API) to perform export and import operations. The expdp and impdp client tools use the DBMS\_DATAPUMP package for all the operations. If you can use this package directly from the database, it gives you more flexibility and endless options. Refer to PL/SQL Reference Guide for complete documentation on DBMS\_DATAPUMP and the procedures and functions available. For a simple export, you would use the following:

- **OPEN:** Define an export or import job.
- **ADD\_FILE:** Define the dump file name and log file name.
- **METADATA\_FILTER:** Provide the exclude or include clauses.
- **START\_JOB:** Kick off the export or import job.
- **WAIT\_FOR\_JOB:** Wait until the job completes.
- **DETACH:** Close the job.

[Listing 13.1](#) utilizes these subprograms and demonstrates how to export tables belonging to a schema.

Listing 13.1 Exporting a Schema Using SQL\*Plus

[Click here to view code image](#)

```
CREATE OR REPLACE PROCEDURE XX_BACKUP_TABLES (
    schema_name VARCHAR2 DEFAULT USER)
IS
    /***** NAME : XX_BACKUP_TABLES *****/
```

```

PURPOSE:      Backup tables under schema
***** */

BEGIN
-- Export all the tables belonging to schema to file system

DECLARE
    dpumpfile    NUMBER;
    dpumpstatus   VARCHAR2 (200);
    tempvar       VARCHAR2 (200);
BEGIN
    EXECUTE IMMEDIATE
        'CREATE OR REPLACE DIRECTORY BACKUP_DIR AS
 ''/u01/app/oracle/expdp'''';

BEGIN
    -- verify if the Data Pump job table exist.
    -- drop if exist.
    SELECT table_name
        INTO tempvar
        FROM user_tables
    WHERE table_name = 'BACKUP_TABLE_EXP';

    EXECUTE IMMEDIATE 'DROP TABLE BACKUP_TABLE_EXP';
EXCEPTION
    WHEN NO_DATA_FOUND
    THEN
        NULL;
END;

-- define job
dpumpfile :=
    DBMS_DATAPUMP.open (OPERATION      => 'EXPORT',
                         JOB_MODE       => 'SCHEMA',
                         JOB_NAME       => 'BACKUP_TABLE_EXP');

-- add dump file name
DBMS_DATAPUMP.add_file (
    HANDLE      => dpumpfile,
    FILENAME    => 'BACKUP_tabs.dmp',
    DIRECTORY   => 'BACKUP_DIR',
    FILETYPE    => DBMS_DATAPUMP.KU$FILE_TYPE_DUMP_FILE,
    REUSEFILE   => 1);

-- add log file name
DBMS_DATAPUMP.add_file (
    HANDLE      => dpumpfile,
    FILENAME    => 'BACKUP_tabs.log',
    DIRECTORY   => 'BACKUP_DIR',
    FILETYPE    => DBMS_DATAPUMP.KU$FILE_TYPE_LOG_FILE);

```

```

-- add filters to export only one schema
DBMS_DATAPUMP.metadata_filter (HANDLE    => dpumpfile,
                                NAME      => 'SCHEMA_LIST',
                                VALUE     => schema_name);

-- start the export job
DBMS_DATAPUMP.start_job (HANDLE => dpumpfile);

-- wait until the job completes
DBMS_DATAPUMP.wait_for_job (HANDLE => dpumpfile, JOB_STATE =>
dpumpstatus);
-- close the job
DBMS_DATAPUMP.detach (HANDLE => dpumpfile);
EXCEPTION
WHEN OTHERS
THEN
    DBMS_DATAPUMP.detach (HANDLE => dpumpfile);
    RAISE;
END;

```

```

END XX_BACKUP_TABLES;
/

```

Monitoring and Altering Resources

Since Data Pump jobs are executed at the database server by server-side processes, you can exit out of the terminal where you started the export or import. After the job is initiated, the export or import job status is displayed on the screen, and the session is occupied until the job is complete. You may press ^C to exit out of the job status logging mode and enter interactive mode.

In interactive mode, several commands are available to control the job:

- Additional dump files may be added to the export job by using the ADD\_FILE command, and dump file size for newer files can be adjusted using the FILESIZE command.
- The STATUS command shows a detailed status of the job, with details on what each worker job is doing. The STOP\_JOB command pauses the job for a later restart, whereas KILL\_JOB terminates the job completely. The PARALLEL command is used to increase or decrease the number of worker processes.
- To get back to the logging mode, use the command CONTINUE\_CLIENT, and if you are ready to exit out of the job status mode to the OS command prompt, use EXIT\_CLIENT command. Though you exited out of the job session, the job continues to run. Maybe it is the end of your shift and you want to gracefully exit out of the logging mode of export job: press ^C and then use the EXIT\_CLIENT command.

- After you reach home and if you want to check the status of the job, you have to attach to the job using the ATTACH parameter of expdp or impdp. If you do not remember the job name, it can be obtained from DBA\_DATAPUMP\_JOBS. To attach to a running or paused job, you have to connect to Data Pump using the same credentials used to start the job.

DBA\_DATAPUMP\_SESSIONS view shows all data pump sessions connected to the database.

Improving Performance

While hardware resources, a faster disk system, and proper setup improve database performance, they will also help Data Pump jobs. Additionally, there are certain parameters you can consider to further boost performance:

- As with any database operation, Data Pump also benefits from accurate statistics available in the database for dictionary objects. Collect DBMS\_STATS.GATHER\_DICTIONARY\_STATS before a Data Pump job for better performance if dictionary statistics were not collected recently.
- If you are using Data Pump import on Oracle Database 12c, consider using the option to not generate archive logs:

[Click here to view code image](#)

```
TRANSFORM=DISABLE_ARCHIVE_LOGGING:Y
```

Thus, the objects are created by the Data Pump in NOLOGGING mode and are switched to their appropriate LOGGING or NOLOGGING mode once import completes. If your database is running in FORCE LOGGING mode, this Data Pump NOLOGGING will have no impact.

- Parallelizing Data Pump jobs using the PARALLEL parameter improves performance, but there are some objects with which Data Pump does not do parallel operation—for example, creating indexes. So when importing a large database, it is better to generate the index creation statements using the SQLFILE option, and exclude indexes during the table import. The PL/SQL objects, views, and several other objects are created after the table import while doing a schema or full database import. You may let the full import without indexes run, and as soon as the table import portion is complete, you can kick off the index creation script. If you have a large, temporary tablespace and PGA area, you could split the index creation to several scripts and run them in parallel in multiple sessions.
- When performing import with a large number of tables and indexes, such as an Oracle E-Business Suite (EBS) database, it takes a while to perform import of statistics. Most of the time, you end up collecting statistics after an import, so it is better to exclude statistics from export/import altogether. You can exclude statistics in export (you may also do so during import):

```
EXCLUDE=STATISTICS
```

- Usually, import jobs involve creating many indexes and thus have a need for sort operations. Having PGA of adequate size or a little larger will help with Data Pump imports.
- If your database export includes a table with a large amount of BLOB data, it is better to exclude this table from the full export (for example, on an EBS database, the FND\_LOBS table), and perform a separate export. Even if you use the PARALLEL parameter for the BLOB table export, Data Pump is going to use only one process. Export the BLOB table as smaller chunks, performing multiple export jobs on the same table—be sure to restrict the rows in each export. For example, if you have 500 rows in the table, and you want to split the table into 50 parallel export sessions, you need to restrict each export to 10 rows each. Use the QUERY parameter to filter the rows. When importing using these dump files, make sure you use the TABLE\_EXISTS\_ACTION=APPEND option.

Upgrading Databases

Though Data Pump is not considered a primary tool for database upgrades, there are situations in which database upgrade using Data Pump is suitable:

- If you are upgrading from a database version that does not support direct upgrade, it may be easier to perform an export/import upgrade rather than going through two or more direct upgrades. For example, to upgrade to Oracle Database 12.0.2, the database has to be version 10.2.0.5, 11.1.0.7, or 11.2.0.2+. If the database isn't any of these versions, export/import upgrade is preferable. Again, the decision depends on the size of the database. One advantage of using the export/import method is that the source database remains intact, so if something did not work the way it was supposed to, rolling back to an old environment is a snap.
- Sometimes it is necessary to convert the character set of a database when you are planning to store multiple languages or need to store 16-byte characters. Export/import is proven to be the most efficient and worry-free method for character set conversion. If you are considering platform migration, instead of migrating the database to a new platform, you can create a new database and perform export/import. This is a common requirement if you are moving from or to Linux.
- If you archived or purged large amounts of data, use the upgrade/migration opportunity to reduce the size of the database.
- In Oracle database 12c, use the multitenant architecture by combining multiple Oracle 10g and 11g databases.

If the database you are upgrading from is older than Oracle 10g, then Data Pump is not available. You will have to use the legacy exp/imp tool. The good news is that the exp/imp tools are still supported in Oracle Database 12c.

When you are exporting from a higher version of database to import to a lower version of database, include the VERSION parameter so that only objects and features compatible with the target version are exported.

Detailed upgrade methods and options are beyond the scope of this chapter. See [Chapter 14](#), “[Strategies for Migrating Data Quickly between Databases](#),” to review the various migration methods available, which can be used to upgrade the database as well.

Summary

In this chapter, you learned the power of Data Pump and its flexibility to filter data and objects. You also learned how to use various combinations of export and import parameters to do various day-to-day and not-so-common DBA tasks.

There are various modes to export data and objects. Although the `FULL` mode is used to export the entire database, appropriate `INCLUDE` or `EXCLUDE` clauses can be used to filter the objects to export only what you need. Having the flexibility to use queries for metadata filtering is very powerful, enabling you to export any object, including public synonyms on a filtered object list.

The content of the export dump file can be verified by using the `SQLFILE` option, though this option is really meant to generate SQL commands out of the dump file.

Exporting subsets of data can be accomplished by using the `QUERY` parameter or by using the Oracle Database 12c feature `VIEWS_AS_TABLES`.

During import, many object properties can be changed. You can change the owner of the object, change the tablespace, merge partitions, adjust storage properties, and much more.

You also learned to consolidate multiple tablespace objects into a few, how to improve Data Pump performance, and how to use the `DBMS_DATAPUMP` API to code PL/SQL program to perform export and import.

The export/import method is used not only for data and object migration between databases but also for upgrading databases.

14. Strategies for Migrating Data Quickly between Databases

Migrating data that has been stored in an existing source database to a new destination database is probably one of the most nerve-wracking experiences any Oracle DBA will encounter during her career. (Granted, patching a database is no less tedious; but if it's been done correctly, it's possible to rewind a database from the results of a failed patching opportunity, and that's not always the case for a failed data migration attempt.) The good news is that Oracle offers numerous utilities and methodologies to successfully migrate all or part of an Oracle database's data to a new destination with virtually no service interruptions, but the bad news is that there are so many different methods that it can be difficult to decide exactly which migration path is the most appropriate in a scenario.

This chapter therefore attempts to answer various frequently asked questions, including the following:

- Why might you consider moving all or part of your database's data between the original source database and a different destination database?
- How do you perform the migration between source and destination databases with a keen regard to the service level agreements (SLAs) of the application workloads affected by the migration?
- Why is *Transportable Tablespace Sets* (TTS) probably your best choice for the majority of situations to migrate data between source and destination databases?
- When would you consider alternatives to TTS such as Data Pump Export and Import, Oracle Streams, Oracle GoldenGate, and Recovery Manager to migrate data?
- What do the newest features of Oracle Database 12c portend for migration of data between source and destination databases, especially during upgrades from pre-12c databases?

Why Bother Migrating?

While there's no doubt that data migration can be a tedious process to plan and execute, there are also some excellent reasons to perform a data migration.

One of the most obvious reasons for data migration is to upgrade an existing Oracle database to a later (perhaps the latest) Oracle database release. While it's certainly possible to perform either an in-place or out-of-place database upgrade, there may be severe restrictions on the amount of time that a database may remain in nontransactional or read-only mode as the upgrade completes.

Also, IT organizations periodically make strategic decisions to move to a new storage platform—say, from one vendor's storage area network (SAN) to another—and the

storage administrators must then present their DBA colleagues with an ultimatum: *Migrate or perish!* In this situation, an Oracle DBA will have little bargaining power because often this mandate is driven by a need to economize on storage costs, improve the reliability of existing storage components, or possibly even make a move to reduce the human capital costs of storage administration.

Finally, many IT organizations in the middle of the 2010 decade have been forced to confront the licensing costs of having multiple Oracle databases spread across dozens or hundreds of dedicated individual servers and SANs. Organizations are therefore slowly but surely moving away from this “siloed” approach, and many have begun adopting a strategy of *consolidation*—that is, hosting multiple databases on an extremely robust enterprise computing system such as an Oracle Exadata Database Machine. Since Exadata is at its core a little-endian operating system, this transition often requires not just the migration of databases and their data to the new environment but also the conversion of data from big-endian operating systems such as Solaris and HP-UNIX.

Determining the Best Strategy

Ironically, the most important consideration when contemplating a database migration project has *nothing* to do with the database itself or even the data stored in the database. The crucial driving factor that determines the best migration strategy is the SLAs for the application workloads that are accessing the data to be migrated within the source database—what can be thought of as an application’s *window of inopportunity*. Understanding the expected SLA for an application workload is the key to the success of any migration, and there are several subfactors to be considered as well.

Real-Time versus Near Real-Time Migration

A mission-critical application may still be able to tolerate a reasonable amount of downtime while the migration completes. If downtime is not tolerable at all, then the only alternative is to consider employing the methods that initially transfer data between the source and destination databases and then keep newly added or modified data in synch between source and destination until it’s time to effect a cutover to the destination database.

Read-Only Tolerance

Equally important, some applications can tolerate at least some and perhaps even considerable time during which the application can only read (but not modify) the data. Real-time systems such as electronic trading systems, banking transactional financial applications, and traffic control applications are obvious exceptions to this rule; however, consider an order entry system that operates only at peak time for a relatively narrow duration during weekdays. It may be possible to ask end users to record new orders via alternative electronic or manual methods and then simply have them rekey the transactions just a few minutes later, after all data has been successfully migrated and validated for completely successful transference.

Reversibility

Finally, it's important to consider a fallback plan in case the data migration effort must be aborted shortly after switchover to the new destination database. In many cases, this is simply due to a failure of forethought while planning the data migration project. For example:

- What if data migration accidentally caused unexpected data corruption?
- What if transaction data has been captured successfully at the source, but the delivery of that changed data failed at the destination and the failure went undetected before the switchover?
- What if unexpected character-set conversion issues are encountered because insufficient test data was available during final system testing?

It's therefore crucial to imagine the unimaginable, draw up plans for a "go/no go" decision point during the migration, and at least consider the likelihood of having to revert to the original production source.

Considering What Data to Migrate

When contemplating what to migrate, it's also helpful to consider just how much historical data needs to be migrated and how much of it will be needed immediately after the completion of the data migration process.

One of the most valuable questions to ask during a migration planning process is, Do we need all this data? Many times, application users expect their IT team to tell them when they should consider archiving at least some of their data. The good news here is that data migration is the perfect time to ask this question and perhaps complete the purging process that application data owners may have been contemplating for a long time but were afraid to ask their Oracle DBA for assistance or advice.

Even if the application data owners are thoroughly convinced that all of their data is absolutely golden and must be migrated in one fell swoop, pressing this question a bit more strenuously may force them to admit that at least some data is outmoded and perhaps even essentially useless. Once the data owners realize this, it's important to reassure them that they are making the right choice to eliminate at least some of their data. And if the Oracle DBA is fortunate enough to be migrating application data to an Oracle 12c database environment, it's important to discuss the new *Information Lifecycle Management* (ILM) features of that release and explain how Oracle 12c's *Automatic Data Optimization* (ADO) and *In-Database Archiving* (IDA) might work to their advantage.

Note

Remember that ILM, ADO, and IDA features do require licensing the Oracle Database 12c Advanced Compression option.

Even if the application data owners insist that all their data must be moved during the

initial migration, it's quite possible that not all the historical data in the source database will be immediately necessary to conduct business on the destination database. For example, recent sales orders from a few days, weeks, months, or even years may still be quite valuable, but will application users need to actively search for 5-year-old data just moments after the initial migration is complete? Since it's improbable that such a query will need to be answered on day one, that may limit the scope of the initial data migration to only a few million rows; then, while the application is accessing the most recent data, the remainder of the historical data can be migrated during a later off-peak period.

Data Migration Methods

Once the window of inopportunity—the SLA of the application(s) accessing the database to be migrated—has been identified, it is relatively simple to choose an appropriate migration method. The migration methodologies discussed in the remainder of this chapter are therefore divided between those that permit the capture of transaction activity—which allows application users to continue to modify data in the source system without any interruptions during the migration—and those that require the source database to be either partially or completely quiesced until the migration to the destination database is complete.

Transactional Capture Migration Methods

A database that supports applications whose data simply must remain accessible until the very last second before the applications are repointed to the new destination database requires special concern during data migration. Migration methods for these databases must leverage a tool or technique that permits data to flow between source and destination databases while transactions continue on the source database until the very last second. [Table 14.1](#) summarizes these migration methods.

| Migration Method | Advantages | Difficulties | Impact on SLAs | Available as of Release |
|--------------------------|---|---|----------------|-------------------------|
| Logical standby database | <ul style="list-style-type: none"> Simple to set up Data can be synchronized in real time Limited support for heterogeneous database transformation | <ul style="list-style-type: none"> Data Guard must be licensed Logical standby database becomes the new primary database after failover Switchback may be impossible Limited datatype support | Minimal | 10.2 |
| Oracle Streams | Available with Oracle EE | <ul style="list-style-type: none"> Difficult to implement Notoriously difficult to tune performance | Minimal | 10.2 |
| Oracle GoldenGate (OGG) | <ul style="list-style-type: none"> Extremely simple implementation Data can be synchronized in real time Extensive datatype support Support of data transfer for wide variety of database sources (including SQL Server, Sybase, Teradata, and DB2) | <ul style="list-style-type: none"> Requires GoldenGate licensing Not an inexpensive option! | Minimal | 10.2 |

Table 14.1 Transactional Migration Methods

Logical Standby Database

When transactional integrity must be preserved between the source and destination systems, a simple method involves using Oracle Data Guard to create a physical standby database on the desired destination platform:

1. That physical standby database can then be transformed into a logical standby database.
2. The destination database can then continue to receive transactional data manipulation language (DML) from the source database via SQL Apply data delivery methods.
3. Once the data migration is complete, SQL Apply is halted at the source database, and the destination database is failed over to.

4. Finally, all application workloads are simply pointed to the new destination database.

The use of database services is strongly recommended to ease the transition between source and destination systems. Note that if database services have been registered with the Oracle Data Guard Broker, then a failover operation will automatically point application workloads to the destination database.

Oracle Streams

Oracle Streams—essentially, database replication using LogMiner techniques—is another option that is included in the licensing costs for Oracle Database Enterprise Edition. Streams does support almost all Oracle datatypes, but it is notoriously complex to set up, and many Oracle DBAs eschew this method in favor of other technologies because it's difficult to tune its performance effectively. However, as a one-time mechanism for migrating data between source and destination databases, it may be sufficient for reasonably sized data migration.

Note

As of Oracle Database 12c, Oracle Streams has been deprecated in favor of Oracle XStreams, which is based on Oracle GoldenGate technology. For more information, see MOS Note 1644090.1, “12c streams and change data capture (CDC) support.”

Oracle GoldenGate

Oracle GoldenGate (OGG) is perhaps the easiest way to migrate data between source and destination databases while also keeping that data synchronized between both the databases.

An in-depth discussion of the myriad features of OGG would fill a book by itself, but in essence, OGG uses two distinct sets of processes to perform data migration:

- EXTRACT processes are responsible for *change data capture*. They mine the transaction logs of the source database for committed transactions, transmit those changes across the network to the destination system, and write the changes in a proprietary format to a series of *trail files*.
- REPLICAT processes are responsible for *change data delivery*. They read the trail files on the destination system and then apply those changes directly to the destination database.

OGG offers several methods for initially populating data between source and destination databases, and it supports transferring of data between dozens of commonly used relational database management systems (RDBMSs), including DB2, SQL Server, Sybase, and Teradata. OGG also offers extreme flexibility for *filtering* of data, robust functions for *modifying* the data on the fly for loading into different data structures and tables, and even the capability to replicate data bidirectionally between two systems.

The list price for OGG licensing can sometimes appear to be quite daunting, but a savvy Oracle DBA can often negotiate a term license for OGG just for the duration of the migration, especially if the migration is from a non-Oracle database to an Oracle database environment.

Nontransactional Migration Methods

These Recovery Manager (RMAN)-based migration methods require that the application workloads are prevented from applying changes to the source database for some period of time—usually determined by how long it will take to complete the final migration of data between source and destination database—to insure against the loss of any transactional data. A brief summary of these migration methods is provided in [Table 14.2](#).

| Migration Method | Benefits | Drawbacks | Impact on SLAs | Available as of Release |
|---|---|---|-------------------|-------------------------|
| DUPPLICATE DATABASE
(database cloning) | <p>Same command set as DUPLICATE DATABASE</p> <p>Enables migration of data between non-Automatic Storage Management (ASM) to ASM storage (and vice versa)</p> <p>Selected tablespace sets can be migrated</p> | Successful database migration requires careful attention to prerequisites | Short to moderate | 8.1.7 |
| Physical standby database | <p>Simple to establish</p> <p>Limited data migration between heterogeneous databases</p> | Data Guard licensing required | Short | 10.2 |
| Transportable database (TDB) | <p>Easiest method to move an entire database from source to destination</p> | <p>Entire source database must remain in read-only mode while data is transferred to destination</p> <p>Endian conversion not supported</p> | Moderate to long | 10.2 |

| | | | | |
|---|---|--|-------------------|----------|
| Transportable tablespaces (TTS) | <p>Most flexible method for data migration between source and destination databases</p> <p>Allows conversion between little- and big-endian platforms at either source or destination</p> <p>Supports all source database versions of 10.2.0.1+</p> | <p>TTS referential integrity required</p> <p>Entire source tablespace set has to stay in read-only mode on source database during its transferal to destination</p> | Short to moderate | 10.2 |
| Cross-platform transportable tablespaces (XTTS) | <p>Scripted transfer of database metadata and data via PERL scripts</p> <p>Tablespaces must remain in read/write mode until the very last transfer operation</p> | <p>TTS referential integrity required</p> <p>Final transfer operation requires TTSs to be open in read-only mode</p> | Short | 11.2.0.3 |
| Cross-platform transport (CPT) | <p>TTS data and metadata captured and loaded in single call to RMAN command</p> | <p>TTS referential integrity required</p> <p>Final transfer operation requires TTSs to be open in read-only mode</p> <p>Limited to most recent database releases</p> | Short | 12.1 |

Table 14.2 Nontransactional Migration Methods

Cloning Databases with Recovery Manager

In many cases, Oracle Recovery Manager (RMAN) can be leveraged to clone the source database as a new physical database at its new destination. This technique leverages the well-known `DUPPLICATE DATABASE` command, which has grown in power and flexibility in each subsequent Oracle database release since its introduction in Oracle 8.1.7.

In essence, cloning the source database to its new destination with `DUPPLICATE DATABASE` comprises a few simple steps that are handled automatically within the command structure itself:

1. The source database's SPFILE is cloned to its new destination, and then the destination database's instance is restarted using the new SPFILE.
2. The source database's control files are cloned at their new destination in the locations specified by DB\_CREATE\_ONLINE\_LOG\_DEST, and then the new control files are MOUNTed.
3. All the required datafiles for specified tablespaces are cloned to the new destination. Note that it's possible to migrate only a limited number of tablespace sets during the initial migration to the new destination database, and other techniques—for example, Transportable Tablespaces—can be used to migrate the remaining tablespace sets from source to destination at a later time.
4. The destination database is opened, and all online redo logs are re-created at the locations specified by DB\_CREATE\_ONLINE\_LOG\_DEST.
5. TEMPFILES are re-added for all existing temporary tablespaces. (Note that prior to Oracle Database 11g Release 1, TEMPFILES had to be re-created manually after cloning was completed.)

DUPLICATE DATABASE also offers some additional advantages over other data migration methods:

- If wide network bandwidth is available between the source and destination platforms, RMAN's ability to leverage multiple parallel channels can take dramatic advantage of that bandwidth to quickly transfer data between the source and destination platforms.
- If limited storage space is available on the destination platform, RMAN can leverage image copies and backup sets already present on the source platform as the source for cloning the data.
- Starting with Oracle Database release 12.1.0.1, DUPLICATE DATABASE automatically decides whether to use either backup sets or image copies when transmitting data directly to the destination database. (Prior releases always used image copies of all datafiles, which tended to consume significant network bandwidth during cloning.)

The biggest disadvantage of using DUPLICATE DATABASE to migrate data to its new destination is that the source database cannot process any transactions while the cloning is in progress. In addition, it's important to remember that the new destination database will be assigned a new DBID after a successful cloning operation, so it will in essence be a completely *new* database.

Note

For more information on the latest features of DUPLICATE DATABASE in Oracle 12c, see MOS Note 369644.1, “Frequently asked questions about restoring or duplicating between different versions and platforms.”

Data Guard: Physical Standby Database

One of the most effective ways to quickly migrate an Oracle database from one platform to another is to leverage Oracle Data Guard to create a new physical standby database at the desired destination. The destination database will continue to receive online redo from the source database until the DBA decides that it's finally time to perform a switchover to the physical standby, which then becomes the new primary database.

Just as for logical standby databases, database services are strongly recommended for routing application connectivity in the case of a physical standby database. Because Data Guard Broker can register a database service specifically to either the primary or physical standby database, application workload activity can immediately resume against the new destination database once switchover to the new standby database is completed.

Note

It's a little-known fact that Oracle Data Guard can be used for transforming and migrating data between *heterogeneous platforms*. See MOS Note 413484.1, "Data Guard support for heterogeneous primary and physical standbys in same Data Guard configuration," for complete information.

Transportable Database

If applications can tolerate a potentially lengthy transactional downtime, or if the database to be migrated is relatively small, another option is *transportable database* (TDB). TDB has been available since Oracle release 10.2.0.1 and offers a simple, relatively painless method to migrate an entire database in one fell swoop between source and destination platforms.

Moving a database from one platform to another using TDB essentially involves the following steps:

1. The entire source database must be shut down and then reopened in read-only mode.
2. TDB is invoked, which generates a script that handles the migration of the entire database to its new destination platform. As part of that migration, a new SPFILE will be generated on the destination platform that will re-create the new database's control files and datafiles in their new proper location.
3. Image-copy backups of the source database's control file and datafiles are shipped automatically to the destination database platform.
4. TDB starts up a new database instance using the new SPFILE.
5. The database's control files are opened on the destination database, bringing the database into MOUNT mode.
6. The database accepts its datafiles in their new physical location; they are automatically renamed via the corresponding TDB script.
7. The database is opened in read/write mode on the destination platform. As part of

this final step, new online redo log groups are re-created at the locations specified by the DB\_CREATE\_ONLINE\_LOG\_DEST parameter and new TEMPFILEs are also created in their proper locations.

However, TDB does have some prerequisites that must be considered:

- First and foremost, the entire source *must* be open in read-only mode for the duration of the migration. While this may be a small concern for a read-mostly application that can tolerate a brief amount of time without transaction processing, it is anathema for a database that supports an OLTP application.
- The determining factor for how long the source database must remain in read-only mode is the time it will take to transfer image copies of all nonsystem datafiles from the source to the destination platform.
- An important consideration not to be missed is that *endian boundaries* cannot be crossed during database migration via TDB. For example, TDB cannot be used to migrate a source database on an HP-UX (big-endian) platform to a Linux (little-endian) platform.

A simple query against view V\$TRANSPORTABLE\_PLATFORM will show the list of OS platforms that Oracle Database supports for TDB operations. And when it's joined to V\$DATABASE, V\$TRANSPORTABLE\_PLATFORM, it becomes simple to verify the endianness of the current database's OS platform, as [Listings 14.1](#) and [14.2](#) show.

Listing 14.1 Determining Platform Support for Database Migration Operations

[Click here to view code image](#)

```
SET LINESIZE 80
COL endian_format      FORMAT A12          HEADING "Endian|Format"
COL platform_name      FORMAT A60          HEADING "Platform Name" WRAP
TTITLE "Platforms Currently Supported for TTS / TDB Operations|
(from
V$TRANSPORTABLE_PLATFORM)"
SELECT
    endian_format
    ,platform_name
    FROM v$transportable_platform
    ORDER BY endian_format, platform_name;
TTITLE OFF
```

Platforms Currently Supported for TTS / TDB Operations
 (from V\$TRANSPORTABLE\_PLATFORM)

| Endian Format | Platform Name |
|---------------|----------------------------|
| Big | AIX-Based Systems (64-bit) |
| Big | Apple Mac OS |

```

Big          HP-UX (64-bit)
Big          HP-UX IA (64-bit)
Big          IBM Power Based Linux
Big          IBM zSeries Based Linux
Big          Solaris[tm] OE (32-bit)
Big          Solaris[tm] OE (64-bit)
Little       Apple Mac OS (x86-64)
Little       HP IA Open VMS
Little       HP Open VMS
Little       HP Tru64 UNIX
Little       Linux IA (32-bit)
Little       Linux IA (64-bit)
Little       Linux x86 64-bit
Little       Microsoft Windows IA (32-bit)
Little       Microsoft Windows IA (64-bit)
Little       Microsoft Windows x86 64-bit
Little       Solaris Operating System (x86)
Little       Solaris Operating System (x86-64)

```

20 rows selected.

Listing 14.2 Discovering Current Platform Details

[Click here to view code image](#)

```

SET LINESIZE 60
TTITLE "Current Database Platform Endianness| (from V$DATABASE +
V$TRANSPORTABLE_PLATFORM)"
COL name           FORMAT A16          HEADING "Database Name"
COL endian_format FORMAT A12          HEADING "Endian|Format"
SELECT
    D.name
    ,TP.endian_format
  FROM
    v$transportable_platform TP
    ,v$database D
 WHERE TP.platform_name = D.platform_name;
TTITLE OFF

```

Current Database Platform Endianness
(from V\$DATABASE + V\$TRANSPORTABLE\_PLATFORM)

| Database Name | Endian Format |
|---------------|---------------|
| DB11203 | Little |

[Listing 14.3](#) shows how the DBMS\_TDB package can be used to verify if the source

database is ready for transport to its destination.

Listing 14.3 Verifying Database Transportability

[Click here to view code image](#)

```
-----
-- Using DBMS_TDB to validate if a database is ready for transport
-----
SET SERVEROUTPUT ON
DECLARE
    db_check BOOLEAN;
BEGIN

-----
-- Can this database be transported to
-- the specified destination platform?
-----
db_check :=
    DBMS_TDB.CHECK_DB(
        target_platform_name => 'Linux IA (32-bit)'
        ,skip_option => DBMS_TDB.SKIP_OFFLINE
    );
IF db_check
    THEN DBMS_OUTPUT.PUT_LINE('Database can be transferred to
destination platform.');
    ELSE DBMS_OUTPUT.PUT_LINE('Warning!!! Database CANNOT be
transported to
destination platform.');
END IF;

-----
-- Are there any directories or external objects that need to
be
-- transferred separately after these tablespace(s) have been
-- transported to the destination platform?
-----
db_check := DBMS_TDB.CHECK_EXTERNAL;
IF db_check
    THEN DBMS_OUTPUT.PUT_LINE('Database can be transferred to
destination platform.');
    ELSE DBMS_OUTPUT.PUT_LINE('Warning!!! Database CANNOT be
transported to
destination platform.');
END IF;

END;
/
```

For a complete discussion on how to leverage TDB as a database migration methodology, see MOS Note 1401921.1, “Cross-platform database migration (across same endian) using RMAN transportable database.”

Transportable Tablespace Sets

Perhaps the most flexible toolset for any database migration effort is the *Transportable Tablespace Set* (TTS) utility. Introduced in Oracle Database 10g Release 1, it offers the capability to transfer just what’s needed from a source Oracle database to its new destination database. Even better, TTS permits crossing endian boundaries during data migration, and the conversion from little-endian to big-endian format (or vice versa) can take place on either the source or destination database via the RMAN CONVERT command.

TTS, however, does have some limitations:

- TTS boundaries must be respected. For example, if a table in the TPCH\_DATA tablespace has a related index in the TPCH\_IDX tablespace, both tablespaces must be moved as a set during their migration. (Remember that the DBMS\_TTS utility can be used to determine if the tablespaces to be moved are self-inclusive.)
- While the migration takes place, the tablespace set’s tablespaces must be open in read-only mode on the source system. While this might appear to be a disadvantage, it’s actually an advantage because any other application activity against other tablespace sets continues unabated.

TTS data migration typically involves a few basic steps: first, an appropriate tablespace set in the source database (DB10205)—in this case, tablespaces TPCH\_DATA and TPCH\_IDX—is identified and verified for completeness, as shown in [Listing 14.4](#).

Listing 14.4 Verifying Tablespace Set Transportability

[Click here to view code image](#)

```
BEGIN
    DBMS_TTS.TRANSPORT_SET_CHECK(
        ts_list => 'TPCH_DATA,TPCH_IDX'
        ,incl_constraints => TRUE
        ,full_check => TRUE
    );
END;
/
SELECT * FROM transport_setViolations;
no rows selected
```

To set up for simpler transferal of datafiles between the source database (DB10205) and the selected destination database (DB12010) using DBMS\_FILE\_TRANSFER, [Listing 14.5](#) shows how to prepare database links.

Listing 14.5 Creating a Transportability Infrastructure

[Click here to view code image](#)

```
-----
-- Prepare the source database
-----

$> mkdir /home/oracle/TTS

DROP DATABASE LINK db10205;
CREATE DATABASE LINK db10205
    CONNECT TO system IDENTIFIED BY "oracle_4U"
    USING 'db10205';

CREATE OR REPLACE DIRECTORY db10205_dbf
    AS '/u02/app/oracle/oradata/db10205';
CREATE OR REPLACE DIRECTORY ttsfiles
    AS '/home/oracle/TTS';
GRANT READ, WRITE ON DIRECTORY ttsfiles TO PUBLIC;

-----
-- Prepare the destination database
-----

$> mkdir /home/oracle/TTS

CREATE OR REPLACE DIRECTORY db12010_data
    AS '+DATA/ORCL/DATAFILE';
CREATE OR REPLACE DIRECTORY ttsfiles
    AS '/home/oracle/TTS';
GRANT READ, WRITE ON DIRECTORY ttsfiles TO PUBLIC;
```

The first step in the migration process is to restrict the accessibility of the selected tablespace sets by bringing them into read-only mode on the source database, as [Listing 14.6](#) shows.

Listing 14.6 Switching Tablespaces to Read-Only Mode

[Click here to view code image](#)

```
ALTER TABLESPACE tpch_data READ ONLY;
Tablespace altered.
ALTER TABLESPACE tpch_idx READ ONLY;
Tablespace altered.
```

Data Pump Export is used to capture the metadata for all objects within the tablespace sets on the source database into a Data Pump dump set. [Listing 14.7](#) shows the Data Pump Export parameter file used to capture just the desired metadata, and [Listing 14.8](#)

shows the invocation of the Data Pump Export operation.

Listing 14.7 Data Pump Export Parameter File

[\*\*Click here to view code image\*\*](#)

```
JOB_NAME = TTS_1
DIRECTORY = TTSFILES
DUMPFILE = tts_1.dmp
LOGFILE = tts_1.log
TRANSPORT_TABLESPACES = "tpch_data,tpch_idx"
TRANSPORT_FULL_CHECK = TRUE
```

Listing 14.8 Results of Data Pump Export Invocation

[\*\*Click here to view code image\*\*](#)

```
$> expdp system/oracle_4U PARFILE=/home/oracle/TTS/TTS_1.dpectl

Export: Release 10.2.0.1.0 - 64bit Production on Friday, 16
January, 2015 18:01:30

Copyright (c) 2003, 2005, Oracle. All rights reserved.

Connected to: Oracle Database 10g Enterprise Edition Release
10.2.0.1.0 - 64bit Production
With the Partitioning, OLAP and Data Mining options
Starting "SYS"."SYS_EXPORT_TRANSPORTABLE_01":
userid="/*****@ (DESCRIPTION=(ADDRESS=(PROTOCOL=beq)
(PROGRAM=/u01/app/oracle/product
/10.2.0/db_1/bin/oracle) (ARGV0=oraclefxkl) (ARGS=\ (DESCRIPTION=\
(LOCAL=YES)\ )\ (ADDRESS=
\ (PROTOCOL=beq\)\ )\ )) (ENVS=ORACLE_SID=fxkl)) (CONNECT_DATA=
(SID=fxkl)))
AS SYSDBA" transport_tablespaces=
TPCH_DATA, TPCH_IDX dumpfile=tts_2.dmp directory=TTSFILES
logfile=tts_2.log
Processing object type TRANSPORTABLE_EXPORT/TABLE
Processing object type
TRANSPORTABLE_EXPORT/GRANT/OWNER_GRANT/OBJECT_GRANT
Processing object type TRANSPORTABLE_EXPORT/INDEX
Processing object type TRANSPORTABLE_EXPORT/CONSTRAINT/CONSTRAINT
Processing object type TRANSPORTABLE_EXPORT/INDEX_STATISTICS
Processing object type
TRANSPORTABLE_EXPORT/CONSTRAINT/REF_CONSTRAINT
Processing object type TRANSPORTABLE_EXPORT(TABLE_STATISTICS
Processing object type
TRANSPORTABLE_EXPORT/POST_INSTANCE/PLUGTS_BLK
```

Master table "SYS"."SYS\_EXPORT\_TRANSPORTABLE\_01" successfully loaded/unloaded

Image copies of all datafiles for the corresponding tablespace sets are copied from the source to the destination database using either OS utilities or Oracle's DBMS\_FILE\_TRANSFER utility, as [Listing 14.9](#) shows.

Listing 14.9 Invoking DBMS\_FILE\_TRANSFER.PUT\_FILE

[Click here to view code image](#)

```
BEGIN
    DBMS_FILE_TRANSFER.PUT_FILE(
        source_directory_object => 'DB10205_DBF'
        ,source_file_name => 'tpch_data.dbf'
        ,destination_directory_object => 'DB12010_DATA'
        ,destination_file_name => 'TPCH_DATA.DBF'
        ,destination_database => 'DB12010'
    );
    DBMS_FILE_TRANSFER.PUT_FILE(
        source_directory_object => 'DB10205_DBF'
        ,source_file_name => 'tpch_idx.dbf'
        ,destination_directory_object => 'DB12010_DATA'
        ,destination_file_name => 'TPCH_IDX.DBF'
        ,destination_database => 'DB12010'
    );
    DBMS_FILE_TRANSFER.PUT_FILE(
        source_directory_object => 'TTSFILES'
        ,source_file_name => 'tts_1.dmp'
        ,destination_directory_object => 'TTSFILES'
        ,destination_file_name => 'tts_1.dmp'
        ,destination_database => 'DB12010'
    );
END;
/
```

On the destination database, Data Pump Import is used to load the TTS metadata contained in the Data Pump dump into the destination database, thus “plugging in” the TTS to that database, as shown in [Listings 14.10](#) and [14.11](#).

Listing 14.10 Data Pump Import Parameter File

[Click here to view code image](#)

```
JOB_NAME = TTS_1
DIRECTORY = TTSFILES
DUMPFILE = tts_1.dmp
LOGFILE = tts_1.log
```

```
TRANSPORT_DATAFILES =
'+DATA/ORCL/DATAFILE/TPCH_DATA.DBF', '+DATA/ORCL/DATAFILE/TPCH_IDX.DI
```

Listing 14.11 Plugging Transported Tablespaces into Destination Database

[Click here to view code image](#)

```
$> impdp system/oracle_4U PARFILE=tts_1.dpictl

>> Results of successful transportable tablespace import at
destination
>> (from destination database's alert log)

. . .
Fri Jan 16 18:32:18 2015
DM00 started with pid=36, OS id=2269, job SYSTEM.TTS_1
Fri Jan 16 18:32:18 2014
DW00 started with pid=37, OS id=2271, wid=1, job SYSTEM.TTS_1
Plug in tablespace TPCH_IDX with datafile
  '+DATA/ORCL/DATAFILE/TPCH_IDX.DBF'
Plug in tablespace TPCH_DATA with datafile
  '+DATA/ORCL/DATAFILE/TPCH_DATA.DBF'
. . .
```

Finally, the newly migrated TTSs are brought into read/write mode on the destination database, and application workloads can now start processing against the transported tablespaces in the destination database, as [Listing 14.12](#) shows.

Listing 14.12 Switching Transported Tablespaces to Read/Write Mode

[Click here to view code image](#)

```
SQL> ALTER TABLESPACE tpch_data READ WRITE;
Tablespace altered.
SQL> ALTER TABLESPACE tpch_idx READ WRITE;
Tablespace altered.
```

Note

It's also possible to transport tablespaces using the RMAN TRANSPORT TABLESPACE command, which also leverages RMAN's *tablespace point-in-time recovery* (TSPITR) capability so that the selected tablespace sets can be transported from the source to the destination as of an earlier *system change number (SCN)*. See MOS Note 1461278.2, "Information Center: Transportable tablespaces (TTS) for Oracle databases," for detailed information on this methodology.

Cross-Platform Transportable Tablespaces

Available for Oracle 11.2.0.3 destination databases, *cross-platform transportable tablespaces* (XTTS) is an intriguing variation on the TTS methodology. XTTS permits the source database's TTSs to remain open in read/write mode for almost the entire duration of the data migration effort except for a brief outage period at the very end.

There are several similarities between TTS and XTTS:

- TTS boundaries must still be respected, and Data Pump Export is still used to capture TTS metadata on the source.
- Endian boundaries can be crossed, and conversion to different endian-ness can be requested at either the source or the destination database.
- Data Pump Import is still used to plug in TTS metadata to the destination database.

However, the main processing flow is significantly different. There are some prerequisites to accomplish, as shown in [Listings 14.13](#) and [14.14](#).

Listing 14.13 XTTS: Prerequisites to Initial Execution

[Click here to view code image](#)

```
-----  
-- On source database:  
-----  
SQL> CREATE OR REPLACE DIRECTORY ora10g_dbf  
      AS '+DATA/ORA10G/DATAFILE';  
SQL> GRANT READ, WRITE ON DIRECTORY ora10g_dbf TO PUBLIC;  
  
SQL> CREATE OR REPLACE DIRECTORY xttsfiles  
      AS '/home/oracle/XTTSFILES';  
SQL> GRANT READ, WRITE ON DIRECTORY xttsfiles TO PUBLIC;  
  
-----  
-- On destination database:  
-----  
SQL> CREATE OR REPLACE DIRECTORY ora12c_dbf  
      AS '+DATA/ORA12010/DATAFILE';  
SQL> GRANT READ, WRITE ON DIRECTORY ora12c_dbf TO PUBLIC;  
  
SQL> CREATE OR REPLACE DIRECTORY xttsfiles  
      AS '/home/oracle/XTTSFILES';  
SQL> GRANT READ, WRITE ON DIRECTORY xttsfiles TO PUBLIC;  
SQL> DROP DATABASE LINK ora10g;  
SQL> CREATE DATABASE LINK ora10g  
      CONNECT TO system IDENTIFIED BY oracle_4U  
      USING 'ORA10G';
```

Listing 14.14 XTTs: Extracting XTT Programs and Configuring xtt.properties

[Click here to view code image](#)

```
$> unzip -d /home/oracle/XTTS rman_xttconvert_1.4.zip

# Editing xtt.properties configuration file on source and
destination:

-- Do this on source and destination platforms
$> unzip -d /home/oracle/XTTS rman_xttconvert_1.4.zip
-- Edit xtt.properties file on source and destination platforms

# xtt.properties
# Parameters for Cross-Platform Transportable Tablespace (XTTS)
demonstrations
# Tablespace(s) for XTTS transport:
tablespaces=AP_DATA,AP_IDX
# Source database parameters:
platformid=2
srcdir=ORA10G_DBF
srclink=ORA10G
dfcopydir=/home/oracle/XTTSFILES
backupformat=/home/oracle/XTTSFILES
# Destination database parameters:
dstdir=ORA12c_DBF
stageondest=/home/oracle/XTTSFILES
storageondest=+DATA
backupondest=+FRA
asm_home=/u01/app/oracle/product/12.1.0/grid
asm_sid=+ASM
parallel=4
rollparallel=2
```

XTTS requires capture, transport, and restoration of an initial set of incremental level 0 image copy backups of all datafiles for each TTS between the source and destination databases, as shown in [Figure 14.1](#). Note that the endian conversion—if it is indeed required—is performed *automatically* during this step.

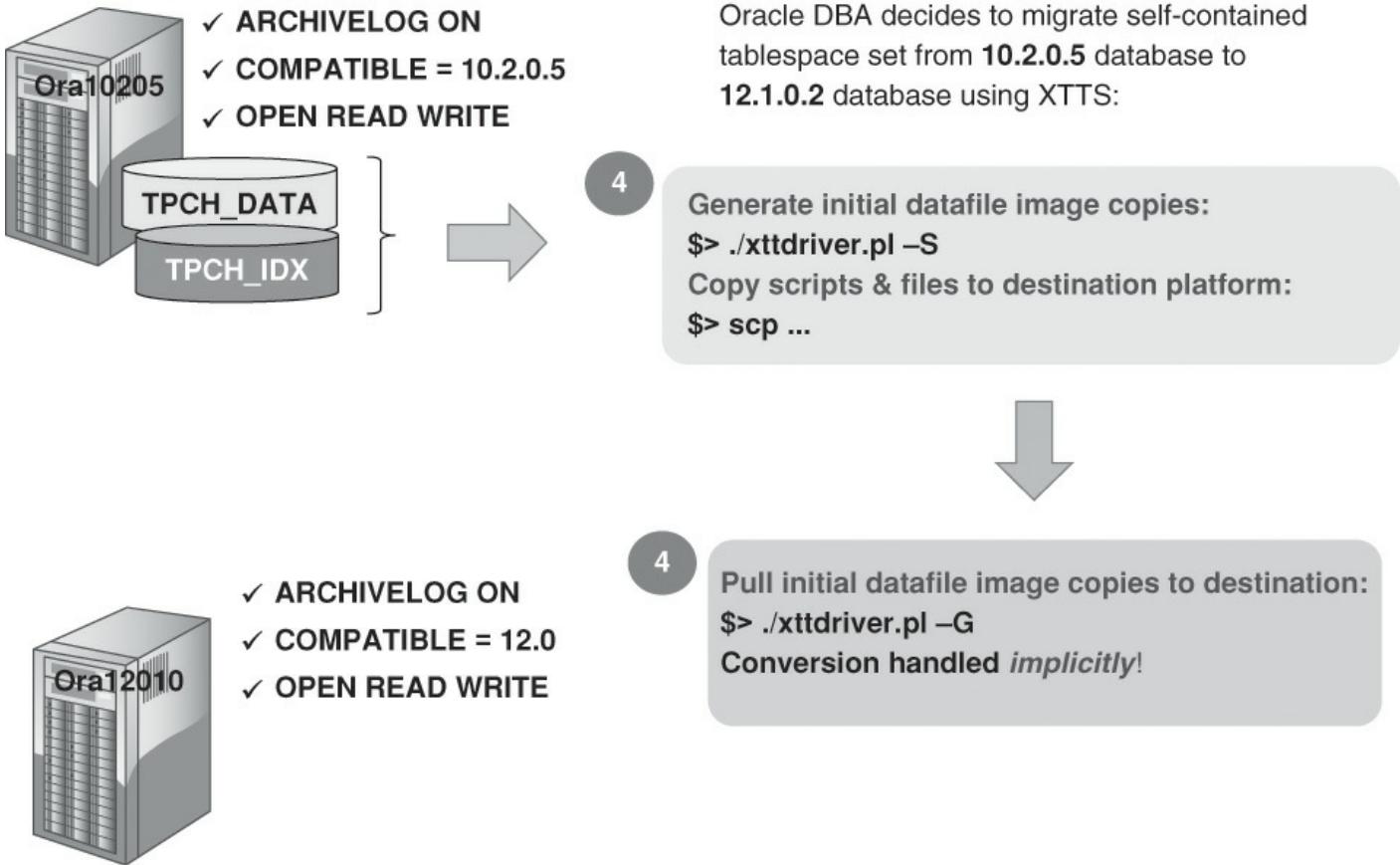
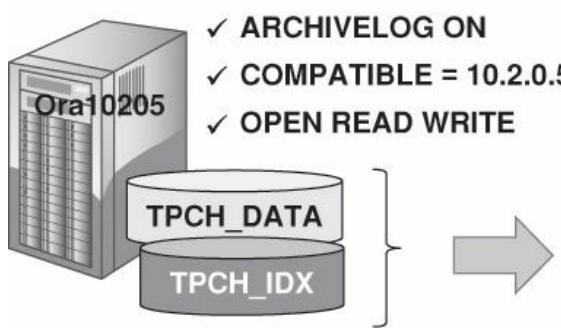


Figure 14.1 Initial extraction in cross-platform transportable tablespaces

Once the initial image copy backups have been transferred from source to destination, however, the XTTS methodology continues to take incremental level 1 backups on the source database. These backup sets are then transported to the destination database and applied against the previously transferred incremental level 0 image copies to recover them forward, as [Figure 14.2](#) shows.

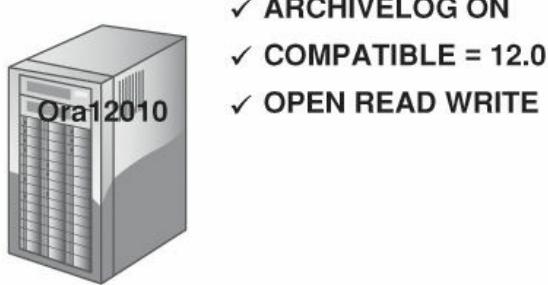


Incremental synchronization:

- Take INCREMENTAL LEVEL 1 backups against source database
- Transfer to destination database
- Use them to recover image copies forward in time

5

Generate INCREMENTAL LEVEL 1 backup sets:
\$> ./xttdriver.pl -i
Copy scripts & files to destination platform:
\$> scp ...
Capture SCN of last synchronization:
\$> ./xttdriver.pl -s



5

Use INCREMENTAL LEVEL 1 backup sets to recover datafile image copies to current SCN:
\$> ./xttdriver.pl -r

This process can be performed *multiple* times before the final synchronization

Figure 14.2 Incremental synchronization in cross-platform transportable tablespaces

When the files at the destination database are as close as desired to the source database and the DBA decides to complete the migration process, all TTSs are brought into read-only mode *just once*. This freezes any transactional activity on the source until one final incremental level 1 backup can be captured, transported to the destination, and then applied against the incremental level 0 image copies to recover them forward one last time, as shown in [Figure 14.3](#).

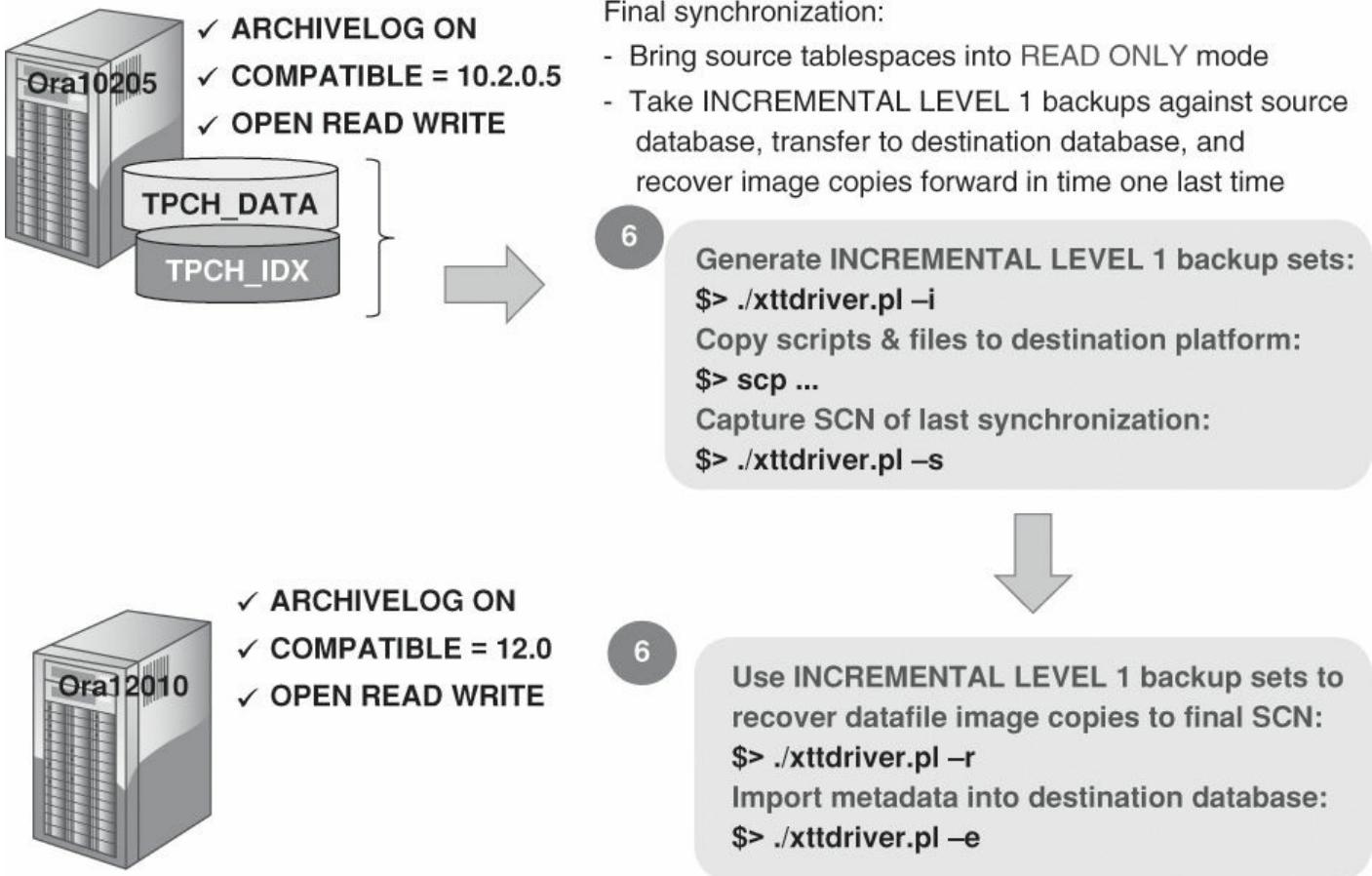


Figure 14.3 Final synchronization in cross-platform transportable tablespaces

For an excruciatingly detailed discussion of how to download and deploy XTTS for a data migration effort, see MOS Note 1389592.1, “Reduce transportable tablespace downtime using cross platform incremental backup.”

Cross-Platform Transport

New in Oracle Database Release 12.1.0.1, the *cross-platform transport* (CPT) utility method uses a combination of RMAN, DBMS\_FILE\_TRANSFER, and Data Pump to perform data migration extremely similar to the methodology that XTTS provides.

As with XTTS, CPT offers similar advantages and restrictions:

- TTS boundaries must still be respected.
- Endian boundaries can be crossed, and conversion to different endian-ness can be requested at either the source or the destination database.
- The tablespace sets selected for migration remain in read/write mode for the duration of the migration except for a short period at the end of the migration process when they must be briefly brought into read-only mode.

However, CPT also offers a much simpler methodology than XTTS, as shown in [Figure 14.4](#).

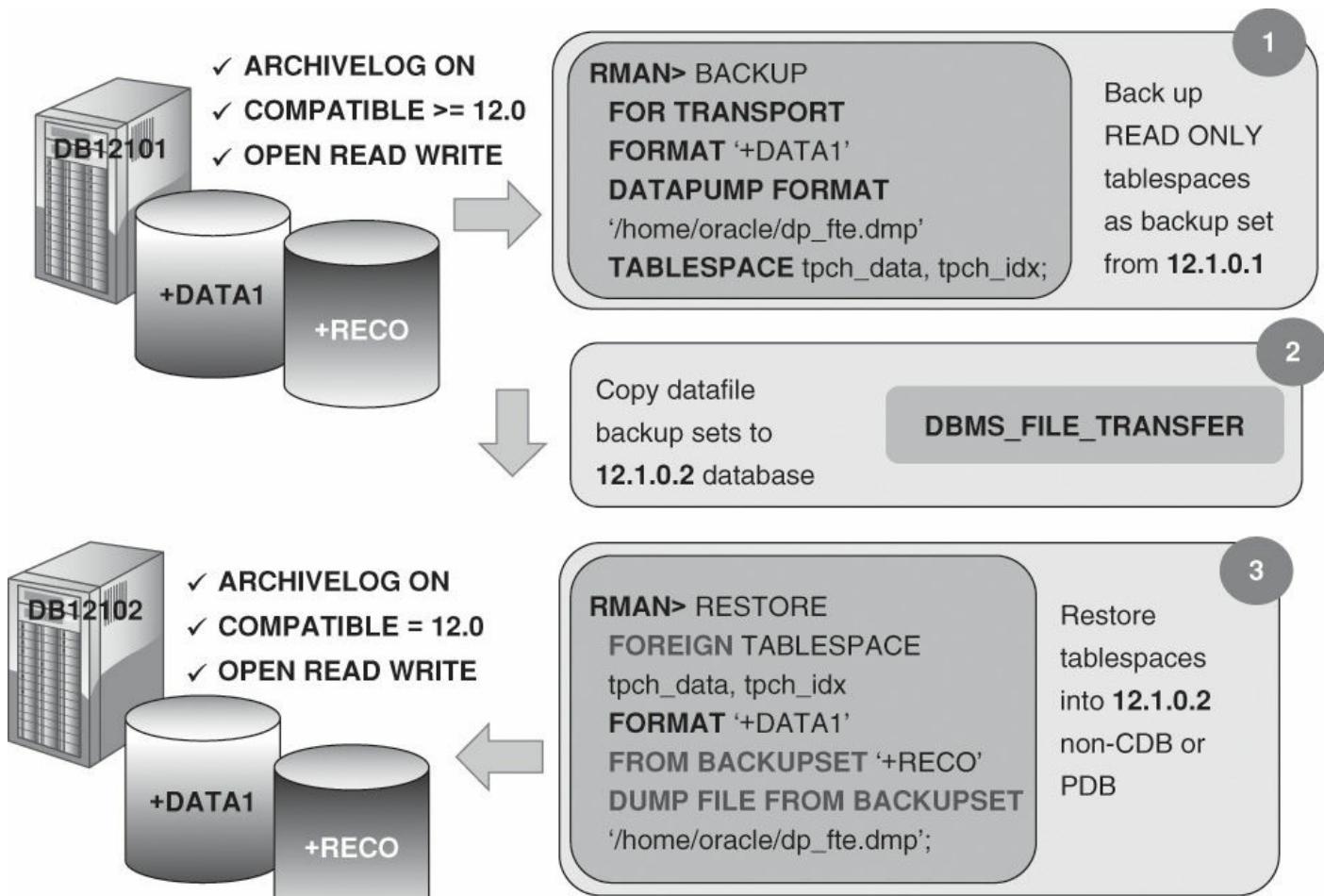


Figure 14.4 Cross-platform transport workflow

The RMAN BACKUP FOR TRANSPORT command is used to initiate the migration. Note that this command now automatically calls Data Pump Export to capture the tablespace set's metadata on the source database.

DBMS\_FILE\_TRANSFER (or any OS utility) transports all backup sets to the destination system, following which the final phase of the migration may commence.

At the destination database, the RMAN RESTORE FOREIGN command is invoked. Data Pump Import locates the specified dump set and uses it to load TTS metadata into the destination database, thus plugging in all TTSs.

Note

As of this book's publication date, CPT is available *only* for migration between source and destination databases at Oracle Database release 12.1.0.1 or higher; however, it may be back-ported to earlier releases at a future date.

Piecemeal Migration Methods

It's also possible to migrate data between source and destination databases using a more piecemeal approach that ranges from database utilities such as Data Pump Export and Import to completely manual methods, as [Table 14.3](#) summarizes.

| Migration Method | Advantages | Difficulties | Impact on SLAs | Available as of Release |
|--|---|---|-------------------|-------------------------|
| Data Pump Export and Import | <p>Extremely flexible</p> <p>Only table data may be moved during migration, with the option of creating indexes later on destination</p> <p>Data can be imported over the network if sufficient network bandwidth exists between source and destination</p> | <p>Data on source database must remain static until after all migrated objects are reloaded</p> <p>May require large amounts of disk space for Data Pump dump sets if not employing import network mode</p> | Short to long | 10.1 |
| Data Pump full transportable export/import (FTE) | <p>Single-step creation of TTS data and metadata</p> <p>Captures all objects related to source database's TTS objects in SYSTEM tablespace and carries them along to destination</p> | <p>TTS referential integrity required</p> <p>Final transfer operation requires TTSs to be open in read-only mode</p> <p>Limited to most recent database releases</p> <p>Entire source <i>tablespace set</i> has to stay in read-only mode on source database during its transfer to destination</p> | Short to moderate | 12.1 |

| | | | | |
|--------------------|--|---|-------------------|-------|
| Partition exchange | Simple to implement
Extremely fast data migration because only metadata is truly affected | Requires understanding of partition exchange methods
Flexibility varies depending on database release | Short | 8.1.7 |
| Programmed methods | As flexible as necessary, depending on application uptime requirements | Must first establish database links before transferring data between source and destination

Requires considerable knowledge of PL/SQL and related utilities for development of efficient, effective, and accurate data transferal

May not be possible to test adequately (and reproducibly) before implementing in production | Short to moderate | All |

Table 14.3 Manual Migration Methods

Data Pump Export/Import

If the application can tolerate a potentially lengthy outage, then using Oracle Data Pump Export and Import utilities may offer an excellent solution to complete a data migration.

If disk space on the source and/or destination platforms is low or unavailable, but there is relatively wide network bandwidth between both platforms, then consider performing Data Pump Import operations directly over the network from the destination database by setting the `NETWORK_LINK` parameter to an appropriate database link to the source database.

Consider using the `FLASHBACK_SCN` parameter to isolate transactions to a particular point in time during data export. However, be aware that this may require an extremely large UNDO tablespace to provide sufficient read consistency for the objects being exported.

Finally, remember that there is no way to transactionally *resynchronize* tables that have been loaded via Data Pump Import; this requires a utility like Oracle GoldenGate, Oracle Streams, or other third-party tools that employ change data capture and delivery methods.

Data Pump Full Transportable Export/Import

If the source database is at least Oracle Database release 11.2.0.3 and the destination database that's resident on Exadata is at least Oracle Database release 12.1.0.1, then Data Pump full transportable export (FTE) may be an attractive option to complete the migration between source and destination platforms. [Figure 14.5](#) shows the basic strategy behind this method.

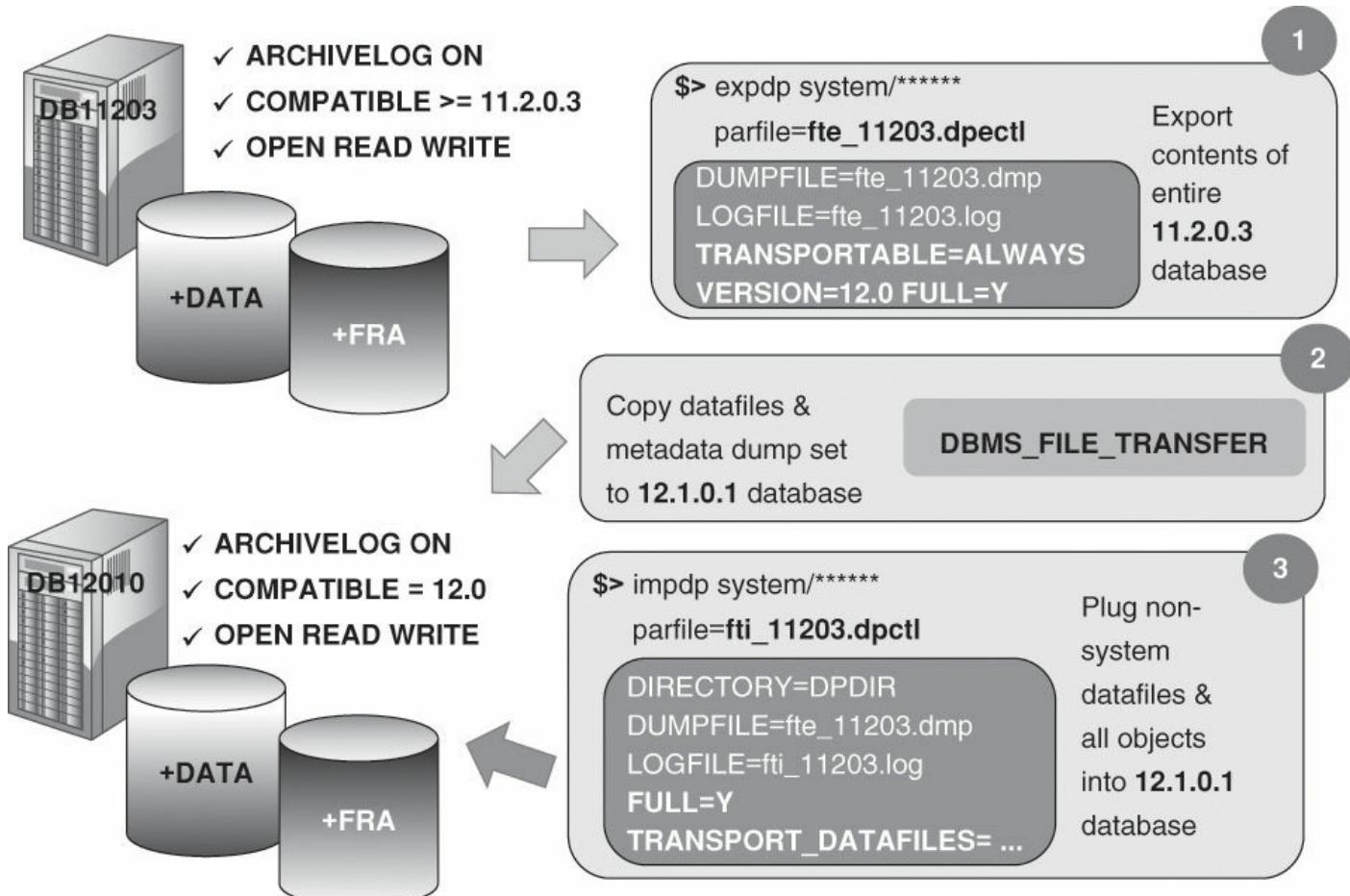


Figure 14.5 Data Pump full tablespace export workflow

First, the desired non-system tablespace sets are identified and verified for completeness, as shown in [Listing 14.15](#).

Listing 14.15 Verifying Self-Contained Tablespace Sets

[Click here to view code image](#)

```
EXEC DBMS_TTS.TRANSPORT_SET_CHECK('TPCH_DATA,TPCH_IDX',TRUE,TRUE);
PL/SQL procedure successful.
```

```
SELECT * FROM transport_setViolations;
No rows returned
```

All non-system tablespaces about to be transported are brought into read-only mode, as shown in [Listing 14.16](#).

Listing 14.16 Bringing Transportable Tablespace into Read-Only State

[Click here to view code image](#)

```
SQL> ALTER TABLESPACE tpch_data READ ONLY;
Tablespace altered.
```

```
SQL> ALTER TABLESPACE tpch_idx READ ONLY;
Tablespace altered.
```

Data Pump Export is used to capture the metadata for all objects within the tablespace sets as well as all database objects that have been stored in the source database's data dictionary (for example, packages, stored procedures) into a Data Pump dump set, as shown in [Listings 14.17](#) and [14.18](#). To execute the Data Pump Export, issue the following command:

[Click here to view code image](#)

```
$> expdp system/oracle_4U parfile=/home/oracle/fte_db11203.dpectl
```

Listing 14.17 Contents of fte\_db11203.dpectl Parameter File

```
DIRECTORY=DATA_PUMP_DIR
DUMPFILE=fte_ora11203.dmp
LOGFILE=fte_ora11203.log
FULL=Y
TRANSPORTABLE=ALWAYS
VERSION=12.0
```

Listing 14.18 Results of FTE Export

[Click here to view code image](#)

```
Export: Release 11.2.0.4.0 - Production on Fri Jan 16 20:23:34 2015
```

```
Copyright (c) 1982, 2011, Oracle and/or its affiliates. All rights reserved.
```

```
Connected to: Oracle Database 11g Enterprise Edition Release  
11.2.0.4.0 - 64bit Production
```

```
With the Partitioning, Automatic Storage Management, OLAP, Data  
Mining
```

```
and Real Application Testing options
```

```
Starting "SYSTEM"."SYS_EXPORT_FULL_01": system/*********
```

```
parfile=/home/oracle/fte_ora11203.dpectl
```

```
Estimate in progress using BLOCKS method...
```

```
Processing object type
```

```
DATABASE_EXPORT/EARLY_OPTIONS/VIEWS_AS_TABLES/TABLE_DATA
```

```
Processing object type DATABASE_EXPORT/NORMAL_OPTIONS/TABLE_DATA
```

```

Processing object type DATABASE_EXPORT/NORMAL_OPTIONS/VIEWS_AS_TABLES/TABLE_DATA
Processing object type DATABASE_EXPORT/SCHEMA/TABLE/TABLE_DATA
Total estimation using BLOCKS method: 64.93 MB
Processing object type DATABASE_EXPORT/PRE_SYSTEM_IMPCALLOUT/MARKER
...
<< many lines omitted for sake of brevity >>
...
. . . exported "SYSTEM"."REPCAT$_TEMPLATE_SITES" 0
KB      0 rows
. . . exported "SYSTEM"."REPCAT$_TEMPLATE_TARGETS" 0
KB      0 rows
. . . exported "SYSTEM"."REPCAT$_USER_AUTHORIZATIONS" 0
KB      0 rows
. . . exported "SYSTEM"."REPCAT$_USER_PARM_VALUES" 0
KB      0 rows
. . . exported "SYSTEM"."SQLPLUS_PRODUCT_PROFILE" 0
KB      0 rows
Master table "SYSTEM"."SYS_EXPORT_FULL_01" successfully
loaded/unloaded
*****
Dump file set for SYSTEM.SYS_EXPORT_FULL_01 is:
  /u01/app/oracle/admin/ora11g/dpdump/fte_db11203.dmp
*****
Datafiles required for transportable tablespace TPCH_DATA:
  +DATA/ora11g/datafile/tpch_data.258.858108115
Datafiles required for transportable tablespace TPCH_IDX:
  +DATA/ora11g/datafile/tpch_idx.257.858108117
Job "SYSTEM"."SYS_EXPORT_FULL_01" successfully completed ...

```

Next, the image copies of all datafiles for the corresponding tablespace sets are copied from the source to the destination database using either OS utilities or Oracle's DBMS\_FILE\_TRANSFER utility.

On the destination database, Data Pump Import is used to load the TTS metadata contained in the Data Pump dump set into the destination database, thus plugging the TTS into that database. The first step in this process is to create a directory object, as shown in [Listing 14.19](#).

Listing 14.19 Creating Directory Object for Pluggable Database Access to Data Pump Directory

[Click here to view code image](#)

```

CREATE OR REPLACE DIRECTORY dpdir
  AS '/u01/app/oracle/admin/cdb122/dpdump';
GRANT READ ON DIRECTORY dpdir TO PUBLIC;
GRANT WRITE ON DIRECTORY dpdir TO PUBLIC;

```

Finally, it's time to import the metadata into the DB12010 database. Data Pump Import is invoked as follows to complete the transfer of all TTS metadata as well as all database objects that have been stored in the database's data dictionary:

[Click here to view code image](#)

```
$> impdp system/oracle_4U@db12010  
parfile=/home/oracle/fti_db12010.dpictl
```

[Listing 14.20](#) shows the Data Pump Export parameter file, and [Listing 14.21](#) displays abbreviated results of the Data Pump Import process.

Listing 14.20 Contents of fti\_db12010.dpictl Parameter File

[Click here to view code image](#)

```
DIRECTORY=DPDIR  
DUMPFILE=fte_db11203.dmp  
LOGFILE=fti_db12010.log  
FULL=Y  
TRANSPORT_DATAFILES='/u01/app/oracle/oradata/db12010/tpch_data.dbf',  
'/u01/app/oracle/oradata/db12010/tpch_idx.dbf'
```

Listing 14.21 Results of FTE Import

[Click here to view code image](#)

```
Import: Release 12.1.0.1.0 - Production on Fri Jan 16 21:14:03 2015  
  
Copyright (c) 1982, 2013, Oracle and/or its affiliates. All rights reserved.  
  
Connected to: Oracle Database 12c Enterprise Edition Release  
12.1.0.1.0 - 64bit Production  
With the Partitioning, OLAP, Advanced Analytics and Real  
Application Testing options  
Master table "SYSTEM"."SYS_IMPORT_FULL_01" successfully  
loaded/unloaded  
Source timezone version is +00:00 and target timezone version is  
-07:00.  
Starting "SYSTEM"."SYS_IMPORT_FULL_01": system/********@db12010  
parfile=/home/oracle/fti_db12010.dpictl  
Processing object type DATABASE_EXPORT/PRE_SYSTEM_IMPCALLOUT/MARKER  
Processing object type  
DATABASE_EXPORT/PRE_INSTANCE_IMPCALLOUT/MARKER  
Processing object type DATABASE_EXPORT/TABLESPACE  
.  
<< many lines omitted for sake of brevity >>  
.  
.
```

Partition Migration

If the destination database is leveraging Oracle Partitioning features, then one of the fastest ways to load data into a partitioned table is to not load it at all but to use *partition exchange* mechanisms instead to exchange partitions between the two databases. Because partition exchange simply modifies the underlying metadata for the partitioned objects—essentially, plugging in a nonpartitioned table as a new partition in an existing partitioned table—it is one of the fastest and easiest ways to migrate data.

This data migration technique uses the following strategy:

1. The metadata for the tablespaces that comprise the partitioned tables and indexes are extracted via Data Pump Export.
2. The tablespace sets that comprise just the older, least frequently used partitions of the partitioned tables and their indexes are brought into read-only mode and are then transported to the destination system and placed in the appropriate storage locations. (By the way, this is also an excellent mechanism for migrating data from non-ASM to ASM storage.)
3. Using Data Pump Import, the tablespaces' metadata is plugged into the destination database to complete the data transfer, and then these new tablespaces are placed into read/write mode.

Manual Methods

Finally, what if none of these methods and utilities provides sufficient flexibility for loading data, or what if an application's window of inopportunity must be somehow reduced to satisfy that application's SLA? The only alternative may be to write custom SQL or PL/SQL code to complete data migration in a timely fashion between the source and destination databases.

These manual methods can definitely leverage database links from the destination to the source database, and they can take advantage of the implicit cursors provided by `INSERT INTO [destination table] SELECT FROM [source table]@database_link` statements or `CREATE TABLE [destination table] AS SELECT FROM [source table]@source_database_link` statements. These two methods allow the use of parallelism at both source and/or destination databases for fast data extraction and loading. If only a handful of database objects must be handled manually, this approach may actually offer exactly the required speed and flexibility that much more expensive tools may offer.

If you are fortunate enough that the DDL for the objects to be loaded match precisely between source and destination databases, consider using either Data Pump Export/Import or the `DBMS_METADATA` package to extract and then construct empty objects on the destination database before loading any data. Finally, remember to use the proper hints to invoke direct-path loading for the `INSERT INTO ... SELECT FROM SQL statement: /* +APPEND NOLOGGING PARALLEL(n) */`. (Recall that `CREATE TABLE AS SELECT [CTAS]` statements automatically leverage direct-path

loading and parallelism without any extra hints.)

My Oracle Support Reference Documents

The following My Oracle Support (MOS) documents contain valuable primary and ancillary information for additional reading and strengthening understanding of the myriad topics discussed in this chapter:

- **369644.1** : “Frequently asked questions about restoring or duplicating between different versions and platforms”
- **413484.1** : “Data Guard support for heterogeneous primary and physical standbys in same Data Guard configuration”
- **831223.1** : “Using RMAN incremental backups to update transportable tablespaces”
- **1389592.1** : “Reduce transportable tablespace downtime using cross-platform incremental backup”
- **1401921.1** : “Cross-platform database migration (across same endian) using RMAN transportable database”
- **1472959.1** : “Moving user datafiles between ASM disk groups using incrementally updated backups”
- **1644090.1** : “12c streams and change data capture (CDC) support”

Additionally, *Oracle Database Upgrade, Migration and Transformation Tips and Techniques* (Oracle Press, June 2015) by Edward Whalen and Jim Czuprynski is also a useful resource.

Summary

You learned in this chapter that you shouldn’t be afraid to ask your users if they really do need all their data at the conclusion of the initial migration. It’s very likely that the applications accessing that data can tolerate a relatively short window of inopportunity during which data is accessible in read-only mode.

Of all the tools on an Oracle DBA’s tool belt, this chapter demonstrated that Transportable Tablespace Sets (TTS) is the closest thing to a Swiss Army knife because it makes short work of migrating extremely large volumes of data between source and destination databases—especially when an endian boundary needs to be crossed.

No single tool is appropriate for all migration scenarios. In fact, it usually makes more sense to leverage multiple tools—Data Pump Export/Import for more static data, TTS for data that can tolerate short periods of read-only access, and Oracle Streams or Oracle GoldenGate for the most volatile data—to accomplish the ultimate goals of the migration.

15. Diagnosing and Recovering from TEMPFILE I/O Issues

Temporary tablespaces are often the bane of Oracle database technology: Oracle DBAs rarely think about them as a possible vector for database performance issues, but when they do cause poor performance, they tend to stick out in a crowd. This chapter briefly reviews the history of temporary tablespace features. It then discusses the benefits and drawbacks of using temporary tablespace to improve database performance. Finally, it looks at some of the symptoms of application performance when a temporary tablespace has been identified as the root cause of the performance problem.

Overview of Temporary Tablespaces

Temporary tablespaces have been available for the Oracle databases since Oracle Release 7.3. [Table 15.1](#) lists the various features of temporary tablespaces and the database release as of which they were made available.

| Feature | Available in Release | Description |
|--|----------------------|--|
| Read-only database | 8.1 | Enables databases to be opened in read-only mode for reporting purposes or to freeze transactional activity for a limited duration |
| Locally managed temporary tablespaces | 8.1 | Alleviates I/O waits caused by dictionary-managed extents |
| Temporary table-space groups (TTGs) | 10.1 | Leverages multiple temporary tablespaces for SQL statements that employ multiple degrees of parallelism |
| Separate temporary tables for global temporary tables (GTTs) | 11.2 | Permits isolation of I/O demands for GTTs to a specific temporary tablespace |
| Undo deactivation for GTTs | 12.1 | Alleviates unnecessary undo generation when populating a GTT |
| Automated statistics gathering for GTTs | 12.1 | Allows statistics to be gathered for different iterations of a GTT on a per-session basis or duplicate representative statistics to all iterations of the same GTT |

Table 15.1 Temporary Tablespace Features

Read-Only Databases

A database that has been opened in read-only mode is the perfect host for a data warehouse or data mart that is blocked from accepting any data manipulation language (DML) modifications against the data segments in permanent storage. When the *program global area* (PGA)—where all SQL statements for a user’s session essentially do their

work, whether that work involves sorting, aggregation, or hash joins—runs out of collective memory for all active user sessions, the only other place besides memory to complete that work is within a temporary segment housed in a temporary tablespace. However, if there were no alternative writeable location once all free memory in the PGA had been completely exhausted during query processing, then it would be *impossible* to process an application workload against a read-only database.

Locally Managed Temporary Tablespaces

Locally managed temporary tablespaces (LMTTs) were introduced in Oracle Database Release 8.1 in concert with the general release of *locally managed tablespaces* (LMTs). Just as with permanent tablespaces, LMTs provide a much more efficient space management mechanism for a tablespace's underlying physical database files than the predecessor file management method called *dictionary-managed tablespaces*. Since an LMT is essentially self-aware in terms of from where the next file extent can be allocated, it effectively eliminated the recursive SQL (and especially the corresponding undo and redo entries) that were required to update the data dictionary tables that stored the information. Since an LMTT could need to expand dramatically any time that the PGA might be exhausted, LMTs thus dramatically increased I/O space management performance—in many cases, by one full order of magnitude (10X).

Temporary Tablespace Groups

It's perfectly acceptable to create *multiple* temporary tablespaces for a single database, which allows an Oracle DBA to assign a temporary tablespace to particular application users on the basis of how those users' sessions are expected to leverage temporary tablespaces. The use of multiple temporary tablespaces also tends to limit or even eliminate file contention between temporary segments. However, this technique may not necessarily help when an application user's queries leverage large degrees of parallelism.

Introduced in Oracle 10g Release 1, *temporary tablespace groups* (TTGs) are designed to help parallel queries benefit from multiple temporary tablespaces when an application workload can benefit from employing multiple degrees of parallelism. It's certainly possible to extend the size of a temporary tablespace by either adding more space to its TEMPFILE or adding new TEMPFILES. However, each set of parallel server processes are assigned to a single temporary *tablespace* regardless of the number of TEMPFILES allocated to that temporary tablespace.

[Listings 15.1](#) and [15.2](#) show how to create a TTG and assign LMTTs to it.

Listing 15.1 Creating a TTG while Building New LMTTs

[Click here to view code image](#)

```
SQL> CREATE BIGFILE TEMPORARY TABLESPACE biglmtt1  
TEMPFILE '+DATA'
```

```
SIZE 256M  
AUTOEXTEND ON  
TABLESPACE GROUP ttg_1;
```

Tablespace created.

Listing 15.2 Adding an Existing LMTT to an Existing TTG

[Click here to view code image](#)

```
SQL> CREATE BIGFILE TEMPORARY TABLESPACE biglmtt2  
TEMPFILE '+DATA2'  
SIZE 256M  
AUTOEXTEND ON  
EXTENT MANAGEMENT LOCAL  
UNIFORM SIZE 64M;
```

Tablespace created.

```
ALTER TABLESPACE biglmtt2  
TABLESPACE GROUP ttg_1;
```

Tablespace altered.

```
SQL> SELECT * FROM dba_tablespace_groups;
```

| GROUP_NAME | TABLESPACE_NAME |
|------------|-----------------|
| TTG_1 | BIGLMTT1 |
| TTG_1 | BIGLMTT2 |

Once a TTG has been successfully created, it can be assigned as the temporary tablespace for an existing Oracle database user account just as if it were a normal temporary tablespace:

[Click here to view code image](#)

```
ALTER USER ap TEMPORARY TABLESPACE ttg_1;
```

User altered.

Note

Even though a TTG usually encompasses multiple temporary tablespaces, a query still might exhaust the space required to complete the sorting, aggregation, or join activity. Once the TEMPFILES for the TTG's temporary tablespaces that are selected and assigned for use have exhausted all the available free space, the statement will *still* fail with an ORA-01652 error ("Unable to extend temp

segment by < ... > in tablespace < ... >” because the temporary segments will simply not be allowed to extend beyond the boundaries of the selected temporary tablespace’s TEMPFILES.

Global Temporary Tables

Global temporary tables (GTTs) are an extremely useful option when an application needs to gather and retain data for an extremely brief duration. For example, it’s often necessary to gather data at the lowest level of necessary detail from an online transaction processing (OLTP) system to populate a data warehouse table with several different levels of rollup; a GTT is a perfect candidate for holding on to that data until the population is completed.

Another useful application is when each user of an application needs to gather and retain data for the duration of a transaction, but each session gathers a different set of data—perhaps for a different range of dates or a different data class—but afterwards, the data can be discarded immediately. GTTs are perfect for this situation because each user session can retain a completely isolated copy of the data without having to tokenize the data in the same table and clean it up via a `DELETE` operation afterwards. Fortunately, several new features available in the most recent Oracle database releases have made GTTs even more attractive. These features are discussed in the following sections.

Separate Temporary Tablespaces for GTTs

As of Oracle 11.2.0.1, a GTT can be placed into an LMTT that’s completely separate from other temporary segments needed for sorting, aggregation, or hash join processing, or even separated from other GTT types (for example, those used for extract-transform-load [ETL] processing versus those used for retaining data temporarily during query execution).

[Listing 15.3](#) illustrates this capability with the creation of an LMTT tablespace, `GTT_LMTT`, that’s designed specifically for GTTs.

Listing 15.3 Creating a Separate LMTT for GTTs

[Click here to view code image](#)

```
SQL> CREATE TEMPORARY TABLESPACE gtt_lmtt
  TEMPFILE '+DATA2'
  SIZE 64M
  AUTOEXTEND ON
  EXTENT MANAGEMENT LOCAL
  UNIFORM SIZE 1M;
```

Tablespace created.

[Listing 15.4](#) shows the creation of a GTT that retains an interim set of data from the AP.INVOICES and AP.INVOICE\_ITEMS tables for later insertion into the GL.REVENUES table that will reside within the GTT\_LMTT tablespace.

Listing 15.4 Creating a GTT within a Separate LMTT

[Click here to view code image](#)

```
SQL> CREATE GLOBAL TEMPORARY TABLE ap.gtt_load_revenue(
      acct_nbr          NUMBER(5)
    ,acct_desc          VARCHAR2(32)
    ,acct_balance      NUMBER(8,2)
  )
  ON COMMIT PRESERVE ROWS
  TABLESPACE gtt_lmtt;
```

Table created.

UNDO Deactivation for GTTs

Oracle Database Release 12.1.0.1 also eliminates one of the more pernicious problems with GTTs used within an LMTT: the retention of UNDO in the database's currently assigned UNDO tablespace. Prior to this release, the creation of rows in any GTT required UNDO entries to be generated and retained whenever data was added to the table, even if the table was going to be used only to stage data for immediate loading into another permanent table.

It's now possible to tell Oracle to shift retention of UNDO blocks into the temporary tablespace in which the GTT resides. This can be established for *all* GTTs by changing the setting for initialization parameter TEMP\_UNDO\_ENABLED to TRUE instead of its default value of FALSE. This setting can also be modified at the *session* level if the Oracle DBA desires to modify this behavior for only *some* GTTs, as shown in [Listing 15.5](#).

Listing 15.5 Enabling Temporary UNDO Generation

[Click here to view code image](#)

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

INSERT INTO gl.gtt_load_revenue VALUES (12300, 'GL Account 12300',
123.00);
INSERT INTO gl.gtt_load_revenue VALUES (12301, 'GL Account 12301',
123.01);
INSERT INTO gl.gtt_load_revenue VALUES (12302, 'GL Account 12302',
123.02);
INSERT INTO gl.gtt_load_revenue VALUES (12303, 'GL Account 12303',
```

```

123.03);
INSERT INTO gl.gtt_load_revenue VALUES (12304, 'GL Account 12304',
123.04);
INSERT INTO gl.gtt_load_revenue VALUES (12305, 'GL Account 12305',
123.05);
INSERT INTO gl.gtt_load_revenue VALUES (12306, 'GL Account 12306',
123.06);
INSERT INTO gl.gtt_load_revenue VALUES (12307, 'GL Account 12307',
123.07);
INSERT INTO gl.gtt_load_revenue VALUES (12308, 'GL Account 12308',
123.08);
INSERT INTO gl.gtt_load_revenue VALUES (12309, 'GL Account 12309',
123.09);
. . .
<< many other transactions! >>
. . .

```

[\*\*Listing 15.6\*\*](#) shows how a query against the new V\$TEMPUNDOSTAT dynamic view will yield information about any temporary UNDO that is retained within the GTT\_LMTT temporary tablespace.

Listing 15.6 Querying Against V\$TEMPUNDOSTAT

[Click here to view code image](#)

```

SET LINESIZE 90
SET PAGESIZE 20000
COL begin_time      FORMAT A20          HEADING "Begin Time"
COL end_time        FORMAT A20          HEADING "End Time"
COL undotsn         FORMAT 9999999999  HEADING "Undo|TSP #"
COL undoblkcnt      FORMAT 9999999     HEADING "Undo|Blocks"
COL txncount        FORMAT 9999999     HEADING "Trxn|Count"
COL maxquerylen     FORMAT 9999999     HEADING "Max|Query|Length"
TTITLE "Temporary UNDO Segment Statistics| (from V$TEMPUNDOSTAT)"
SELECT
    TO_CHAR(begin_time, 'mm-dd-yyyy hh24:mi:ss') begin_time
    ,TO_CHAR(end_time, 'mm-dd-yyyy hh24:mi:ss') end_time
    ,undotsn
    ,undoblkcnt
    ,txncount
    ,maxquerylen
FROM v$tempundostat
;
TTITLE OFF

```

Thu Jan

29

Temporary UNDO Segment Statistics
(from V\$TEMPUNDOSTAT)

| Begin Time | # | Blocks | End Time | Length | Undo TSP | Undo | |
|---------------------|---|------------|------------|--------|----------|------|--|
| 01-29-2015 19:57:00 | | | 01-29-2015 | | | | |
| 20:06:07 | | 3 | | 0 | 0 | 0 | |
| 01-29-2015 19:47:00 | | | 01-29-2015 | | | | |
| 19:57:00 | | 3 | | 1 | 1 | 0 | |
| 01-29-2015 19:37:00 | | | 01-29-2015 | | | | |
| 19:47:00 | | 3 | | 1 | 1 | 0 | |
| 01-27-2015 20:57:00 | | | 01-29-2015 | | | | |
| 19:37:00 | | 2147483647 | | 0 | 0 | 0 | |

Automatic Statistics Gathering for GTTs

Finally, as of Oracle Database Release 12.1.0.1, it is now possible to gather statistics *automatically* and *individually* for each session's version of a GTT via the `SET_GLOBAL_PREFS` procedure of the `DBMS_STATS` package. The advantage of this new feature is that more accurate optimizer statistics provide potentially better execution plan generation when a GTT is joined directly to other permanent tables for which accurate statistics already exist.

It's also possible to use the same procedure to capture statistics automatically from *one* representative session and then *share* those statistics with all other sessions that access the same GTT. This is particularly useful when the end result of populating a GTT varies only mildly between user sessions.

[Listing 15.7](#) shows two examples of how to implement *session-only* and *shared* GTT statistics gathering, respectively.

Listing 15.7 Gathering Session-Only and Shared GTT Statistics

[Click here to view code image](#)

```
-----
-- Gather statistics specific to each GTT:
-----
BEGIN
    DBMS_STATS.SET_GLOBAL_PREFS(
        pname => 'GLOBAL_TEMP_TABLE_STATS'
        ,pvalue => 'SESSION'
    );
END;
/
PL/SQL procedure completed successfully.
```

```

-----
-- Gather statistics and share them for all GTT iterations:
-----
BEGIN
    DBMS_STATS.SET_GLOBAL_PREFS(
        pname => 'GLOBAL_TEMP_TABLE_STATS'
        ,pvalue => 'SHARED'
    );
END;
/

```

PL/SQL procedure completed successfully.

Correcting TEMPFILE I/O Waits

The good news is that there are only a few situations when temporary tablespaces become a problem, and in all cases, it's only a small number of wait events and instance statistics that need to be monitored.

Undersized PGA

Since LMTTs are used only when there is insufficient space in the PGA, it's not uncommon to see a sudden spike in the related I/O wait events for a TEMPFILE. In [Listing 15.8](#), the PGA has been sized (via initialization parameter `PGA_AGGREGATE_TARGET`) to an absurdly small value of only 10 MB purely to demonstrate a corresponding spike in TEMPFILE I/O statistics and events when a query with large sort demands is issued.

Listing 15.8 Undersized PGA Resulting in TEMPFILE I/O Wait

[Click here to view code image](#)

```

SQL> ALTER SYSTEM SET pga_aggregate_target = 10M;
System altered.

SELECT /*+ MONITOR ReallyBadSorting */ DISTINCT
    cust_id
    ,prod_id
    ,SUM(qty)
    ,SUM(amt)
FROM (SELECT
    cust_id
    ,prod_id
    ,SUM(quantity_sold) qty
    ,SUM(amount_sold) amt
    FROM sh.sales
    WHERE cust_id BETWEEN 1 and 15000
    GROUP BY cust_id, prod_id

```

```

UNION
SELECT
    cust_id
    ,prod_id
    ,SUM(quantity_sold) qty
    ,SUM(amount_sold) amt
    FROM sh.sales
    WHERE cust_id BETWEEN 90000 and 105000
    GROUP BY cust_id, prod_id
UNION
SELECT
    cust_id
    ,prod_id
    ,SUM(quantity_sold) qty
    ,SUM(amount_sold) amt
    FROM sh.sales
    WHERE cust_id BETWEEN 20000 and 35000
    GROUP BY cust_id, prod_id
UNION
SELECT
    cust_id
    ,prod_id
    ,SUM(quantity_sold) qty
    ,SUM(amount_sold) amt
    FROM sh.sales
    WHERE cust_id BETWEEN 80000 and 95000
    GROUP BY cust_id, prod_id
UNION
SELECT
    cust_id
    ,prod_id
    ,SUM(quantity_sold) qty
    ,SUM(amount_sold) amt
    FROM sh.sales
    WHERE cust_id BETWEEN 30000 and 45000
    GROUP BY cust_id, prod_id
UNION
SELECT
    cust_id
    ,prod_id
    ,SUM(quantity_sold) qty
    ,SUM(amount_sold) amt
    FROM sh.sales
    WHERE cust_id BETWEEN 70000 and 85000
    GROUP BY cust_id, prod_id
UNION
SELECT
    cust_id
    ,prod_id
    ,SUM(quantity_sold) qty

```

```

        ,SUM(amount_sold) amt
    FROM sh.sales
    WHERE cust_id BETWEEN 40000 and 55000
    GROUP BY cust_id, prod_id
UNION
SELECT
        cust_id
        ,prod_id
        ,SUM(quantity_sold) qty
        ,SUM(amount_sold) amt
    FROM sh.sales
    WHERE cust_id > 100000
    GROUP BY cust_id, prod_id
UNION
SELECT
        cust_id
        ,prod_id
        ,SUM(quantity_sold) qty
        ,SUM(amount_sold) amt
    FROM sh.sales
    WHERE cust_id BETWEEN 50000 and 65000
    GROUP BY cust_id, prod_id
UNION
SELECT
        cust_id
        ,prod_id
        ,SUM(quantity_sold) qty
        ,SUM(amount_sold) amt
    FROM sh.sales
    WHERE cust_id BETWEEN 10000 and 25000
    GROUP BY cust_id, prod_id
UNION
SELECT
        cust_id
        ,prod_id
        ,SUM(quantity_sold) qty
        ,SUM(amount_sold) amt
    FROM sh.sales
    WHERE cust_id BETWEEN 60000 and 75000
    GROUP BY cust_id, prod_id
)
GROUP BY ROLLUP(cust_id, prod_id)
ORDER BY
    1 DESC
    ,2 ASC
    ,3 DESC
    ,4 ASC;

```

<< output from AUTOTRACE: >>

Statistics

```
        41 recursive calls
        35 db block gets
      352 consistent gets
    4858 physical reads
        0 redo size
 8014678 bytes sent via SQL*Net to client
 211026 bytes received via SQL*Net from client
 19136 SQL*Net roundtrips to/from client
        0 sorts (memory)
  3 sorts (disk)
 287014 rows processed
```

The queries and corresponding output shown in [Listings 15.9](#), [15.10](#), and [15.11](#) respectively illustrate the I/O statistics, session-level statistics, and session-level wait events that correlate to such a severely undersized PGA.

Listing 15.9 TEMPFILE I/O Statistics

[Click here to view code image](#)

```
SET LINESIZE 130
SET PAGESIZE 20000
COL tablespace  FORMAT A12          HEADING "Tablespace"
COL file#       FORMAT 9999         HEADING "TEMP|File|#"
COL phyrd$       FORMAT 9999999    HEADING "Phys|Reads"
COL phyrw$       FORMAT 9999999    HEADING "Phys|Writes"
COL phyblkrd$   FORMAT 9999999    HEADING "Phys|Blocks|Read"
COL phyblkwr$   FORAMAT 9999999   HEADING "Phys|Blocks|Written"
COL sbrs         FORMAT 9999999    HEADING "Single|Block|Reads"
COL readtim      FORMAT 9999999    HEADING "Read|Time"
COL writetim     FORMAT 9999999    HEADING "Write|Time"
COL sbrtm        FORMAT 99999      HEADING
"Single|Block|Read|Time"
COL avgiotim    FORMAT 99999       HEADING "Avg|I/O|Time"
COL lstiotim    FORMAT 99999       HEADING "Last|I/O|Time"
COL miniotim    FORMAT 99999       HEADING "Min|I/O|Time"
COL maxiortm    FORMAT 99999       HEADING "Max|I/O|Read|Time"
COL maxiowtm    FORMAT 99999       HEADING "Max|I/O|Write|Time"
TTITLE "TEMPFILE I/O Statistics"
SELECT
  TSP.name tablespace
 ,TS.file#
 ,phyrd$
 ,phyrw$
 ,phyblkrd$
 ,phyblkwr$
 ,singleblkrd$ sbrs
```

```

,readtim
,writetim
,singleblkrdtim sbrtm
,avgiotim
,lstiotim
,miniotim
,maxiortm
,maxiowtm
FROM v$tempstat TS, v$tempfile TF, v$tablespace TSP
WHERE TS.file# = TF.file#
    AND TF.ts# = TSP.ts#
ORDER BY TS.file#
;
TTITLE OFF
          TEMPFILE I/O
Statistics

          TEMP          Phys          Phys          Phys          Single
          File       Phys       I/O      Phys      Blocks      Blocks      Block   Re:
Write   Read    I/O      I/O      I/O      Read     Write
Tablespace      #      Reads    Writes      Read    Written      Reads   Tir
-----  -----  -----  -----  -----  -----  -----  -----  -----
-----  -----  -----  -----  -----  -----  -----  -----  -----
TEMP           1       839      852      6340      6692      57      6:
GTT_LMTT       2        2       6         2        6        2
BIGLMTT1       3        0       0         0        0        0
      0      0       7       7         0        0        0
BIGLMTT2       4        0       0         0        0        0
-----  -----  -----  -----  -----  -----  -----  -----  -----

```

Listing 15.10 TEMPFILE I/O Session-Level Statistics

[Click here to view code image](#)

```

SET LINESIZE 60
SET PAGESIZE 20000
COL name    FORMAT A45          HEADING "Statistic"
COL value   FORMAT 99999999     HEADING "Value"
TTITLE "Temporary Tablespace - Current Session Instance Statistics"
SELECT SN.name, SS.value
  FROM .
    v$mystat SS
   ,v$statname SN
 WHERE SS.statistic# = SN.statistic#
   AND (SN.name LIKE '%physical reads direct temporary%'
     OR SN.name LIKE '%physical writes direct temporary%')
;
TTITLE OFF

```

Temporary Tablespace - Current Session Instance Statistics

| Statistic | Value |
|--|-------|
| <hr/> | |
| physical reads direct temporary tablespace | 5060 |
| physical writes direct temporary tablespace | 5648 |

Listing 15.11 TEMPFILE I/O Session-Level Wait Events

[Click here to view code image](#)

```
SET LINESIZE 80
SET PAGESIZE 20000
COL event           FORMAT A40  HEADING "System Wait Event"
COL total_waits     HEADING "Total|Waits"
COL wait_secs       HEADING "Total|Wait|Time|(s)"
COL avg_wait_secs   HEADING "Avg|Wait|Time|(s)"
TTITLE "Temporary Tablespace - Current Session Wait Events"
SELECT
    DISTINCT event
    ,total_waits
    ,(time_waited / 100) wait_secs
    ,(average_wait / 100) avg_wait_secs
FROM
    v$session_event E
    ,v$mystat S
WHERE event LIKE '%temp%'
    AND E.sid = S.sid
;
TTITLE OFF
```

| Temporary Tablespace - Current Session Wait Events | | | |
|--|-------|------|-------|
| | Total | Wait | Total |
| System Wait | | | Time |
| Event | Waits | (s) | (s) |
| <hr/> | | | |
| direct path write | | | |
| temp | 1 | 0 | .0031 |
| direct path read | | | |
| temp | 278 | 1.79 | .0064 |

Inappropriate TEMPFILE Extent Sizing

Oracle recommends the following best practices to avoid inappropriately sizing temporary tablespaces:

- If the application workload is primarily a decision support system (DSS) that's dominated by complex joins—especially hash joins between multiple row sources—as well as intensive sorts and aggregations, data will be written to temporary segments in 64 KB multiples. It's therefore best to configure the corresponding temporary tablespace with an extent size that is a multiple of 64 KB; usually, an extent size of 1 MB is more than sufficient in this case.
- Conversely, if a large number of temporary large objects (LOBs) are going to be used—usually present because the database is processing transient, semistructured, or unstructured data such as XML or binary large objects (BLOBs)—a larger extent size is recommended, no smaller than 1 MB but no larger than 10 MB.

If these two simple rules of thumb for sizing LMTT extents are followed, then unexpectedly high wait events related to TEMPFILE I/O should rarely be encountered.

Inappropriate Use of GTTs

Finally, be wary of applications that use GTTs inappropriately, especially when the application has been migrated from other RDBMSs such as Sybase or SQL Server. These databases offer the capability to use a scratch area to create structures that *appear* to be very much like GTTs because they can be populated via SQL, but in reality they are more analogous to a PL/SQL *associative array*.

In these situations, it often makes sense to encourage application developers to rewrite their application code to leverage appropriate PL/SQL in-memory structures such as VARRAYS or PIPELINED table functions instead of GTTs. This strategy entirely avoids the unnecessary generation of UNDO as well as the overhead of the physical I/O required to write data to temporary segments.

My Oracle Support Reference Documents

MOS Note 1576956.1, “How to address high wait times for the ‘direct path’ write temp wait event,” contains valuable primary and ancillary information for additional reading and strengthening understanding of the myriad topics discussed in this chapter.

Summary

This chapter showed that without temporary tablespaces, it would be impossible to bring a database into read-only mode. Temporary tablespaces also make it possible to store large amounts of transient data in global temporary tables on a per-session basis. When a database instance exhausts its PGA memory and needs to complete a sort, aggregation, or hash join operation, temporary tablespaces can provide the necessary temporary repository for that information.

The most common I/O issues arise when a temporary tablespace is undersized, uses extents sized inappropriately for the temporary segments of the objects stored within, or

is overloaded because of the misuse of GTTs.

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 (11g and 12c), 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 10g, 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 8i 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 10g, 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

[Click here to view code image](#)

```
SQL> WITH system_event AS
  2      (SELECT CASE WHEN (event LIKE '%latch%' or event
  3                          LIKE '%mutex%' or event like 'cursor:%')
  4                          THEN event ELSE wait_class
  5                          END wait_type, e.*
  6      FROM v$system_event e)
  7  SELECT wait_type, SUM(total_waits) total_waits,
  8         round(SUM(time_waited_micro)/1000000,2)
time_waited_seconds,
  9         ROUND(  SUM(time_waited_micro)
10             * 100
11             / SUM(SUM(time_waited_micro)) OVER (), 2) pct
12  FROM (SELECT wait_type, event, total_waits, time_waited_micro
13      FROM system_event e
14      UNION
15      SELECT 'CPU', stat_name, NULL, VALUE
16      FROM v$sys_time_model
17      WHERE stat_name IN ('background cpu time', 'DB CPU')) l
18  WHERE wait_type <> 'Idle'
19  GROUP BY wait_type
20  ORDER BY 4 DESC
21  /
```

| WAIT_TYPE | TOTAL_WAITS | | | |
|----------------------|-------------|--------|--------|----------|
| TIME_WAITED_SECONDS | PCT | | | |
| CPU | | | | 1,494.63 |
| 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 | 1.21 |
| latch: cache buffers | | | | |
| chains | 2,058 | 8.74 | .40 | |
| Other | | 50 | | 7.26 |
| System | | | | |
| I/O | 6,166 | | 6.37 | .30 |
| cursor: pin | | | | |
| S | 235 | 3.05 | .14 | |
| Concurrency | | 60 | | 3.11 |
| 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

[Click here to view code image](#)

```
SQL> WITH latch AS (
  2   SELECT name,
  3         ROUND(gets * 100 / SUM(gets) OVER (), 2) pct_of_gets,
  4         ROUND(misses * 100 / SUM(misses) OVER (), 2)
  5         pct_of_misses,
  6         ROUND(sleeps * 100 / SUM(sleeps) OVER (), 2)
  7         pct_of.sleeps,
  8         ROUND(wait_time * 100 / SUM(wait_time) OVER (), 2)
  9         pct_of_wait_time
 10    FROM v$Latch)
 11   SELECT *
 12  FROM latch
 13 WHERE pct_of_wait_time > .1 OR pct_of.sleeps > .1
 14 ORDER BY pct_of_wait_time DESC;
```

| NAME | Pct of Gets | Pct of Misses | Pct of Sleeps | Pct of 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

[Click here to view code image](#)

```

SQL> 1
  1  WITH ash_query AS (
  2      SELECT event, program,
  3              h.module, h.action, object_name,
  4              SUM(time_waited)/1000 reltime, COUNT( * ) waits,
  5              username, sql_text,
  6              RANK() OVER (ORDER BY COUNT(*) DESC) AS wait_rank
  7      FROM v$active_session_history h
  8      JOIN dba_users u USING (user_id)
  9      LEFT OUTER JOIN dba_objects o
 10          ON (o.object_id = h.current_obj#)
 11      LEFT OUTER JOIN v$sql s USING (sql_id)
 12      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 WHERE id BETWEEN 124410 AND 360759

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

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

latch: shared
pool      SQL*Plus      OPSG      LT_SALES_PK      1
SELECT  quantity_sold , amount_sold FROM lt_sales t539555 WHERE 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

[Click here to view code image](#)

```
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 /
  8             SUM(concurrency_wait_time) OVER (), 2)
pct_of_con_time,
  9       RANK() OVER (ORDER BY concurrency_wait_time DESC)
ranking
10      FROM v$sql
11      WHERE elapsed_time > 0)
12  SELECT sql_text, con_time_ms, con_time_pct,
13        pct_of_con_time
14  FROM sql_conc_waits
15  WHERE ranking <= 10
16  ORDER BY ranking ;
```

SQL

| Conc | % | Tot | SQL | Text | Conc | Time (ms) | Time% |
|--|-------|-------|-------|------------------|-------|-----------|-------|
| SQL | | | | | | | |
| Text | | | | | | | |
| 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 10g 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:

[Click here to view code image](#)

```
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:

[Click here to view code image](#)

```
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:

[Click here to view code image](#)

```
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

[Click here to view code image](#)

```
SQL> WITH force_matches AS
  2      (SELECT force_matching_signature,
  3          COUNT( * ) matches,
  4          MAX(sql_id || child_number) max_sql_child,
  5          DENSE_RANK() OVER (ORDER BY COUNT( * ) DESC)
  6              ranking
  7      FROM v$sql
  8      WHERE force_matching_signature <> 0
  9          AND parsing_schema_name <> 'SYS'
 10      GROUP BY force_matching_signature
 11      HAVING COUNT( * ) > 5)
 12  SELECT sql_id,      matches, parsing_schema_name schema,
 13        sql_text
 14    FROM      v$sql JOIN force_matches
 15      ON (sql_id || child_number = max_sql_child)
 16  WHERE ranking <= 10
```

```
16 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:

[Click here to view code image](#)

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

with a statement like this:

[Click here to view code image](#)

```
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

[Click here to view code image](#)

```
SQL> SELECT COUNT(DISTINCT l.addr) cbc_latches,
  2          SUM(COUNT( * )) buffers,
  3          MIN(COUNT( * )) min_buffer_per_latch,
  4          MAX(COUNT( * )) max_buffer_per_latch,
  5          ROUND(AVG(COUNT( * ))) avg_buffer_per_latch
  6  FROM        v$Latch_Children l
  7  JOIN        x$bh b
  8  ON (l.addr = b.hladdr)
  9 WHERE name = 'cache buffers chains'
10 GROUP BY l.addr;
```

| CBC Latch Count | Buffer Count | Cache Min Per Latch | Max Per Latch | Avg 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

[Click here to view code image](#)

| 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 11g 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.

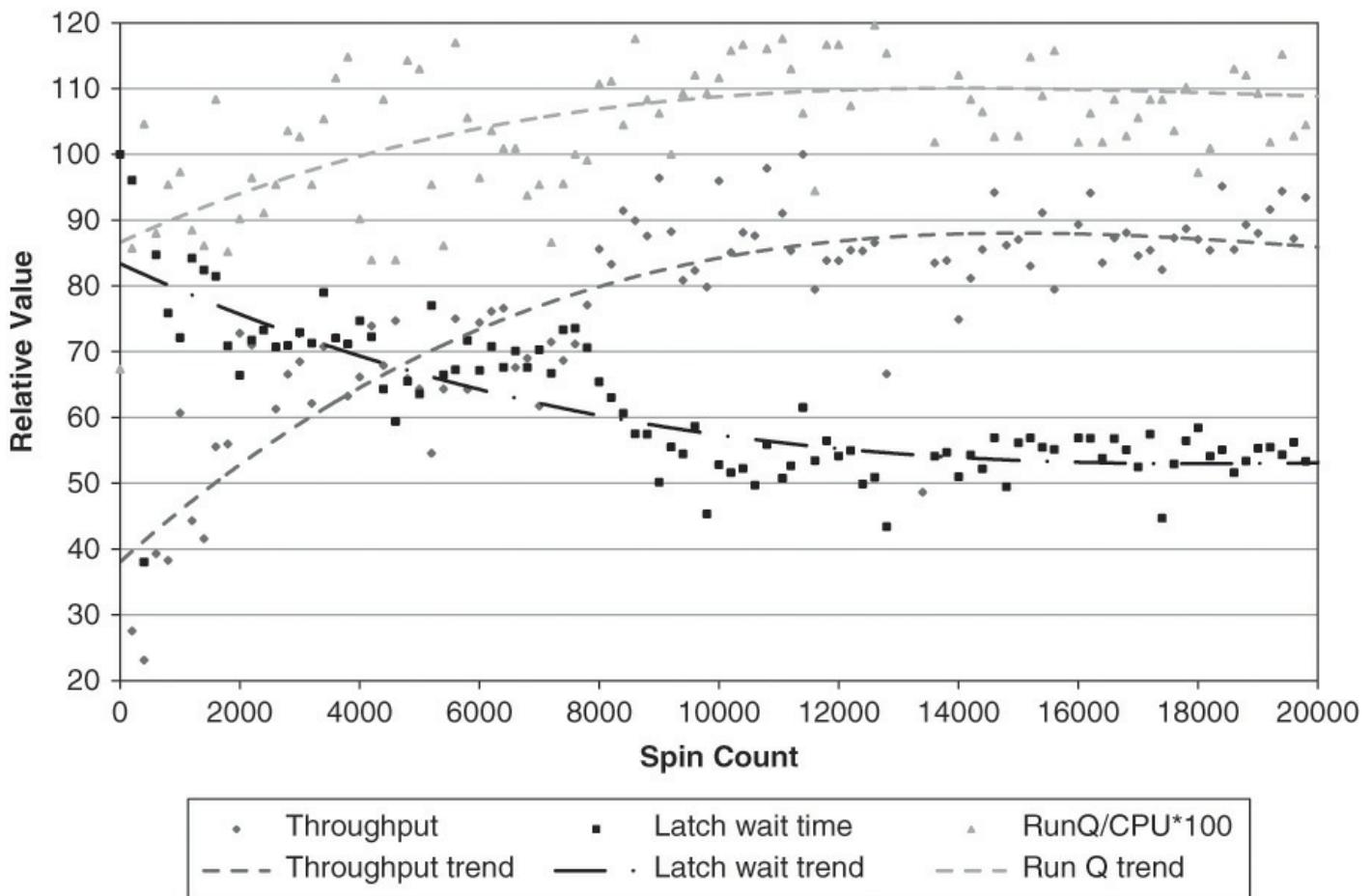


Figure 16.1 Relationship between `_spin_count`, latch waits, and throughput

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 11g), 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` 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.

17. Using SSDs to Solve I/O Bottlenecks

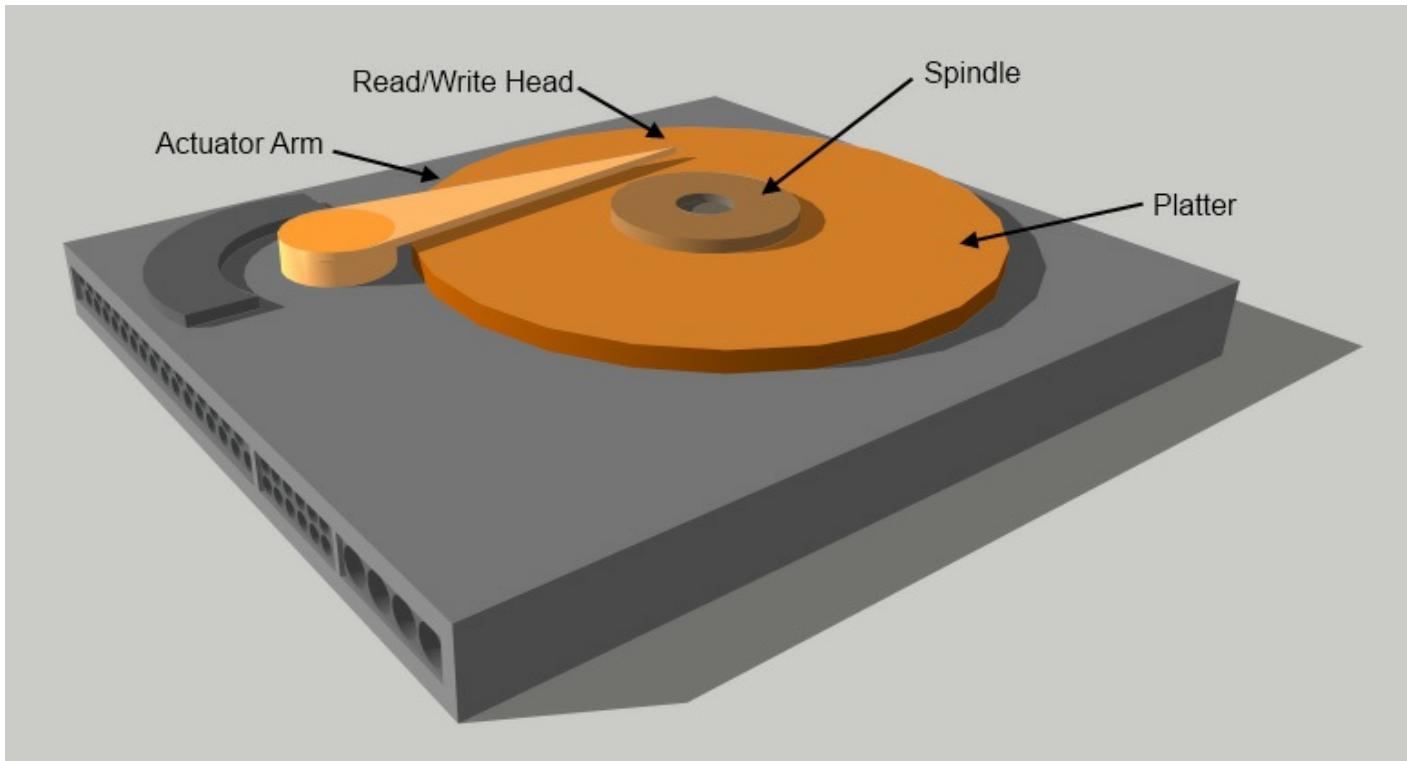
For most of the history of the Oracle relational database management system (RDBMS), the primary goal of performance tuning has been to avoid I/O at all cost. I/O to magnetic disk devices has always been many orders of magnitude slower than memory or CPU access, and the situation has only grown worse as Moore's law accelerated the performance of CPU and memory while leaving mechanical disk performance behind. Solid-state drive (SSD) represents a revolutionary technology that provides a quantum leap in disk performance.

In recent years, SSD technology has shifted from an expensive luxury to a mainstream technology that has a place in almost every performance-critical Oracle database. Replacing all magnetic disk with SSD is sometimes infeasible because of the economics of increasingly large databases. Therefore, it becomes a new and critical requirement of the Oracle performance practitioner to understand the physics of solid-state devices and to employ them selectively to get the best return on investment.

To understand how flash technology contributes to Oracle performance and how best to exploit that technology, this chapter starts by comparing the performance and economics of flash and spinning disk technology. We then discuss how SSD can be deployed in Oracle databases to improve performance and solve I/O bottlenecks.

Disk Technologies: SSD versus HDD

Magnetic disks have been a continuous component of mainstream computer equipment for generations of IT professionals. First introduced in the 1950s, the fundamental technology has remained remarkably constant: one or more *platters* contain magnetic charges that represent bits of information. These magnetic charges are read and written by an *actuator arm*, which moves across the disk to a specific position on the radius of the platter and then waits for the platter to rotate to the appropriate location (see [Figure 17.1](#)). The time taken to read an item of information is the sum of the time taken to move the head into position (*seek time*), the time taken to rotate the item into place (*rotational latency*), and the time taken to transmit the item through the disk controller (*transfer time*).



Source: Farooq, T, et al. *Oracle Exadata Expert's Handbook*. New York: Addison-Wesley, 2015.

Figure 17.1 Magnetic disk architecture

Moore's law—first articulated by Intel founder Gordon Moore—observes that transistor density on a computer microchip doubles every 18 to 24 months. In its broadest interpretation, Moore's law reflects the exponential growth that is commonly observed in almost all electronic components: improving CPU speed, RAM, and disk storage capacity.

While this exponential growth is observed in almost all electronic aspects of computing, including hard disk densities, it does not apply to mechanical technologies such as those underlying magnetic disk I/O. For instance, had Moore's law been in effect for the rotational speed of disk devices, disks today would be rotating about 100 million times faster than in the early 1960s—in fact, they are rotating only eight times faster.

Note

If an 8-inch disk rotating at 2800 RPM in 1962 was subject to Moore's law, by now the velocity on the outside of the disk would be about 10 times the speed of light. As Scotty from *Star Trek* would say, “Ye canna change the laws of physics!”

So while the other key components of computer performance have been advancing exponentially, magnetic disk drives have been improving only incrementally. There have been some significant technical innovations—Perpendicular Magnetic Recording, for instance—but these in general have led to improved capacity and reliability rather than speed. Consequently, disk drives are slower today (when compared to other components or even their own storage capacity) than in the past. As a result, disk I/O has been increasingly limiting the performance of database systems, and the practice of database

performance tuning has largely become a process of avoiding disk I/O whenever possible.

The Rise of Solid-State Flash Disks

Heroic efforts have been made over the years to avoid the increasingly onerous bottleneck of the magnetic disk drive. The most prevalent, and until recently most practical, solution has been to *short stroke* and *stripe* magnetic discs: essentially installing more disks than are necessary for data storage in order to increase total I/O bandwidth. This approach increases the overall I/O capacity of the disk subsystem but has a limited effect on latency for individual I/O operations.

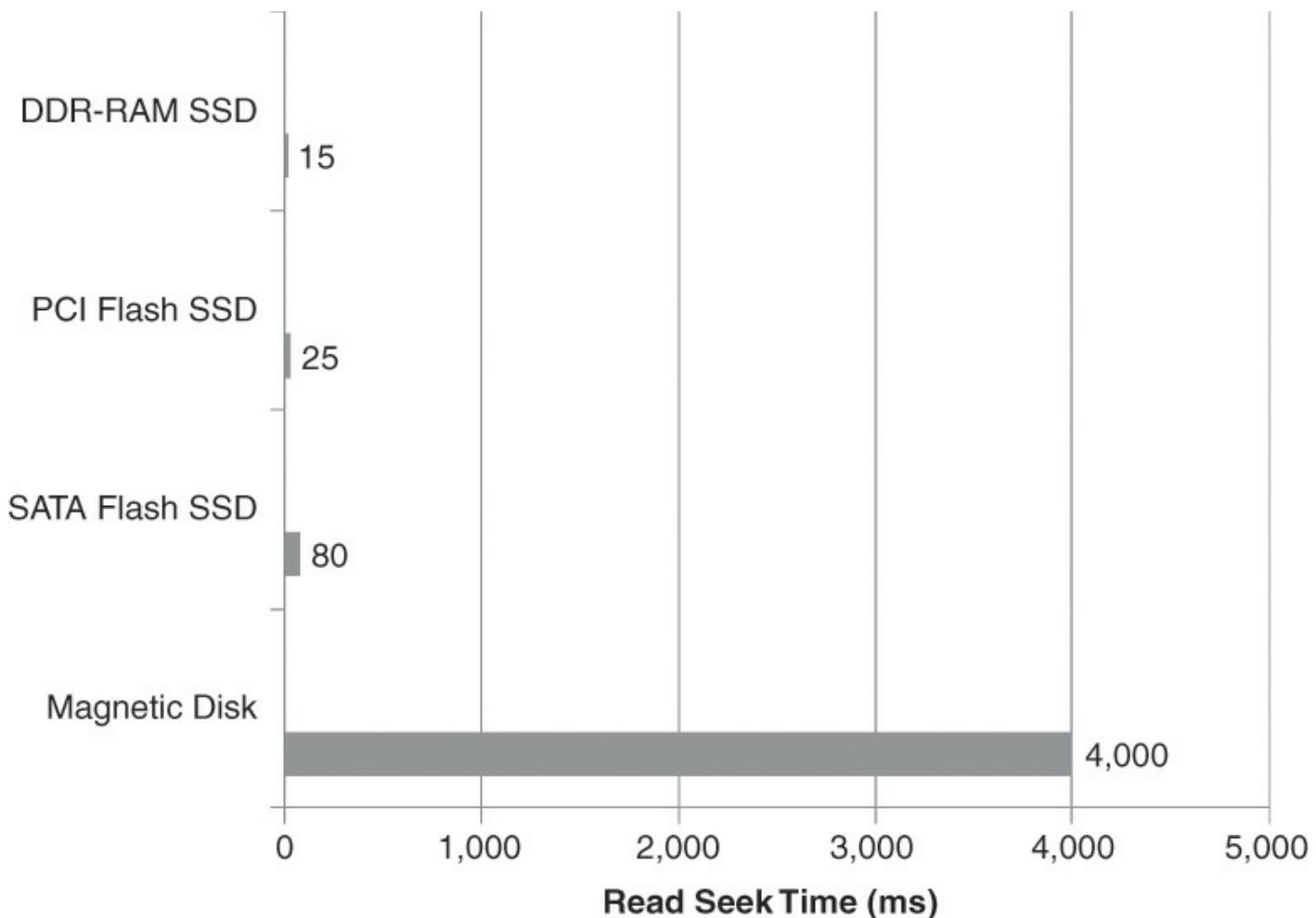
Note

Short stroking involves limiting the amount of data stored on a disk so that all of the data is concentrated on the perimeter of the disk. This technique can improve I/O seek latency by reducing the average distance the actuator arm needs to move across the disk and by concentrating data in the outer sectors of the platter where the overall rotational speed is highest. A short-stroked disk drive may deliver maybe twice the throughput of a drive that is at full capacity depending on how much of the storage capacity is discarded.

In contrast to a magnetic disk, SSDs contain no moving parts and provide tremendously lower I/O latencies. Commercial SSDs are currently implemented using either double-data rate (DDR) RAM—effectively a battery-backed RAM device—or NAND flash. NAND flash is an inherently nonvolatile storage medium and almost completely dominates today's SSD market.

Flash SSD Latency

Performance of flash SSD is orders of magnitude superior to performance of magnetic disk devices, especially for read operations. [Figure 17.2](#) compares the read latency of various types of SSD and traditional magnetic disk (note these are approximate and vary markedly dependent on the drive make and configuration).



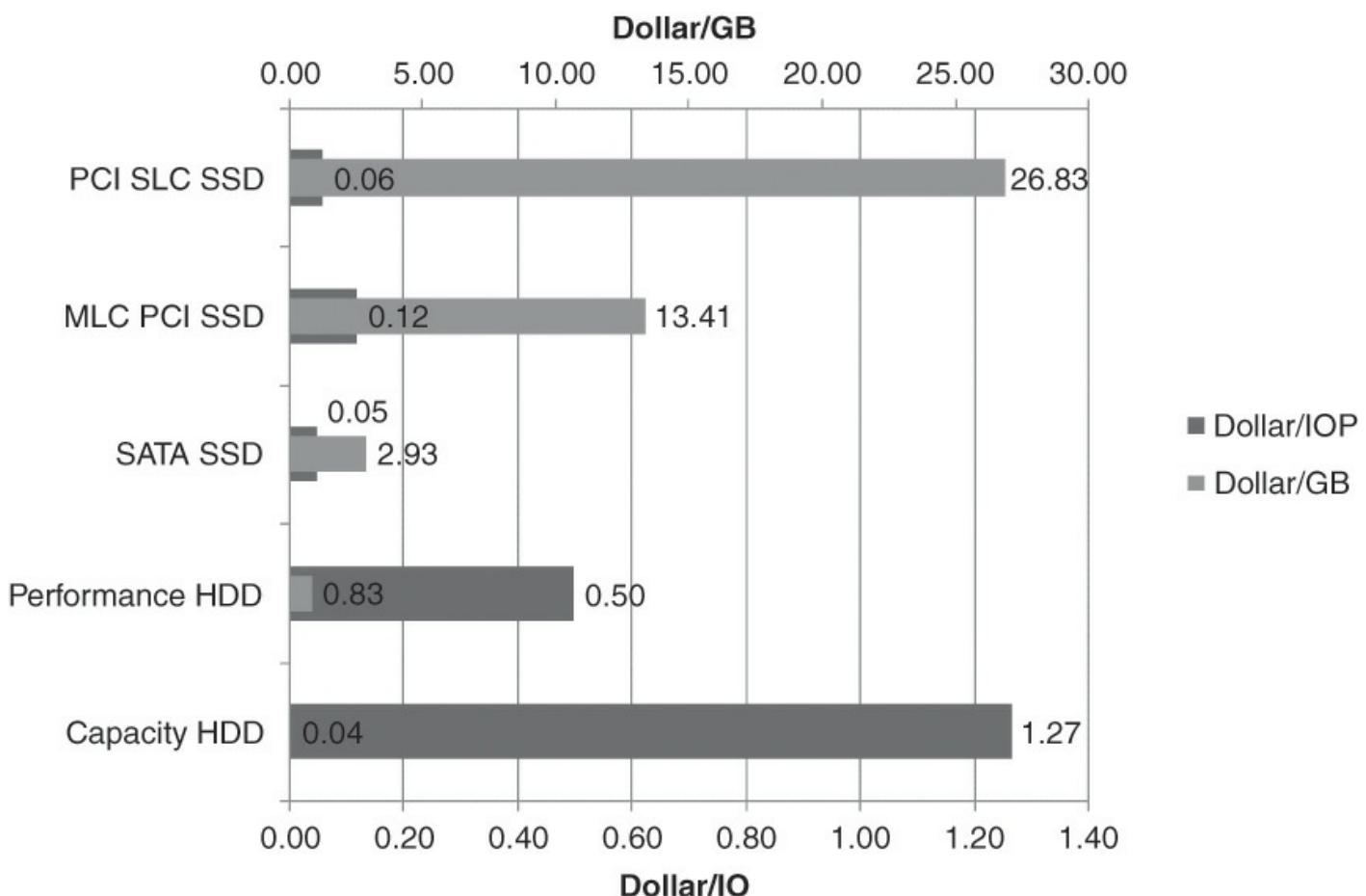
Source: Farooq, T, et al. *Oracle Exadata Expert's Handbook*. New York: Addison-Wesley, 2015.

Figure 17.2 Seek times for various disk technologies

Economics of SSD

The promise of SSD has led some to anticipate a day when all magnetic disks are replaced by SSDs. While this might someday come to pass, in the short term, the economics of storage and the economics of I/O are at odds: magnetic disk provides a more economical medium per unit of storage, while flash provides a more economical medium for delivering high I/O rates and low latencies.

[Figure 17.3](#) illustrates the two competing trends: while the cost of I/O decreases with solid-state technology, the cost per terabyte increases. Various flavors of SSD (PCIe SATA and MLC/SLC) offer different price and performance characteristics compared to the various categories of magnetic disks (15 K vs. 7 K RPM). The SSD devices that offer good economies of I/O offer poorer economies for mass storage. Of course, the cost per gigabyte for SSD is dropping rapidly, but no faster than the falling cost of magnetic disks or the growth in database storage demand, especially in the era of Big Data.



Source: Farooq, T, et al. *Oracle Exadata Expert's Handbook*. New York: Addison-Wesley, 2015.

Figure 17.3 Economics of storage for solid-state and magnetic disk

Since most databases include both hot and cold data—small amounts of frequently accessed data as well as large amounts of idle data—most databases experience the best economic benefit by combining both solid-state and traditional magnetic disk technologies.

Oracle has implicitly acknowledged this in the architecture of the Exadata Database Machine, which combines both magnetic disks and flash disks to provide the optimal balance between storage economics and performance. If Exadata contained only magnetic disks, it could not provide superior online transaction processing (OLTP) performance; if it contained only SSD, it could not offer compelling economical storage for large databases.

The performance differences between SSDs and magnetic disks involve more than simply a reduction in read latency. Just as the fundamental architecture of magnetic disks favors certain I/O operations, the architecture of SSDs favors specific and different types of I/O. Understanding how an SSD handles the different types of operations helps us make the best decisions when choosing configuration options.

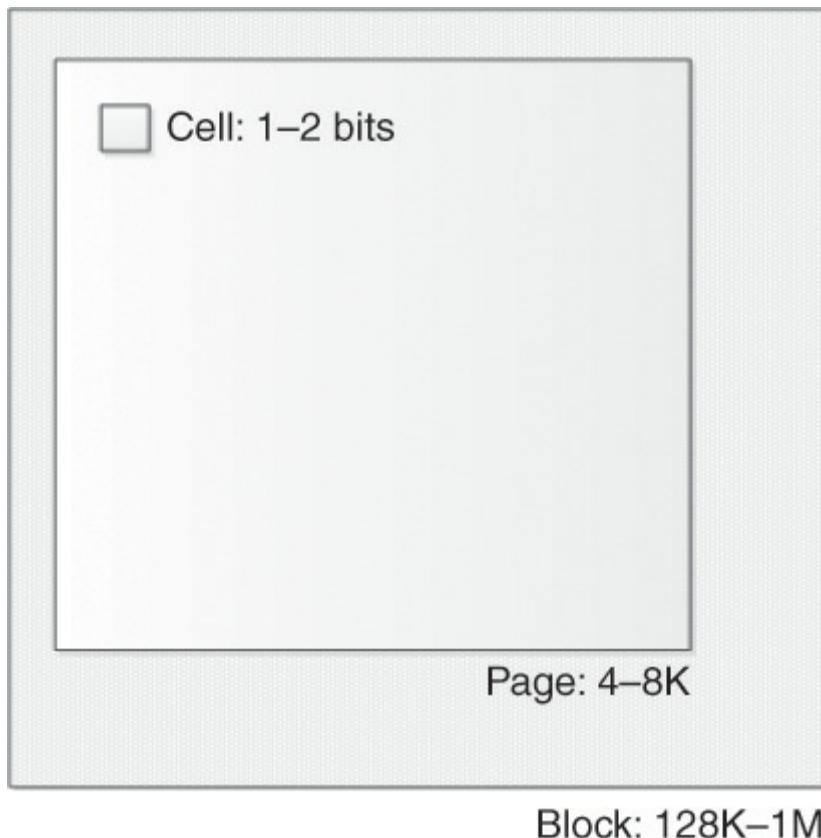
SLC, MLC, and TLC Disks

Flash-based SSDs have a three-level hierarchy of storage. Individual bits of information are stored in *cells*. In a single-level cell (SLC) SSD, each cell stores only a single bit. In a multilevel cell (MLC), each cell may store two or more bits of information. MLC SSD

devices consequently have greater storage densities but lower performance and reliability. However, because of the economic advantages of MLC, flash storage vendors have been working tirelessly to improve the performance and reliability of MLC flash, and it is now generally possible to get excellent performance from an MLC-based device.

Until recently, MLC SSDs contained only 2 bits of information per cell. However, triple-level cache (TLC) SSDs are now becoming available: these are MLC devices that can store 3 bits of information, and in theory, higher-density MLCs may appear in the future. However, increasing the number of bits in each cell reduces the longevity and performance of the cell.

Cells are arranged in *pages*, typically 4 K or 8 K in size, and pages are arranged in *blocks* of between 128 K and 1 M, as shown in [Figure 17.4](#).

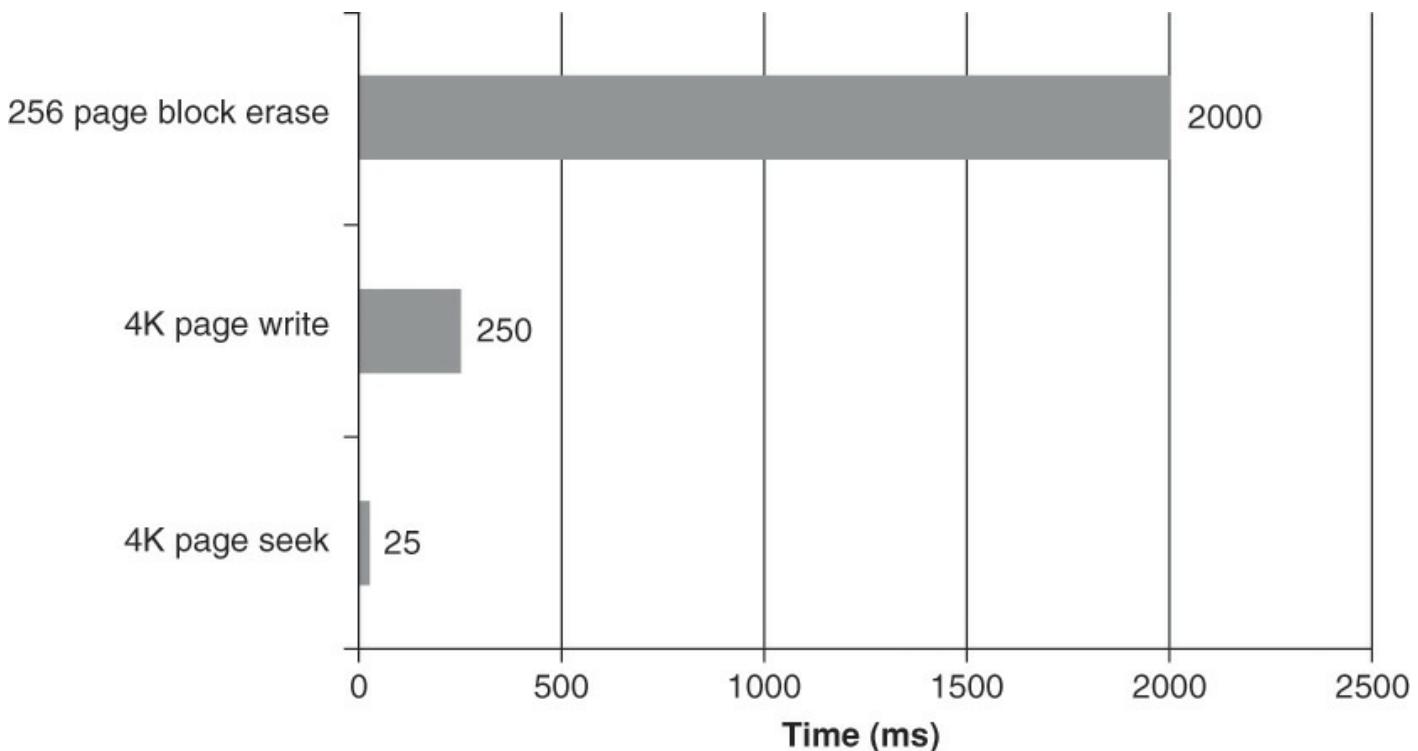


Source: Farooq, T, et al. *Oracle Exadata Expert's Handbook*. New York: Addison-Wesley, 2015.

Figure 17.4 SSD storage hierarchy (logarithmically scaled)

Write Performance and Endurance

The page and block structures are particularly significant for flash SSD performance because of the special characteristics of write I/O in flash technology. Read operations and initial write operations require only a single page I/O. However, changing the contents of a page requires an erase and overwrite of a complete block. Even the initial write can be significantly slower than a read, but the block erase operation is particularly slow—around 2 milliseconds. [Figure 17.5](#) shows the approximate times for a page seek, page write, and block erase.



Source: Farooq, T, et al. *Oracle Exadata Expert's Handbook*. New York: Addison-Wesley, 2015.

Figure 17.5 Flash SSD performance characteristics

Write I/O has another consequence in flash SSDs: after a certain number of writes, a cell may become unusable. This write endurance limit differs among drives but is typically between 10,000 cycles for a low-end MLC device and up to 1,000,000 cycles for a high-end SLC device. SSDs will generally fail safe when a cell becomes unwritable, marking the page as bad and moving the data to a new page.

Garbage Collection and Wear Leveling

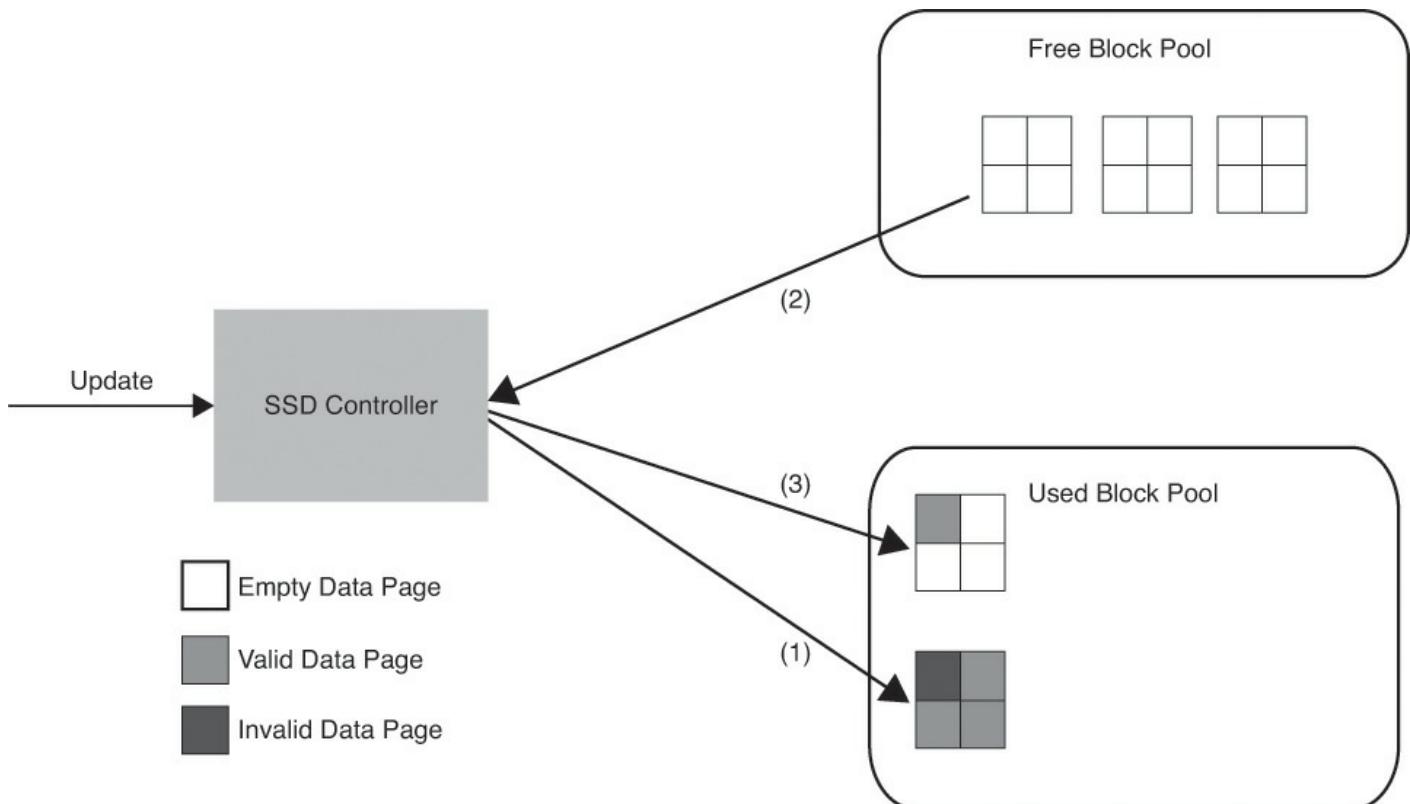
Enterprise SSD manufacturers go to great lengths to avoid the performance penalty of the erase operation and the reliability concerns raised by write endurance. Sophisticated algorithms are used to ensure that erase operations are minimized and that writes are evenly distributed across the device.

Erase operations are avoided through the use of *free lists* and *garbage collection*. During an update, the SSD marks the block to be modified as invalid and copies the updated contents to an empty block, retrieved from a free list. Later, garbage collection routines recover the invalid block, placing it on a free list for subsequent operations. Some SSDs maintain storage above the advertised capacity of the drive to ensure that the free list does not run out of empty blocks for this purpose. This is known as *overprovisioning*.

Note

Microsoft Windows includes a TRIM command, which allows the operating system to inform the SSD when entire files are deleted and can therefore be moved to the free block pool. However, since we almost never delete Oracle database files in production systems, this command has little usefulness for

[Figure 17.6](#) illustrates a simplified SSD update algorithm. To avoid a time-consuming ERASE operation, the SSD controller marks a block to be updated as invalid (1), then takes an empty block from the free list (2) and writes the new data to that block (3). Later on, when the disk is idle, the invalid blocks are garbage collected by erasing the invalidated block.



Source: Farooq, T, et al. *Oracle Exadata Expert's Handbook*. New York: Addison-Wesley, 2015.

Figure 17.6 Flash SSD garbage collection

Wear leveling is the algorithm that ensures that no particular block is subjected to a disproportionate number of writes. It may involve moving the contents of hot blocks to blocks from the free list and eventually marking overused blocks as unusable.

Wear leveling and garbage collection algorithms in the disk controller are what separates the men from the boys in SSDs. Without effective wear leveling and garbage collection, we would expect SSD drives to exhibit performance degradation and a reduced effective life. With the sophisticated algorithms employed by major SSD vendors, these issues are rarely significant.

However, the implications of garbage collection and wear leveling do influence what we expect from a database system that utilizes flash SSDs. Sustained heavy sequential write I/O or write operations that concentrate on a small number of hot pages may not allow the garbage collection and wear leveling algorithms time to clean up invalid pages or distribute hot pages between operations. As a result, SSDs subject to these sorts of workloads may exhibit performance or storage capacity degradation. This may influence decisions around the use of flash SSD for sequential write-intensive workloads such as those involved in redo log operations.

SATA versus PCIe SSD

Flash SSD drives come in two fundamental types: serial advanced technology attachment (SATA) and Peripheral Component Interconnect Express (PCIe). A SATA SSD connects to the computer using the SATA interface employed by most magnetic disks. A PCIe drive connects directly to the PCI bus, familiar to most of us as the slot in our home computers to which graphic cards are attached.

SATA SSDs are convenient because they can be attached wherever a traditional magnetic SATA disk is found. Unfortunately, the SATA interface was designed for magnetic disks, which have latencies in the order of milliseconds—thousands of microseconds. When a flash SSD, which has latencies in the order of microseconds, uses a SATA interface, the overhead of SATA becomes significant and may account for as much as two-thirds of the total read latency.

The PCIe interface was designed for extremely low-latency devices such as graphics adaptors, and allows these devices to interact directly with the computer's processor bus. Consequently, PCIe SSD devices have much lower latencies than SATA SSD—read latencies in the order of 25 microseconds versus perhaps 75 microseconds for a typical SATA SSD.

Storage arrays are available that include PCIe- or SATA-based SSDs. These arrays may be composed exclusively of SSDs or may combine SSDs and magnetic disks in a hybrid array.

Using SSD Devices in an Oracle Database

Regardless of the flavor of SSD, the use of an SSD from an Oracle database point of view is identical to the use of a magnetic disk. Locally attached SATA, locally attached PCI flash, or flash devices in a storage array will always present to the operating system as just another logical device (logical unit number [LUN]) that you can use to create a file system, a redundant array of independent disks (RAID) group, an Automatic Storage Management (ASM) disk group, or as just a bunch of disks (JBOD).

However, if you have a database server that has access to both magnetic disk and SSD, then your configuration may be more complex. You may choose to use SSD for selected tablespaces, segments, or in some cases as part of the Oracle Database Flash Cache.

The Oracle Database Flash Cache

Oracle introduced the *Database Flash Cache* (DBFC) in Oracle 11.2 but has made it available only on Oracle operating systems (Solaris and Oracle Enterprise Linux [OEL]). Provided you are using one of these operating systems, the DBFC is probably the easiest way to leverage SSD in your database configuration.

The DBFC serves as a secondary cache to the Oracle buffer cache. Oracle manages data blocks in the buffer cache using a modified least recently used (LRU) algorithm. Simplistically speaking, blocks *age out* of the buffer cache if they have not been

accessed recently. When the DBFC is active, blocks that age out of the buffer cache are not discarded but are instead written by the database writer (DBWR) to the flash device. Should the blocks be required in the future, they can be accessed from the flash cache instead of from database files residing on slower magnetic disk.

Free Buffer Waits

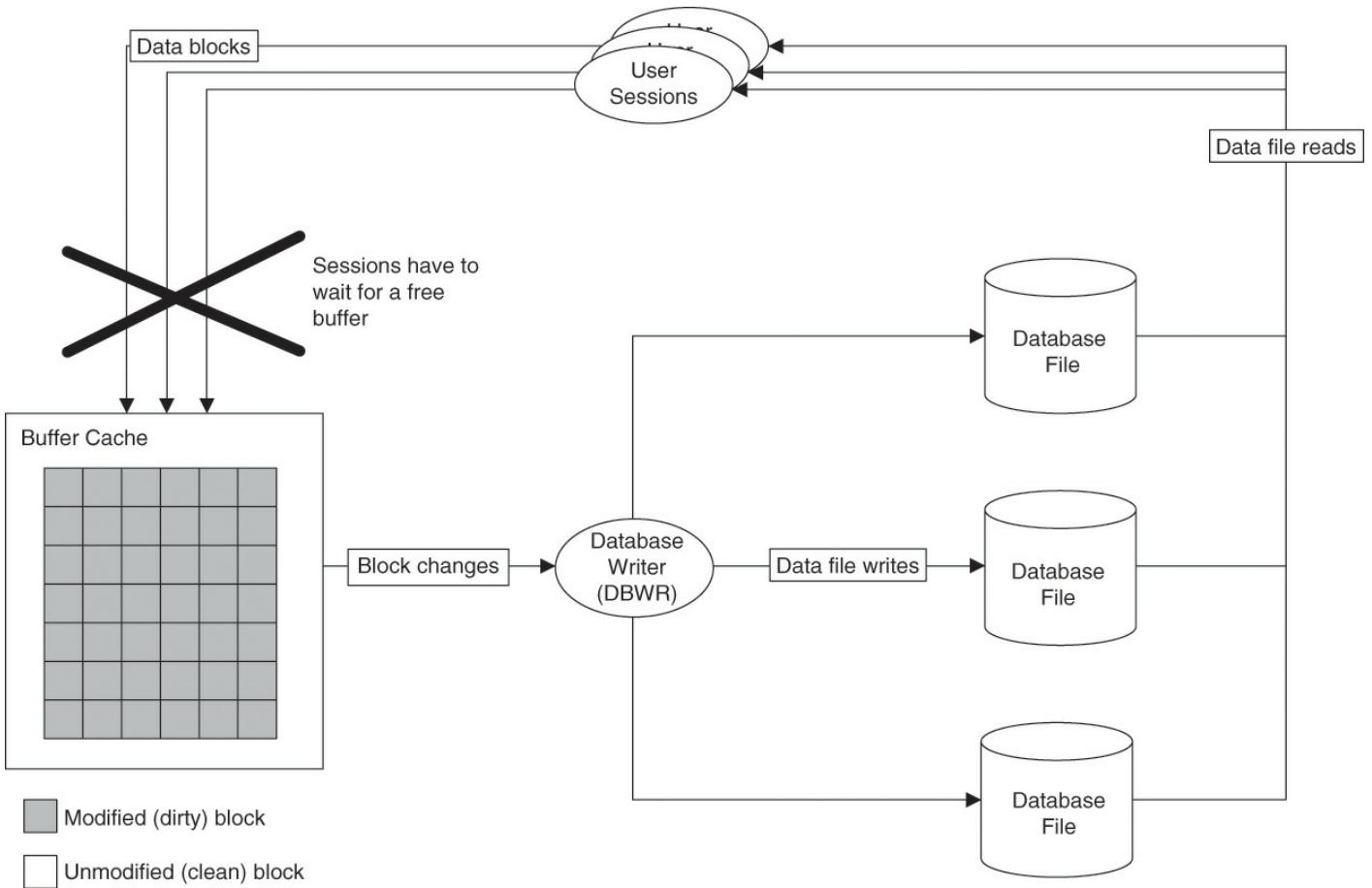
The design of the DBFC is specifically intended to complement the buffer cache without creating any contention points within the combined buffer cache/flash cache areas. In particular, the designers of the DBFC were concerned that it should not contribute to free buffer waits contention. So to begin, let's briefly review the *free buffer waits* contention scenario.

When a session wants to access an individual block of data from a table or index, it first looks for that block in the buffer cache. Oracle implements a complex algorithm to identify which blocks should be kept in memory. From a simplistic point of view, the longer it has been since the block was accessed, the more likely it is to be removed from the cache to make room for other blocks. This modified LRU algorithm is implemented by the *LRU list*—if a block is accessed, it may be moved up the list (metaphorically speaking). If blocks are not accessed, they may move down the list and eventually be aged out of the buffer cache.

When a data manipulation language (DML) statement modifies the contents of a block, the changes are made to a copy of the block in memory. The changed “dirty” buffer will not immediately be written to disk. The DBWR background process will write the dirty buffers out to database files at a later time. This deferred writing of changed blocks is generically known as a *lazy write*, and the principle is used by most databases.

Dirty (modified) buffers cannot be overwritten until the DBWR has written them to disk. If a session cannot find a clean buffer after scanning through the buffer cache, then it signals the DBWR to write modified buffers to disk and experiences a *free buffer wait*. By default, the session will scan 40 percent of the buffer cache before signaling the DBWR.

Free buffer waits are a commonly encountered form of buffer cache contention. They often occur when database read bandwidth exceeds write bandwidth—sessions attempt to load blocks into the buffer cache faster than they can be flushed to disk. [Figure 17.7](#) illustrates the phenomenon.



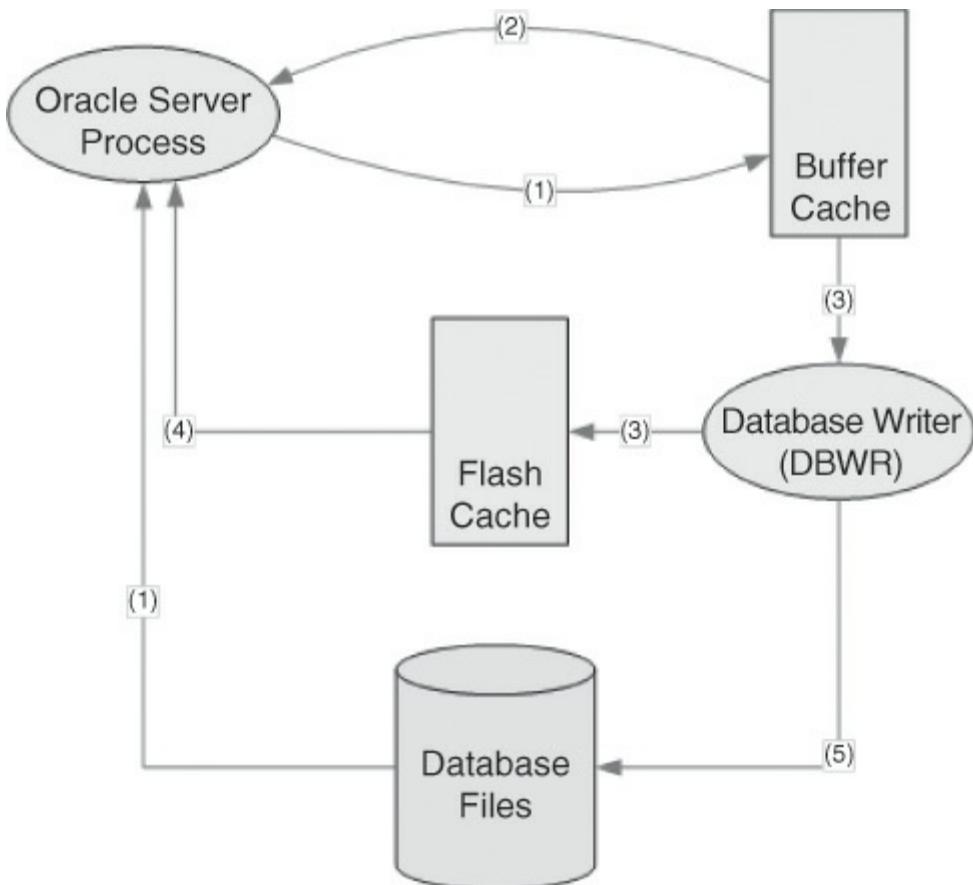
Source: Farooq, T, et al. *Oracle Exadata Expert's Handbook*. New York: Addison-Wesley, 2015.

Figure 17.7 Free buffer waits

The DBFC serves as a secondary cache to the Oracle buffer cache. When the flash cache is present, blocks that age out of the buffer cache are not discarded; they are instead written to the flash cache. Should the blocks be required in the future, they can be read from the flash cache instead of from database files residing on (presumably) slower magnetic disks.

Blocks are written to the flash cache by the DBWR process, which is normally responsible for writing modified blocks to datafiles. However, the DBWR maintains the flash cache as a secondary priority to its primary responsibility of writing out modified blocks to disk. Should the DBWR be busy with datafile writes, it will bypass flash cache maintenance. This “willful neglect” is designed to ensure that the maintenance of the flash cache does not inadvertently cause free buffer wait contention, which occurs when the DBWR cannot flush dirty buffers from the cache to disk in a timely manner.

[Figure 17.8](#) shows the Oracle DBFC architecture. An Oracle server process reads blocks from the database files and stores them in the buffer cache (1). Subsequent reads can obtain the data buffers from the buffer cache without having to access the database file (2). When the block *ages out* of the buffer cache, the DBWR will load those blocks into the flash cache (3), but only if it does not interfere with writing modified blocks to disk (5). Subsequent reads may now find a block either in the flash cache (4) or the buffer cache (2).



Source: Farooq, T, et al. *Oracle Exadata Expert's Handbook*. New York: Addison-Wesley, 2015.

Figure 17.8 Oracle Database Flash Cache architecture

Configuring and Monitoring DBFC

The DBFC is configured through the database parameters `DB_FLASH_CACHE_FILE` and `DB_FLASH_CACHE_SIZE`. Individual segments (tables, indexes, etc.) may be prioritized or excluded from the flash cache by using the `FLASH_CACHE` segment properties clause, which takes the values `KEEP`, `NONE`, or `DEFAULT`. Segments that have the `KEEP` property will be maintained in the flash cache in preference to those with the `DEFAULT` property.

The `DB_FLASH_CACHE_FILE` parameter may reference a file location on an SSD-backed file system or to an ASM disk group. In either case, Oracle will not necessarily validate that the location is SSD backed, so be careful to ensure that the locations are in fact located on a SSD device.

Following is an example of creating a DBFC on a file system. Note that the database must be restarted before the parameters take effect.

[Click here to view code image](#)

```
SQL> ALTER SYSTEM SET db_flash_cache_size=1024M SCOPE=SPFILE;
System altered.
```

```
SQL> ALTER SYSTEM SET
  db_flash_cache_file='/ssdfs/ora/dbfc/g11g_dbfc1.dbf'
  SCOPE=SPFILE;
System altered.
```

```

SQL> shutdown immediate
SQL> startup
ORACLE instance started.
SQL> show parameter flash_cache
NAME          TYPE        VALUE
-----
db_flash_cache_file  string      /ssdfs/ora/dbfc/g11g_dbfc1.dbf
db_flash_cache_size big integer 1G

```

Setting the DB\_FLASH\_CACHE\_SIZE parameter to 0 on a running database disables the flash cache.

You may encounter error ORA-00439 “feature not enabled: Server Flash Cache” when starting the database after configuring the flash cache. This is expected if the operating system is not an Oracle operating system (e.g., neither Solaris nor OEL) but can also occur due to bugs in early versions of Oracle 11g and 12g running on OEL 6—see MOS Note 1550735.1, “Database startup failing with ORA-00439 after enabling flash cache.”

Using the FLASH\_CACHE Clause

The STORAGE (FLASH\_CACHE) clause controls the caching of segments in the DBFC. The clause has three settings:

- **NONE:** Blocks will not be cached.
- **DEFAULT:** Blocks will be cached at the normal priority.
- **KEEP:** Blocks will be cached at high priority and will not be evicted from the cache unless there are no default blocks available.

The DBFC is mostly effective during indexed reads—because full table scans are likely to employ direct path reads, bypassing the buffer cache. Therefore, it is likely that you will want to apply the FLASH\_CACHE clause to indexes as well as tables, as in the following example:

[Click here to view code image](#)

```

SQL> ALTER TABLE sales_fc_none STORAGE (FLASH_CACHE NONE);
Table altered.

SQL> ALTER INDEX sales_fc_pk STORAGE (FLASH_CACHE NONE);
Index altered.

SQL> ALTER TABLE sales_fc_default STORAGE (FLASH_CACHE DEFAULT);
Table altered.

SQL> ALTER INDEX sales_fc_default_pk STORAGE (FLASH_CACHE DEFAULT);
Index altered.

SQL> ALTER TABLE sales_fc_keep STORAGE (FLASH_CACHE KEEP);
Table altered.

```

```
SQL> ALTER INDEX sales_fc_keep_pk STORAGE (FLASH_CACHE KEEP);
Index altered.
```

Flash Cache Performance Statistics

As usual, Oracle is pretty generous when it comes to providing statistics and instrumentation. A variety of V\$SYSSTAT counters show the activity of the DBFC. [Listing 17.1](#) shows some of the more important counters.

Listing 17.1 DBFC SYSSTAT Counters (flash\_insert\_stats.sql)

[Click here to view code image](#)

```
SQL> WITH stats AS (SELECT /*+ materialize */  
  2          name, VALUE  
  3      FROM v$sysstat  
  4     WHERE name LIKE 'flash cache%')  
 5  SELECT name, VALUE,  
  6         ROUND (VALUE * 100 / tot_inserts, 2) pct_of_inserts  
  7  FROM (SELECT SUM (VALUE) tot_inserts  
  8         FROM stats where name = 'flash cache inserts')  
  9  CROSS JOIN stats  
 10 ORDER BY value DESC  
 11 /
```

| NAME | VALUE | Pct
of
Insert: |
|--|-----------|----------------------|
| flash cache insert skip: exists | 3,224,749 | 551.26 |
| flash cache insert skip: not useful | 317.33 | 1,856,276 |
| flash cache inserts | 584,974 | 100.00 |
| flash cache eviction: aged out | 454,144 | 77.63 |
| flash cache insert skip: DBWR overloaded | 22,202 | 3.80 |
| flash cache insert skip: modification | 1,600 | .27 |
| flash cache eviction: invalidated | 22 | .00 |
| flash cache insert skip: not current | 1 | .00 |
| flash cache eviction: buffer pinned | 0 | .00 |
| flash cache insert skip: | | |

These counters show how often blocks are inserted into the cache, how often they are evicted, and how often the DBWR is overloaded and so does not populate the cache.

Because the DBFC is an extension of the buffer cache, we can see the contents of the cache in the V\$BH view. DBFC contents show up with a status of flashcur or flashcr. [Listing 17.2](#) presents a query that displays the segments whose blocks are cached in buffer cache and/or DBFC.

Listing 17.2 DBFC Contents (flashContents.sql)

[Click here to view code image](#)

```
SQL>   SELECT    owner || '.' || object_name object,
  2          SUM (CASE WHEN b.status LIKE 'flash%' THEN 1 END)
flash_blocks,
  3          SUM (CASE WHEN b.status LIKE 'flash%' THEN 0 else 1 END)
cache_blocks,
  4          count(*) total_blocks
  5      FROM        v$bh b
  6      JOIN
  7            dba_objects
  8      ON (objd = object_id)
  9  GROUP BY    owner, object_name
10  order by 4 desc ;
```

| Object Name | DBFC Blocks | Buf Blocks | Cache Blocks | Tot Cached Blocks |
|--------------------------|-------------|------------|--------------|-------------------|
| OPSG.SALES_FC_KEEP | 70,450 | 1,325 | 71,775 | |
| OPSG.SALES_FC_KEEP_PK | 34,155 | 2,325 | 36,480 | |
| OPSG.SALES_FC_DEFAULT | 13,721 | 969 | 14,690 | |
| OPSG.SALES_FC_DEFAULT_PK | 12,032 | 1,939 | 13,971 | |
| OPSG.SALES_FC_PK | | 3,394 | 3,394 | |
| OPSG.SALES_FC_NONE | | 2,305 | 2,305 | |

Because a read from the flash cache is always a read that would otherwise have had to go to disk, it is possible to calculate how much time is saved by the flash cache. We do this by counting the number of flash cache reads and using the average flash cache latency together with the normal disk latency to calculate the amount of I/O time saved. [Listing 17.3](#) shows the results. In general, if your database would benefit from a larger buffer cache, then it will almost certainly benefit from a DBFC.

Listing 17.3 DBFC Time Saved (flash\_time\_savings.sql)

[Click here to view code image](#)

```

SQL> WITH wait_table
  2      AS (SELECT      SUM (total_waits) total_waits,
  3                  SUM (time_waited_micro) total_time,
  4                  MAX(CASE event
  5                      WHEN 'db file sequential read'
  6                          THEN time_waited_micro /
  7
  8                      END) avg_db_file_micro,
  9
 10                 MAX(CASE event
 11                     WHEN 'db flash cache single block
 12                         physical read'
 13                             THEN time_waited_micro /
 14
 15                     END) avg_db_flash_micro,
 16
 17                 SUM(CASE event
 18                     WHEN 'db flash cache single block
 19                         physical read'
 20                         THEN total_waits
 21
 22                     END) flash_waits
 23
 24                 FROM v$system_event
 25                 WHERE event IN ('db file sequential read',
 26                               'db flash cache single block
 27                         physical read'))
 28     SELECT      total_waits,flash_waits,
 29
 30             ROUND (avg_db_file_micro) avg_disk_time,
 31             ROUND (avg_db_flash_micro) avg_flash_time,
 32             total_time,
 33             ROUND(flash_waits *
 34                   (avg_db_file_micro - avg_db_flash_micro))
 35
 36             time_saved,
 37             ROUND(flash_waits *
 38                   (avg_db_file_micro - avg_db_flash_micro) * 100
 39                   / (total_waits* avg_db_file_micro),2)
 40
 41             pct_io_time_saved
 42
 43     FROM      wait_table
 44
 45

```

| Time | Flash | Cache | Avg | Disk | Avg | DBFC | Total |
|---------------|------------|-----------|-------|-------|-------|-------|-----------|
| Time | | | | | | | |
| Total Waits | | | Waits | time | us | time | |
| us | | | us | | | | |
| ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |
| Time saved | Time saved | Pct of IO | | | | | |
| us | Time saved | | | | | | |
| ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |
| 2,817,955 | 1,555,408 | 1,785 | | | | 90 | 2,393,741 |
| 2,636,254,812 | 52.41 | | | | | | |

Comparing SSD Options

Oracle database administrators have a range of options for taking advantage of the performance benefits offered by SSDs:

- The entire database might be deployed on SSD. For smaller databases, this may be the simplest option. However, this option is often economically impractical for large databases.
- Selected datafiles, tables, indexes, or partitions could be located on SSD. This requires substantial planning and administration but is likely to be very effective if the correct segments are selected.
- For databases that are of a compatible version and operating system type, the Oracle DBFC could be used. This option is simple to implement and can be very effective for read-intensive, index-based workloads.
- The temporary tablespace could be relocated to SSD storage, accelerating the performance of temporary segment I/O, as occurs with disk sort or hashing operations.
- Redo logs are inherently write intensive and have been suggested as candidates for SSD storage. However, as we shall see, the nature of redo log I/O is not a perfect fit for SSD.

In order to assess the likely performance benefits for each of the deployment options, the following sections provide simple performance tests that measure the performance of each configuration under appropriate workloads.

Indexed Reads

This test compares the performance benefits provided by directly storing a segment on flash SSD, on magnetic disk with the DBFC enabled, and with the segment on traditional magnetic disk without the flash cache. The workload consisted of index-based reads against a 20-million-row table using a small (128 M) buffer cache and a 16 GB DBFC. [Figure 17.9](#) summarizes the results.

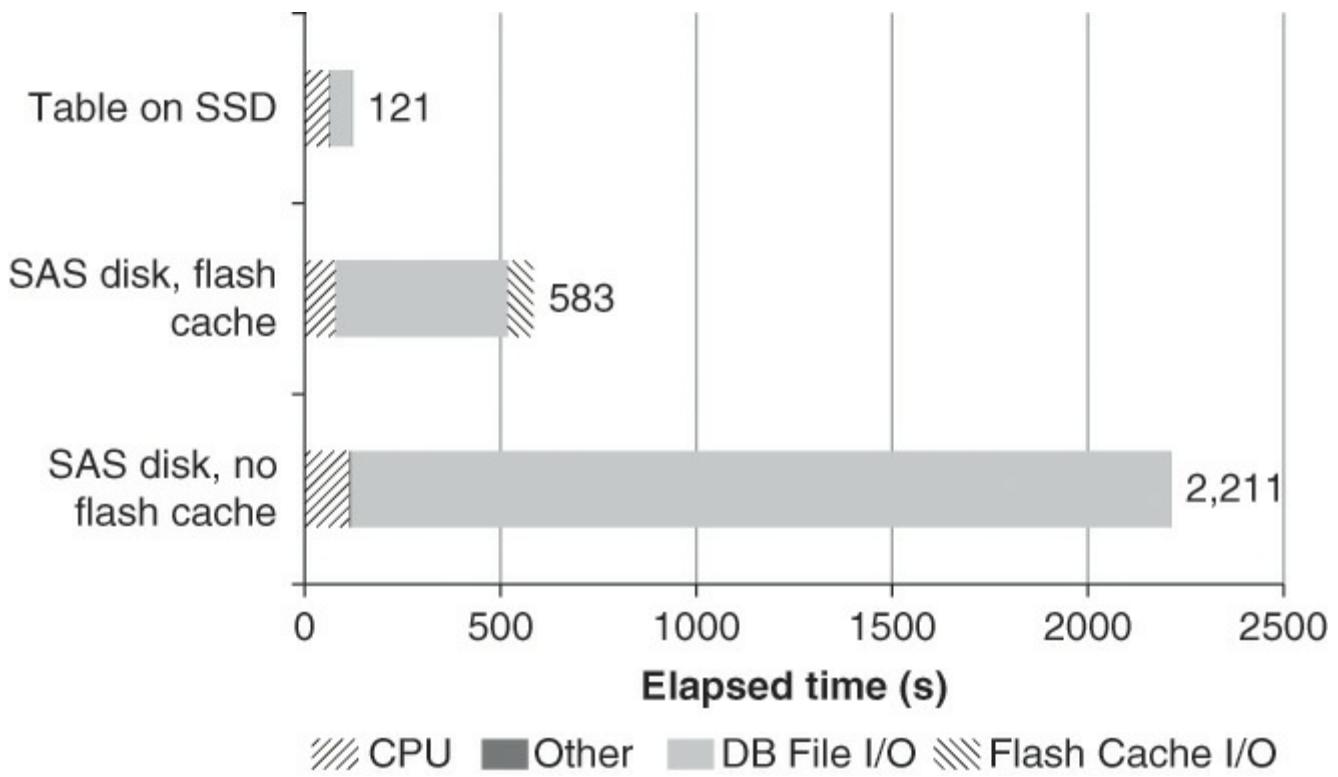


Figure 17.9 Random read performance for SSD and DBFC

Using the DBFC resulted in a significant boost in performance—a reduction of 74 percent in elapsed time—whereas hosting the entire table (and its index) on SSD resulted in a 95 percent reduction.

Of course, you might expect DBFC performance to lag behind that of direct SSD, as the DBFC requires that the block be read from magnetic disk at least once. However, the DBFC does not require that there be sufficient SSD to store the entire segment, so it will be a more practical option in some circumstances.

OLTP Read/Write Workload

In this test, each row was selected twice and updated once. Commits were issued every 10,000 rows. [Figure 17.10](#) shows the results of the test.

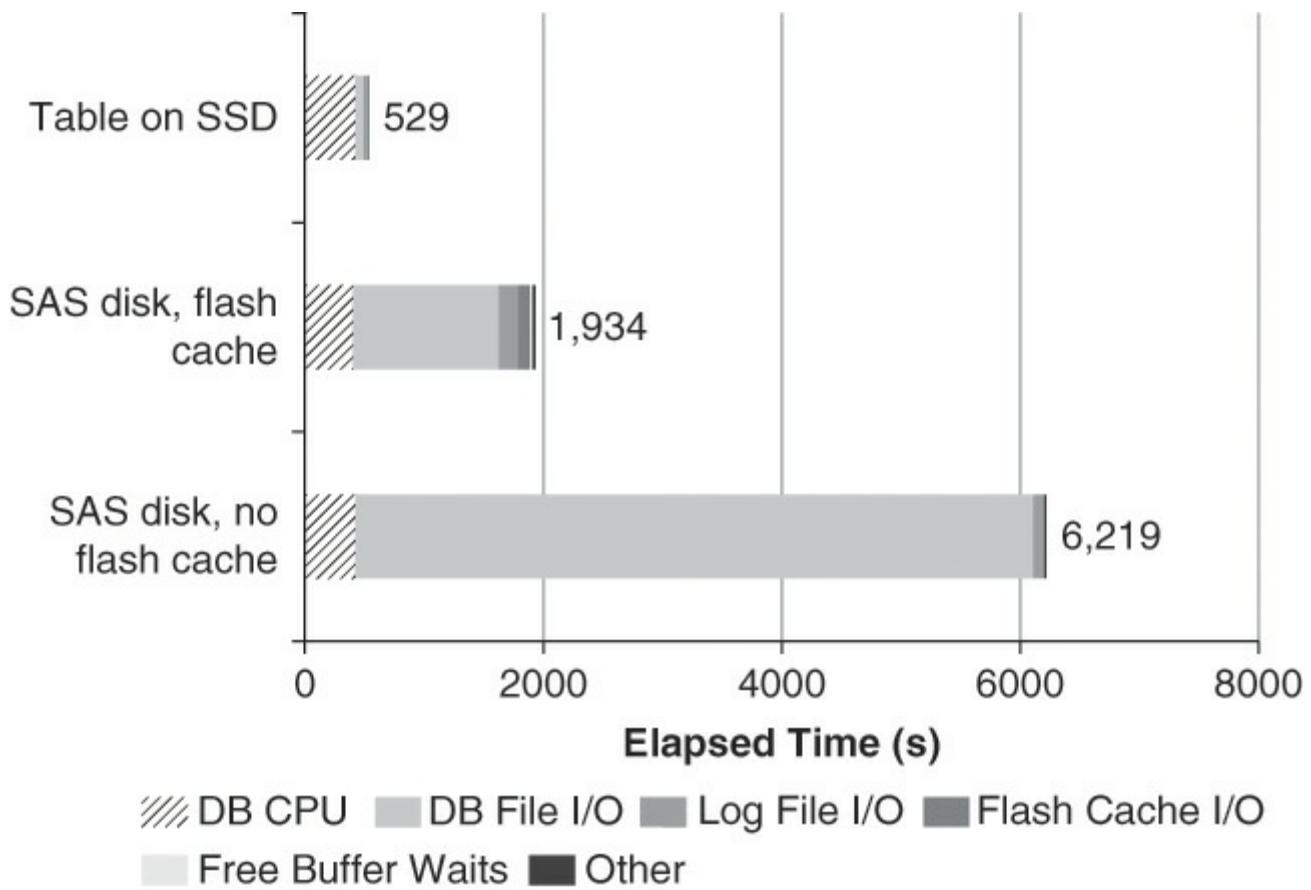


Figure 17.10 Read/write performance test

The performance advantage of SSD is somewhat diminished when writes are involved, particularly for the DBFC. However, the performance advantages were still substantial: 69 percent reduction in elapsed time for the DBFC and 91 percent reduction when directly mounting the segment on SSD.

Full Table Scan Performance

This test measured the time to perform a full table scan against the test table. [Figure 17.11](#) shows the results.

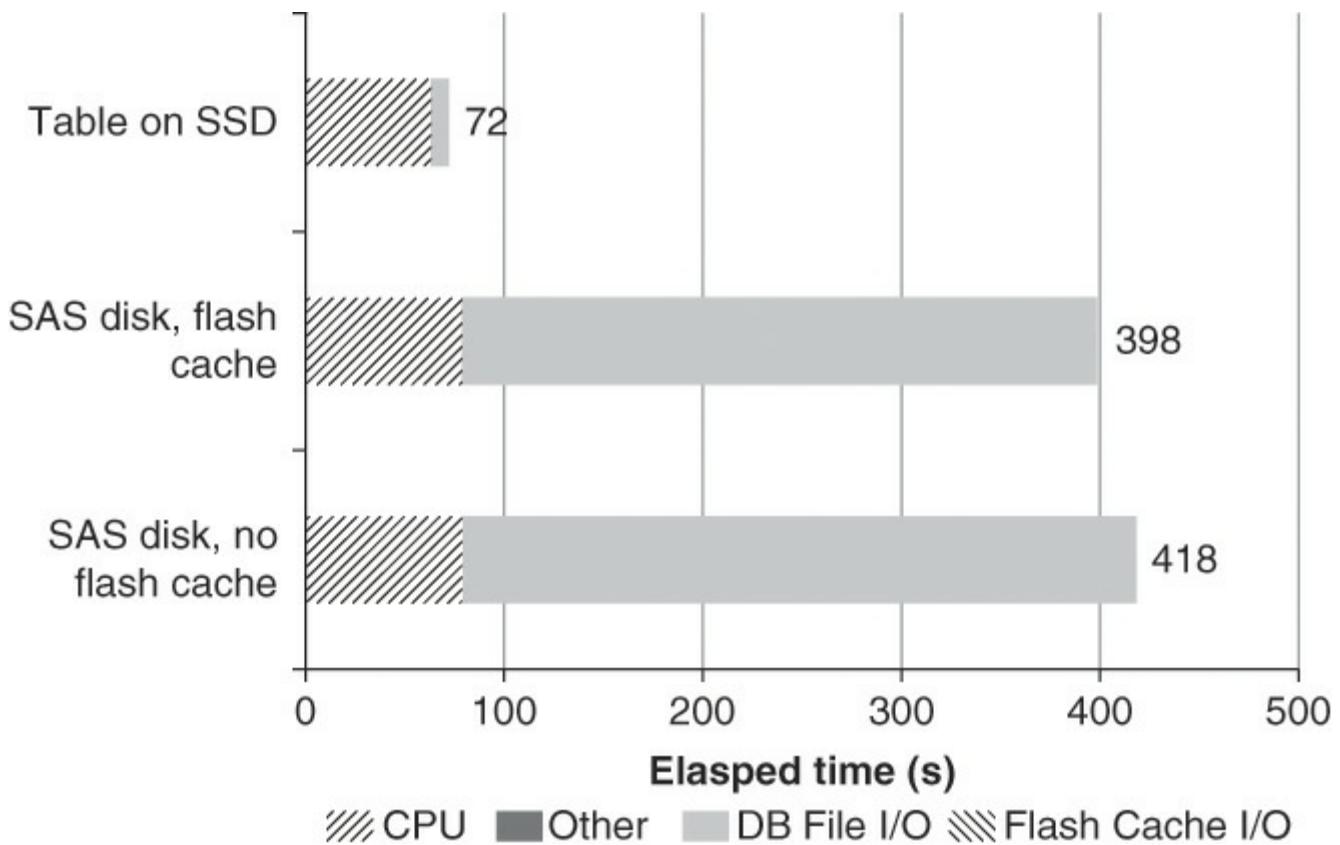


Figure 17.11 Full table scan performance

Not surprisingly, placing the table on SSD resulted in a dramatic improvement in full table scan performance. However, using the DBFC did not result in any measurable performance improvement. From Oracle 11g onward, full table scans typically bypass the buffer cache: the Oracle process reads directly from disk and does not place the blocks thus read in the buffer cache. Since the Database Flash Cache is populated from the buffer cache, full table scans cannot normally benefit from the DBFC.

SSD Native Caches and Full Table Scans

Some SSD vendors offer caching technologies that operate in a similar way to the Oracle DBFC but are independent of Oracle software. These caching layers can accelerate direct path reads from full table scans that bypass the Oracle buffer cache and, therefore, the DBFC. Examples include directCache from Fusion-io and FluidCache from Dell.

[Figure 17.12](#) illustrates this approach using Dell FluidCache as an example. The cache interposes itself between some traditional storage devices and an SSD device. The merged device is presented to the operating system as a single LUN, with the SSD being used to cache frequently accessed data.

- Temp Tablespace
 - Hot Segments
 - Hot Partitions
 - DB Flash Cache
- (limited to the size of the SSD)

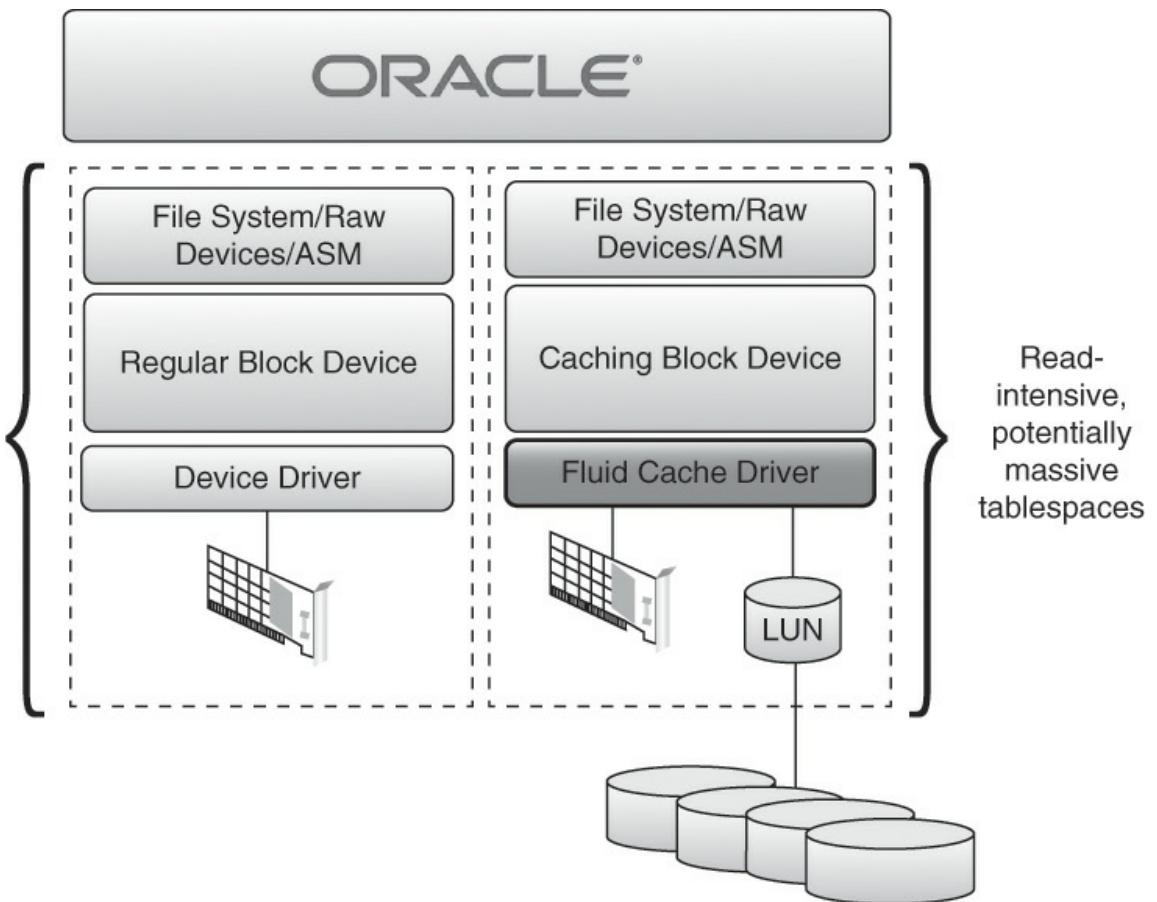


Figure 17.12 SSD native caching

The advantage in an Oracle context is that this form of cache makes no differentiation between direct path I/O and any other I/O. Therefore, this form of caching is capable of accelerating full table scans in a way that the DBFC cannot.

[Figure 17.13](#) illustrates this phenomenon using Fusion-io directCache. When the cache is enabled, second and subsequent scans of a table show acceleration.

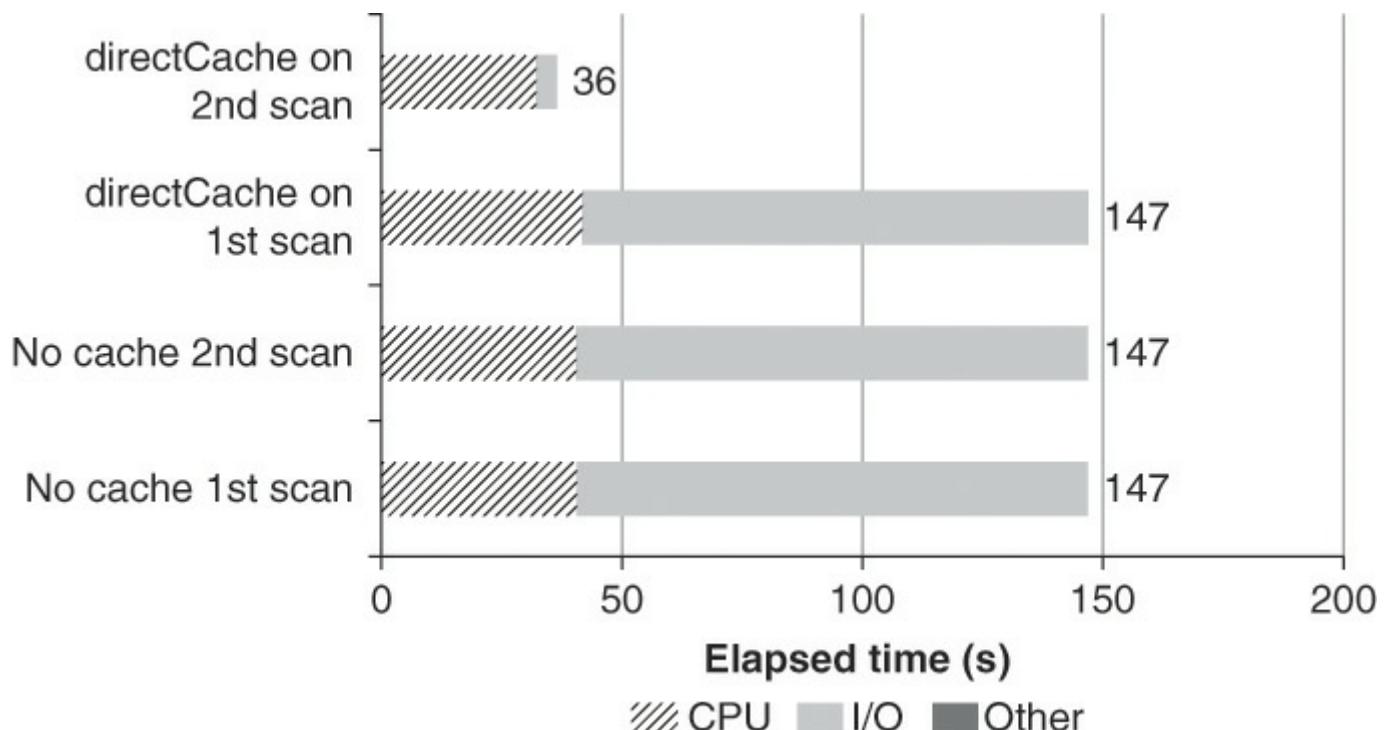
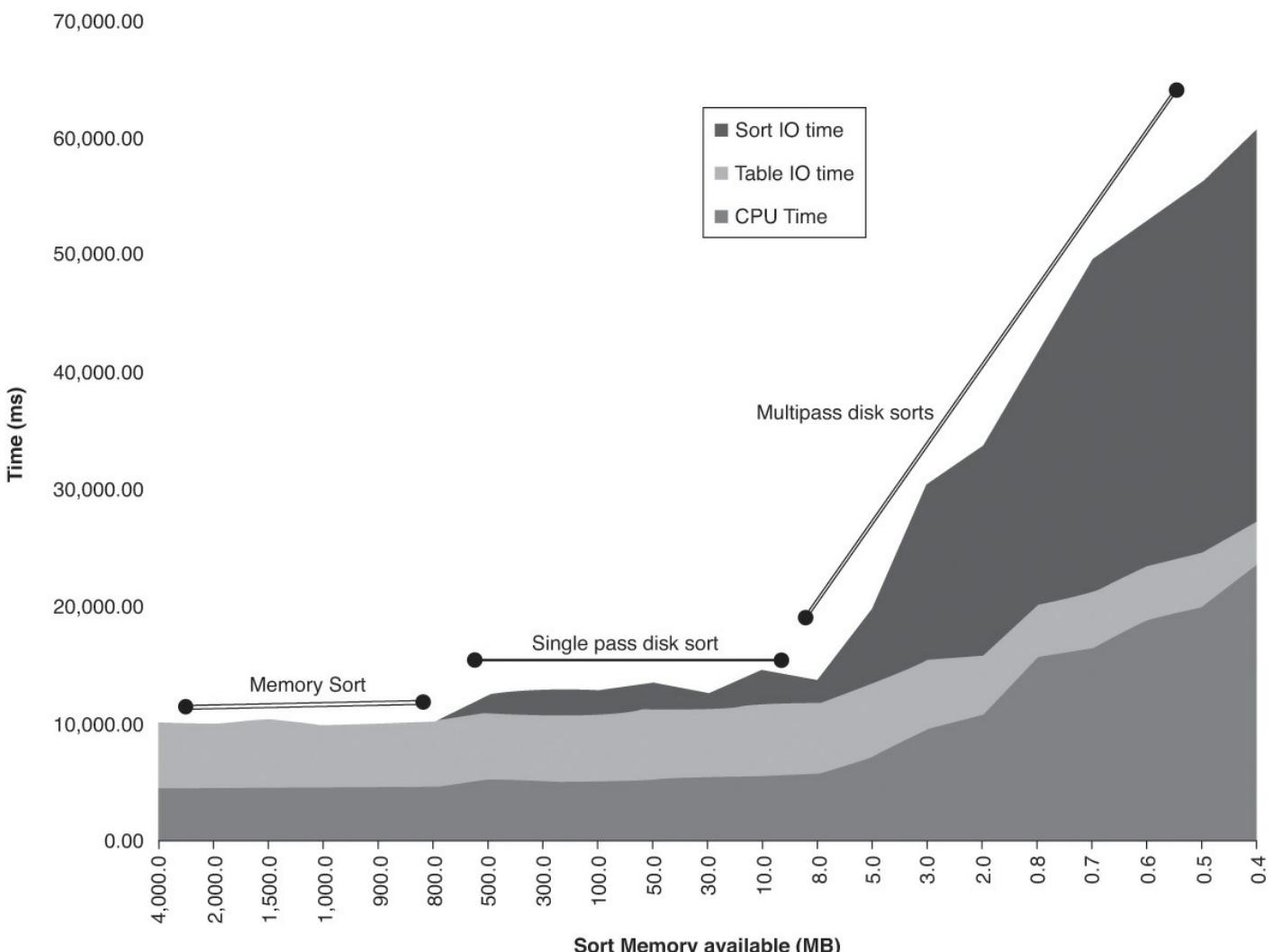


Figure 17.13 Fusion-io directCache accelerating full table scans

Another advantage of these technologies is that they can be used on any operating system, while the DBFC can be used only on Oracle operating systems.

Disk Sort and Hash Operations

Oracle performs I/O to the temporary tablespace when a disk sort or hashing operation occurs—typically as a consequence of ORDER BY or join processing—and there is insufficient program global area (PGA) memory available to allow the join to complete in memory. Depending on the shortfall of memory, Oracle may need to perform single-pass or multipass disk operations. The more passes are involved, the heavier is the overhead of the temporary segment I/O. [Figure 17.14](#) shows what we have come to expect from disk sort performance.



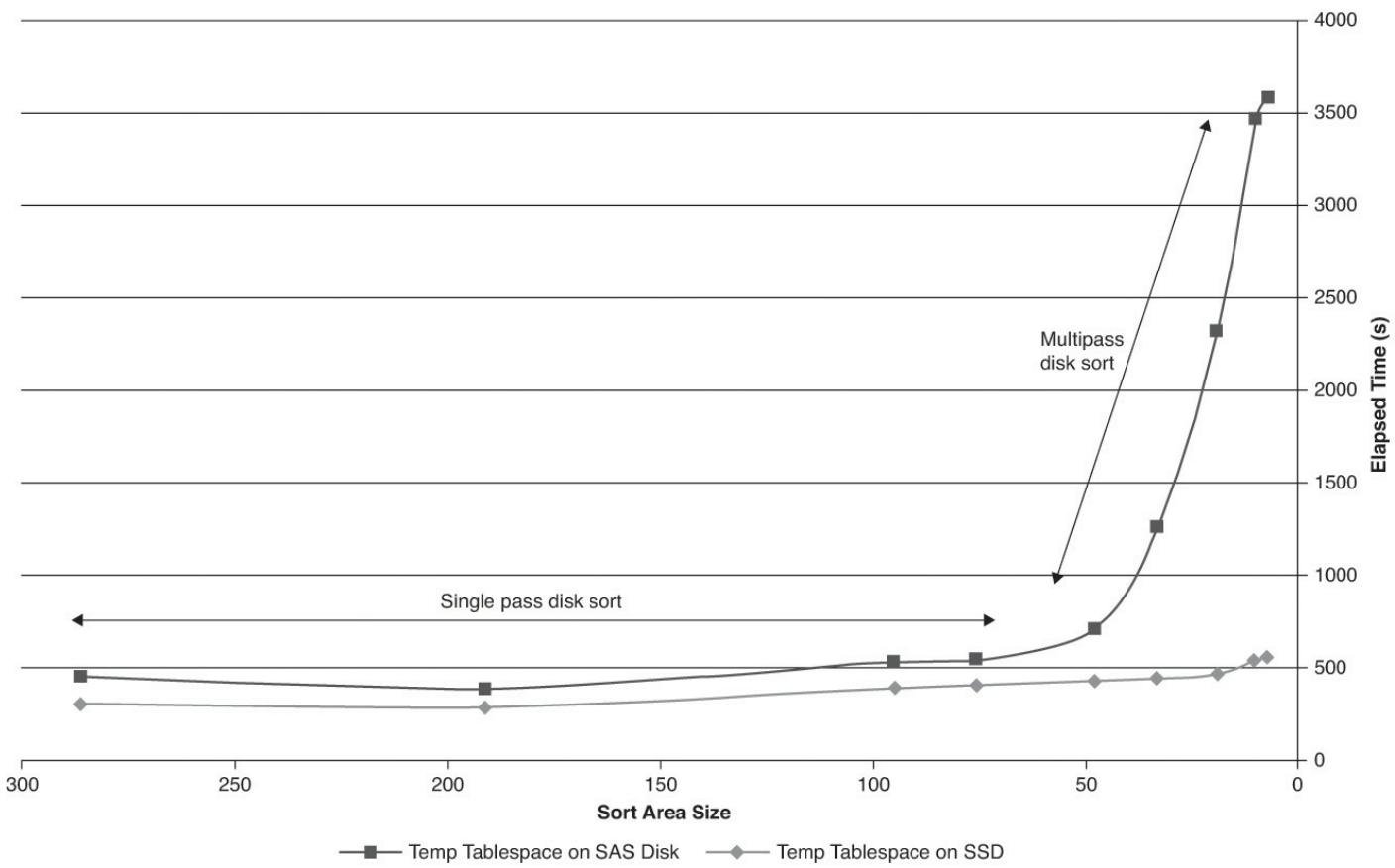
Source: Farooq, T, et al. *Oracle Exadata Expert's Handbook*. New York: Addison-Wesley, 2015.

Figure 17.14 Traditional performance profile for disk sorts

As memory becomes constrained, temporary segment I/O requirements for the operation come to dominate performance and rise sharply as the number of temporary segment passes increase.

When the temporary tablespace is placed on SSD, a completely different performance profile emerges, as shown in [Figure 17.15](#). While the overhead of single-pass sorts are significantly improved, the overhead of multipass disk sorts is drastically reduced. The

more temporary segment passes are involved, the greater is the improvement.



Source: Farooq, T, et al. *Oracle Exadata Expert's Handbook*. New York: Addison-Wesley, 2015.

Figure 17.15 SSD radically reducing the overhead of multipass disk sorts

Redo Log Optimization

The Oracle architecture is designed so that sessions are blocked by I/O requests only when absolutely necessary. The most common circumstance is when a transaction entry must be written to the redo log in order to support a COMMIT. The Oracle session must wait in this circumstance in order to ensure that the commit record has actually been written to disk.

Since redo log I/O is so often a performance bottleneck, many have suggested locating the redo logs on SSD for performance optimization. However, the nature of redo log I/O is significantly different from that of datafile I/O; redo log I/O consists almost entirely of sequential write operations without any seek time for which magnetic disk is well suited, since write throughput is limited only by the rotation of the magnetic disk. In contrast, the sequential write-intensive workload is a worst-case scenario for SSD, since as we have seen, write I/O is far slower than read I/O for SSDs.

[Figure 17.16](#) shows how placing of redo logs on an SSD device in the case of a redo I/O-constrained workload resulted in no measurable improvement in performance. In this case, the SAS drive was able to support a write rate equal to that of the SSD device.

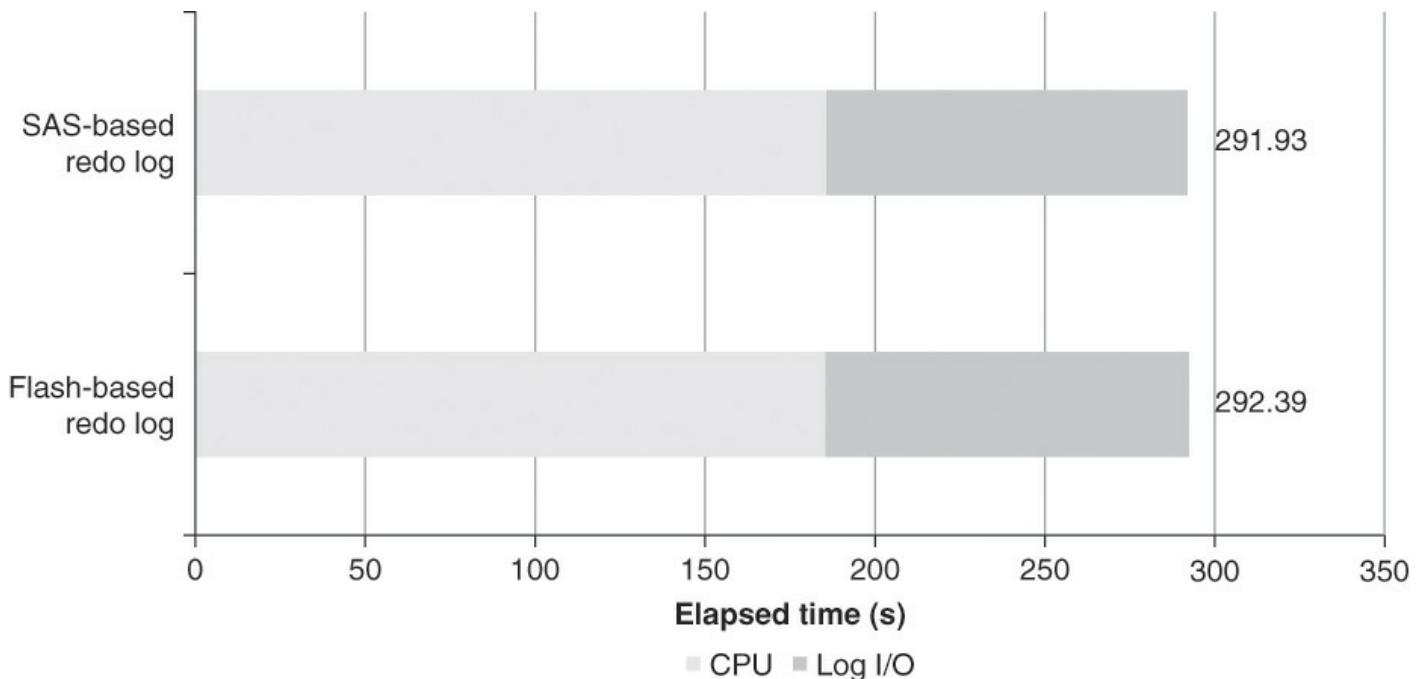


Figure 17.16 Redo log performance on SSD

In more than 15 years of writing on Oracle performance, nothing I've ever presented has caused so much controversy as the results I've presented on redo log performance using SSD. Many people have argued that SSD is a good foundation for redo logs. However, to date, no one has presented data that clearly contradicts the essential point shown in [Figure 17.16](#). Although it is definitely possible to get a slight improvement in performance with very high-end dedicated SSD, we never see the huge improvements that we get with other workloads, such as for index reads or temporary tablespaces. And these results do match with theoretical expectations—we know that sequential write operations are the worst possible type of I/O for SSD and the best possible type of I/O for magnetic disk.

Storage Tiering

Placing a segment on an SSD tablespace can provide a greater optimization and greater predictability than using the DBFC. But one of the killer advantages of the DBFC is that it can optimize tables and indexes that are too large to be hosted on flash disk.

Oracle DBAs must balance two competing demands:

- Increasing data volumes (Big Data) require solutions that provide economical storage for masses of data, which essentially requires systems that incorporate magnetic disk.
- Increasing transaction rates and exponential increases in CPU capacity require solutions that provide economical provision of I/O operations per second (IOPS) and minimize latency. This is the province of SSD and in-memory solutions.

For many or even most databases, the only way to balance both of these trends is to “tier” various forms of storage, including RAM, SSD, and magnetic disk. The Oracle database provides a variety of mechanisms that allow you to move data between the tiers to balance IOPS and storage costs and maximize performance. Following are the

key capabilities that you should consider:

- **Partitioning:** This technique allows an object (table or index) to be stored across various forms of storage and allows data to be moved online from one storage medium to another.
- **Compression:** Compression can be used to reduce the storage footprint (but may increase the retrieval time).
- **Oracle 12 Automatic Database Optimizer (ADO):** ADO allows policy-based compression of data based on activity or movement of segments to alternative storage based on free space.

Using Partitions to Tier Data

Almost any tiered storage solution requires that a table's data be spread across multiple tiers. Typically, the most massive tables contain data that has been accumulated over time. Also, the most recently collected data typically has the greatest activity, while data created further in the past tends to have less activity.

Oracle partitioning allows a table's data to be stored in multiple segments (i.e., partitions), and those partitions can be stored in separate tablespaces. It is therefore the cornerstone of any database storage tiering solution.

A complete discussion of all of the Oracle partitioning capabilities is beyond the scope of this chapter. However, let's consider a scheme that could work to spread the contents of a table across two tiers of storage. The hot tier is stored on a flash-based tablespace, while the cold tier is stored on an HDD-based tablespace.

Interval partitioning allows us to nominate a default partition for new data while selecting specific storage for older data.

[Listing 17.4](#) provides an example of an interval partitioned table. New data inserted into this table is stored in partitions on the SSD\_TS tablespace, while data older than July 1, 2013, is stored on the SAS\_TS tablespace.

Listing 17.4 Interval Partitioned Table

[Click here to view code image](#)

```
CREATE TABLE ssd_partition_demo
(
    id          NUMBER PRIMARY KEY,
    category    VARCHAR2 (1) NOT NULL,
    rating      VARCHAR2 (3) NOT NULL,
    insert_date DATE NOT NULL
)
PARTITION BY RANGE (insert_date)
  INTERVAL ( NUMTOYMINTERVAL (1, 'month') )
  STORE IN (ssd_ts)
(PARTITION cold_data VALUES LESS THAN
```

```
(TO_DATE ('2013-07-01', 'SYYYY-MM-DD'))  
TABLESPACE sas_ts);
```

After some data has been loaded into the table, we can see that new data is stored on the SSD-based tablespace (SSD\_TS), and older data is stored on the SAS based tablespace (SAS\_TS):

[Click here to view code image](#)

```
SQL> 1  
1  SELECT partition_name, high_value, tablespace_name  
2    FROM user_tab_partitions  
3*   WHERE table_name = 'SSD_PARTITION_DEMO'  
SQL> /
```

| PARTITION | HIGH_VALUE | TABLESPACE |
|-----------|---|------------|
| COLD_DATA | TO_DATE(' 2013-07-01 00:00:00', 'SYYYY-M DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA') | SAS_TS |
| SYS_P68 | TO_DATE(' 2013-11-01 00:00:00', 'SYYYY-M DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA') | SSD_TS |
| SYS_P69 | TO_DATE(' 2013-12-01 00:00:00', 'SYYYY-M DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA') | SSD_TS |
| SYS_P70 | TO_DATE(' 2013-10-01 00:00:00', 'SYYYY-M DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA') | SSD_TS |
| SYS_P71 | TO_DATE(' 2013-09-01 00:00:00', 'SYYYY-M DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA') | SSD_TS |
| SYS_P72 | TO_DATE(' 2013-08-01 00:00:00', 'SYYYY-M DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA') | SSD_TS |
| SYS_P73 | TO_DATE(' 2014-01-01 00:00:00', 'SYYYY-M DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA') | SSD_TS |
| SYS_P74 | TO_DATE(' 2014-02-01 00:00:00', 'SYYYY-M DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA') | SSD_TS |

This configuration is initially suitable, but of course as data ages, we would want to move less frequently accessed data from the SSD tablespace to a SAS-based tablespace. To do so, we issue an ALTER TABLE MOVE PARTITION statement.

For instance, the PL/SQL in [Listing 17.5](#) moves all partitions with a HIGH\_VALUE of more than 90 days ago from the SSD\_TS tablespace to the SAS\_TS tablespace.

Listing 17.5 PL/SQL to Move Old Partitions from SSD to SAS Tablespace

[Click here to view code image](#)

```
DECLARE  
  num_not_date      EXCEPTION;  
  PRAGMA EXCEPTION_INIT (NUM_NOT_DATE, -932);  
  invalid_identifier EXCEPTION;  
  PRAGMA EXCEPTION_INIT (invalid_identifier, -904);
```

```

l_highdate           DATE;
BEGIN
  FOR r IN (SELECT table_name,
                  partition_name,
                  high_value
                 FROM user_tab_partitions
                WHERE tablespace_name <> 'SSD_TS')
LOOP
  BEGIN
    -- pull the highvalue out as a date
    EXECUTE IMMEDIATE 'SELECT ' || r.high_value || ' from
dual'
    INTO l_highdate;

    IF l_highdate < SYSDATE - 90
    THEN
      EXECUTE IMMEDIATE
        'alter table '
        || r.table_name
        || ' move partition "''
        || r.partition_name
        || '"' tablespace sas_ts';
    END IF;
  EXCEPTION
    WHEN num_not_date OR invalid_identifier -- max_value not
a date
    THEN
      NULL;
  END;
  END LOOP;
END;

```

In Oracle Database 11g, this operation blocks DML on each partition during the move (or fails with error ORA-00054 if the partition can't be locked). In Oracle Database 12c, you may specify the `ONLINE` clause to allow transactions on the affected partition to continue. After the move, local indexes corresponding to the moved partition and all global indexes are marked as unusable unless you specify the `UPDATE INDEXES` or `UPDATE GLOBAL INDEXES` clause.

The Oracle Database 12c syntax for moving partitions online is simple and effective, whereas in Oracle Database 11g, we can achieve the same result albeit in a more complex manner. Using the `DBMS_REDEFINITION` package, we can create an interim table in the target tablespace, synchronize all changes between that interim table and the original partition, and then exchange that interim table for the original partition.

[Listing 17.6](#) provides an example of using `DBMS_REDEFINITION`. We create a distinct interim table `INTERIM_PARTITION_STORAGE` in the target tablespace `SAS_TS`, which is synchronized with the existing partition `SYS_P86`. When the

`FINISH_REDEF_TABLE` method is invoked, all transactions that may have been applied to the existing partition `SYS_P86` are guaranteed to have been applied to the interim table, and the table is exchanged with the partition concerned. The interim table, which is now mapped to the original partition segment, can now be removed.

Listing 17.6 Using DBMS\_REDEFINITION to Move a Tablespace Online

[Click here to view code image](#)

```
-- Enable/ Check that table is eligible for redefinition
BEGIN
  DBMS_REDEFINITION.CAN_REDEF_TABLE(
    uname      => USER,
    tname      => 'SSD_PARTITION_DEMO',
    options_flag => DBMS_REDEFINITION.CONS_USE_ROWID,
    part_name   => 'SYS_P86');
END;
/
-- Create interim table in the tablespace where we want to move to
CREATE TABLE interim_partition_storage TABLESPACE sas_ts
AS SELECT * FROM ssd_partition_demo PARTITION (sys_p86) WHERE
ROWNUM <1;

-- Begin redefinition
BEGIN
  DBMS_REDEFINITION.START_REDEF_TABLE(
    uname      => USER,
    orig_table => 'SSD_PARTITION_DEMO',
    int_table   => 'INTERIM_PARTITION_STORAGE',
    col_mapping => NULL,
    options_flag => DBMS_REDEFINITION.CONS_USE_ROWID,
    part_name   => 'SYS_P86');
END;
/
-- If there are any local indexes create them here
-- Synchronize
BEGIN
  DBMS_REDEFINITION.SYNC_INTERIM_TABLE(
    uname      => USER,
    orig_table => 'SSD_PARTITION_DEMO',
    int_table   => 'INTERIM_PARTITION_STORAGE',
    part_name   => 'SYS_P86');
END;
/
-- Finalize the redefinition (exchange partitions)
BEGIN
  DBMS_REDEFINITION.FINISH_REDEF_TABLE(
    uname      => USER,
```

```

orig_table => 'SSD_PARTITION_DEMO',
int_table  => 'INTERIM_PARTITION_STORAGE',
part_name   => 'SYS_P86');
END;
/

```

Using DBMS\_REDEFINITION is cumbersome, but generally it is the best approach in Oracle Database 11g when you expect that the partition being moved may be subject to ongoing transactions. In Oracle Database 12c, using the ONLINE clause of MOVE PARTITION is far easier.

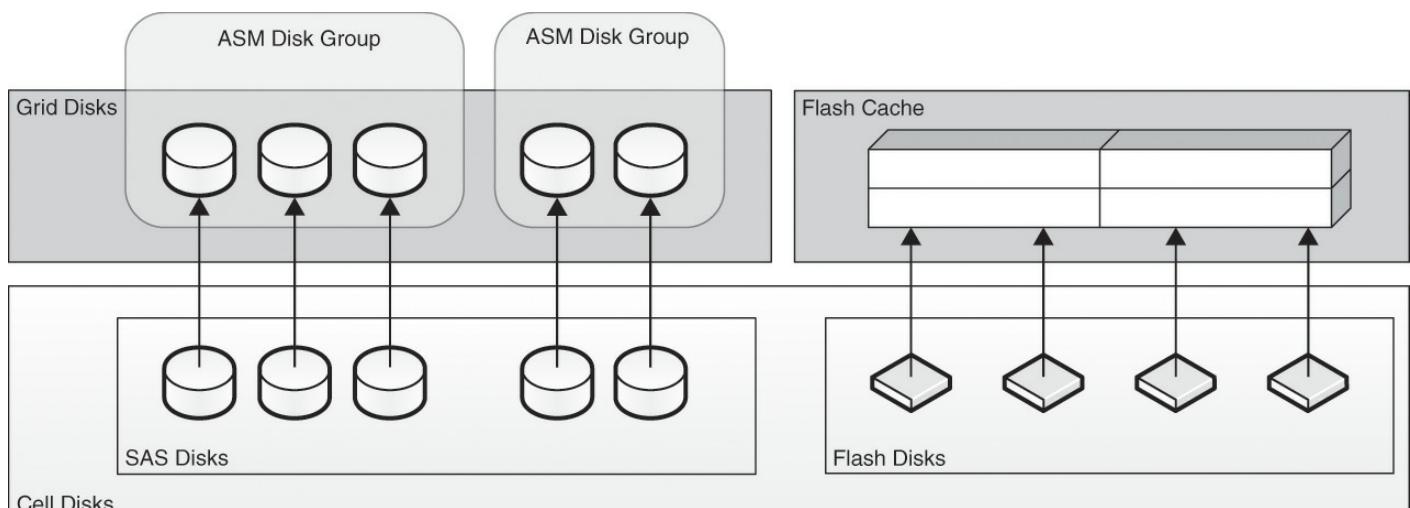
Flash and Exadata

Flash SSD is an important component of Oracle Exadata systems. In this chapter, we have space only for a brief summary of Exadata flash. For a more complete coverage, see [Chapters 15](#) and [16](#) of *Oracle Exadata Expert's Handbook* (Pearson, 2015).

Oracle Exadata systems combine SSD and magnetic disk to provide a balance between economies of storage and high performance. In an Exadata system, flash SSD is contained in the storage cells only. There is no SSD configured within the compute nodes.

Each storage cell contains four PCIe flash SSD drives. The exact configuration depends on the Exadata version. On an X4 system, each cell contains four 800 GB Sun F80 MLC PCI flash cards. That's 3.2 TB of flash per cell and a total of 44.8 TB of flash for a full Exadata rack!

The default configuration for Exadata flash configures all flash storage as Exadata Smart Flash Cache (ESFC). The Exadata Smart Flash Cache is analogous to the Oracle DBFC but has a significantly different architecture. The primary purpose of the Exadata Smart Flash Cache is to accelerate read I/O for database files. This is done by configuring flash as a cache over the grid disks that service datafile read I/O. [Figure 17.17](#) illustrates the mapping on flash disks to Exadata Smart Flash Cache.

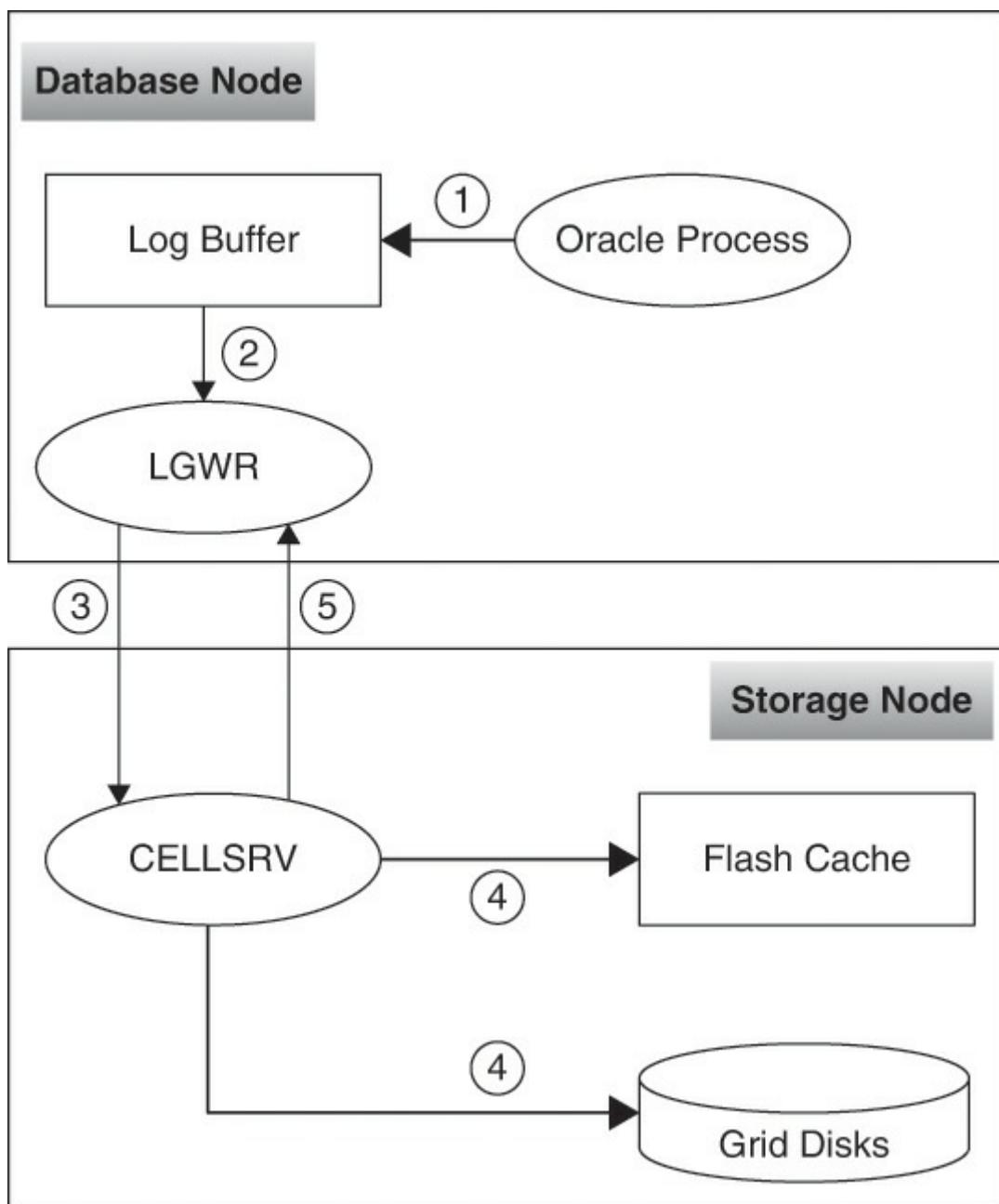


Source: Farooq, T, et al. *Oracle Exadata Expert's Handbook*. New York: Addison-Wesley, 2015.

Figure 17.17 Default mapping of cell and grid disks in Exadata

For suitable OLTP-style workloads, the Exadata Smart Flash Cache provides a fourfold or even better improvement in throughput. By default, the Exadata Smart Flash Cache does not accelerate smart scans or full table scans. You can enable caching of full and smart scans by setting `CELL_FLASH_CACHE KEEP` in the segment `STORAGE` clause. This clause, if set to `NONE`, can completely disable caching for a segment. Although the `KEEP` setting can accelerate scan performance, it does involve some overhead as the cache is loaded and might even degrade overall cache efficiency—so apply it judiciously.

From Oracle Database 11.2.2.4 onward, Exadata Smart Flash Logging allows the flash cache to participate in redo log write operations also. As shown in [Figure 17.18](#), during a redo log write, the Exadata storage server will write to the flash and grid disks simultaneously and return control to the log writer when the first of the two devices completes. This helps alleviate the occasional very high redo write “outliers.”



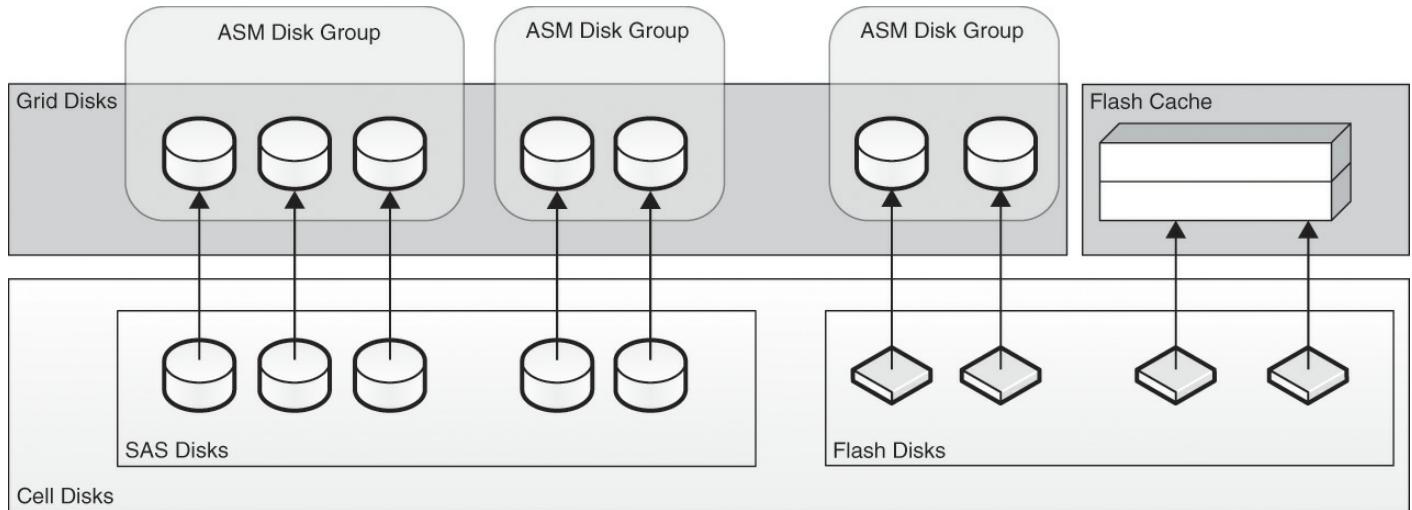
Source: Farooq, T, et al. *Oracle Exadata Expert's Handbook*. New York: Addison-Wesley, 2015.

Figure 17.18 Exadata smart flash logging

From Oracle Database 11.2.3.2.1 onward, the Exadata Smart Flash Cache can become a write-back cache, allowing it to satisfy write requests as well as read requests. In normal circumstances, Oracle sessions do not wait on database writes, but in the event that free buffer waits are experienced as a result of read bandwidth surpassing write bandwidth, the write-back cache can break the bottleneck and accelerate throughput.

Creating Flash-Backed ASM Disk Groups on Exadata

The Exadata Smart Flash Cache makes effective use of the Exadata flash disks for a wide variety of workloads and requires little configuration. However, with a little bit of effort we can create ASM disk groups based entirely on flash storage and use them to selectively optimize hot segments, or we can experiment with placing temporary tablespaces or redo logs on flash storage (see [Figure 17.19](#)).



Source: Farooq, T, et al. *Oracle Exadata Expert's Handbook*. New York: Addison-Wesley, 2015.

Figure 17.19 Defining Exadata flash disks as both flash cache and grid disks

Placing a table directly on flash storage will provide better performance than the Exadata Smart Flash Cache in most circumstances, since it would obviate the need for initial reads from SAS disk to populate the cache, and data will never age out of flash. The degree of improvement in performance will depend on the data access patterns and the size of the segment (or segments). [Figure 17.20](#) shows the results of performance comparison for an OLTP workload with data on HDD only, on HDD with flash cache, and on SSD.

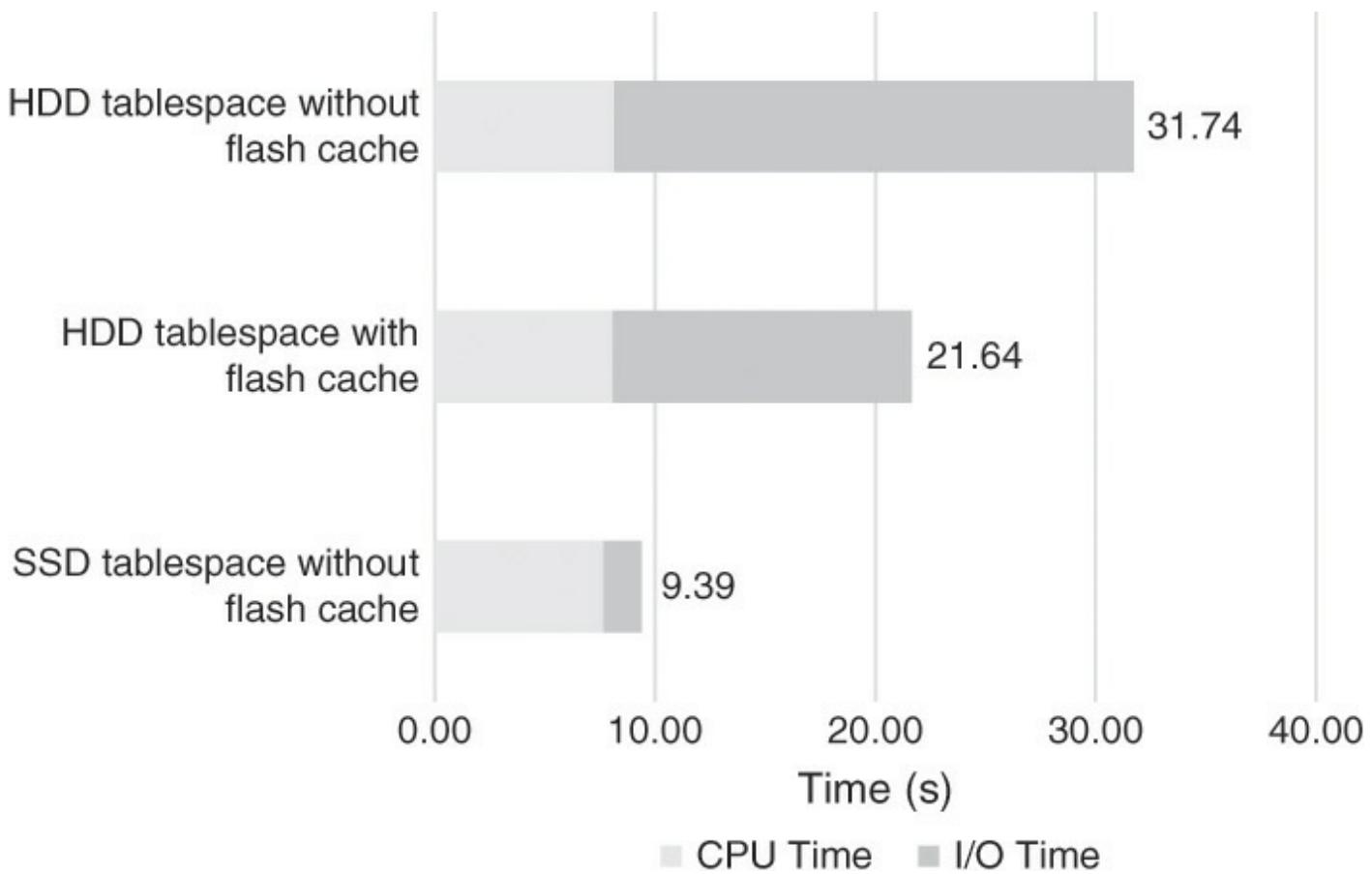


Figure 17.20 Exadata SSD disk group versus Exadata Smart Flash Cache

Smaller segments that are subject to infrequent scans will show the most benefit, since by default the Exadata Smart Flash Cache does not optimize scan I/O. Index reads will also benefit, though the benefit may be marginal when the hit rate in the Exadata Smart Flash Cache is very high.

Redo logs generate significant I/O and can be a transactional bottleneck. However, as we discussed earlier, the I/O patterns of redo activity—sequential write activity—favor spinning magnetic disk over flash SSD.

Summary

This chapter showed how SSDs have revolutionized database performance by providing order of magnitude reduction in disk access times. However, the economics of SSD for mass storage are still not competitive as compared to magnetic disk. Since most databases include both small amounts of frequently accessed *hot* data and large amounts of idle *cold* data, most databases will experience the best economic benefit by combining both solid-state and traditional magnetic disk technologies.

As we learned in this chapter, there are a variety of ways that we can use SSD in Oracle databases:

- Place the entire database on flash—if we can afford it.
- Use the Oracle DBFC to accelerate index read I/O. However, this is possible only if your database is running on an Oracle operating system—Oracle Linux or Solaris.

- Place selected hot segments on a tablespace based on SSD.
- Locate the temporary tablespace on SSD. This approach is attractive if temporary tablespace I/O dominates performance.
- Place redo logs on SSD. However, both theory and observation suggest this solution provides the lowest return on SSD investment.

18. Designing and Monitoring Indexes for Optimal Performance

There is an old saying among carpenters: “If the only tool you have is a hammer, everything starts to look like a nail.” Likewise, a properly chosen and deployed index can be one of the most flexible and sharpest tools in an Oracle DBA’s toolbox, as long as the DBA realizes that an index is not the only tool that may be appropriate for solving query performance issues.

This chapter discusses the different types of indexes that are available for Oracle databases, as well as each index type’s strengths and weaknesses. This chapter also looks at issues that can typically arise when indexes are misused or overused in an Oracle database and how to detect when this is the case. New features in Oracle Database Releases 11g and 12c that may alleviate the dependency on indexes for improved query performance are also covered in this chapter.

Types of Indexes

Indexes are very much like a double-edged razor blade: they are often excellent at cutting through the Gordian knot of query performance tuning when the only other alternative is to retrieve all of the data in a table via a full table scan. However, like any balanced, dual-edged blade, an index tends to cut both ways: it can become quite expensive to maintain, especially when implemented against a table that is updated frequently via an OLTP application workload. And it can become especially deleterious to application performance when deletions are permitted against the index’s underlying data segments.

The Oracle database offers two types of indexes—B-tree indexes and bitmap indexes—and each index type has distinct advantages and drawbacks.

B-Tree Indexes

A *B-tree index* (*balanced tree* index) is typically the most common type of index found in Oracle databases. A B-tree index is extremely efficient at organizing information about the values stored in one or more columns of a table. Moreover, it generally remains in a balanced state even after its underlying data has been modified via data manipulation language (DML) statements—hence its name—and does not require any maintenance to retain that balance except in the most extreme of circumstances.

[Figure 18.1](#) illustrates the basic concepts behind a B-tree index storage structure. As data values are added to the first block (also sometimes called a *page*) of a B-tree index, it continues to fill up until it’s about 50 percent full, at which point it splits into a *root page* and two *leaf* pages. Each leaf contains roughly one-half of the total entries in the index, with the split based on the median values of those present in the index. The root page retains just two entries about where to find the data in the two leaf pages. As

the number of entries in the index continues to grow, this page splitting will continue, and the appropriate entries will be stored initially in the root page until it fills up and subsequently in a new *branch* page that's created when the initial root page's space has been exhausted.

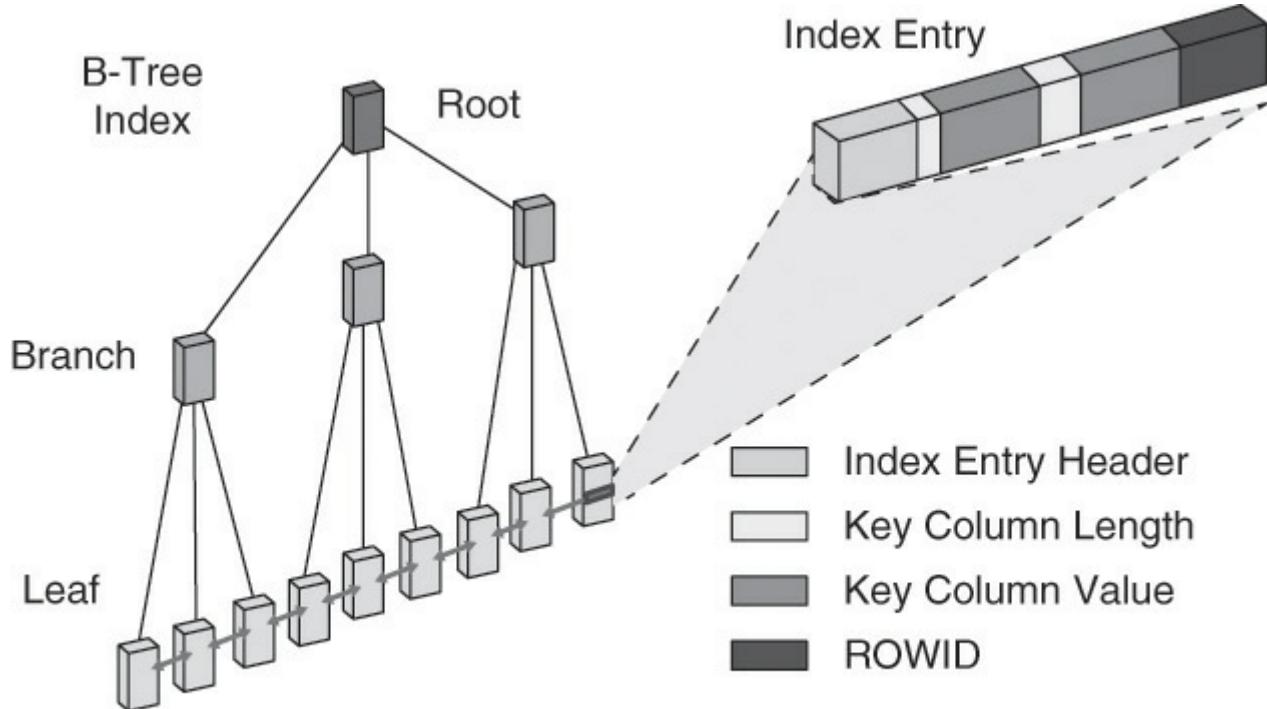


Table data will be retrieved via each ROWID entry

Figure 18.1 Concepts behind the B-tree index

This storage structure usually enables the index to be traversed extremely quickly to locate the particular values being sought after, as is generally the purpose of indexing columns in a table. Subsequently, the ROWIDs of the desired values are captured to access the required data from the table.

B-tree indexes are, of course, useful for other purposes as well:

- They are excellent structures for guaranteeing *uniqueness* of a primary key or unique key constraint.
- If the columns of two row sets being joined are indexed, then it may be much faster to use the indexes themselves as the row sources to look for common members in each set.
- If indexed values are sorted in the same order as the ORDER BY or GROUP BY clause in a query, the index can return the data in sorted order, thereby eliminating the expensive sorting process.
- If all the columns needed to answer a query can be found in the index itself, retrieval from the underlying table may be completely bypassed.

Consider a simple table, AP.RANDOMIZED\_SORTED, with just four columns that have been populated with 10 million rows using DBMS\_RANDOM to generate anonymous data, as shown in [Listing 18.1](#).

Listing 18.1 Constructing a Sample Table for Index Demonstrations

[Click here to view code image](#)

```
DROP TABLE ap.randomized_sorted PURGE;
CREATE TABLE ap.randomized_sorted (
    key_id      NUMBER(8)
    ,key_date   DATE
    ,key_desc   VARCHAR2(32)
    ,key_sts    NUMBER(2) NOT NULL
)
TABLESPACE ado_cold_data
NOLOGGING;

TRUNCATE TABLE ap.randomized_sorted;

SET TIMING ON
DECLARE
    TYPE tnb_KeyID IS
        TABLE OF ap.randomized_sorted.key_id%TYPE
        INDEX BY PLS_INTEGER;
    TYPE tdt_KeyDate IS
        TABLE OF ap.randomized_sorted.key_date%TYPE
        INDEX BY PLS_INTEGER;
    TYPE tvc_KeyDesc IS
        TABLE OF ap.randomized_sorted.key_desc%TYPE
        INDEX BY PLS_INTEGER;
    TYPE tnb_KeySts IS
        TABLE OF ap.randomized_sorted.key_sts%TYPE
        INDEX BY PLS_INTEGER;

    tb_KeyId      tnb_KeyID;
    tb_KeyDate    tdt_KeyDate;
    tb_KeyDesc    tvc_KeyDesc;
    tb_KeySts     tnb_KeySts;
    ctr          PLS_INTEGER;
    nMaxIterations CONSTANT PLS_INTEGER := 10000000;

BEGIN
    -- Populate collections
    FOR ctr IN 1..nMaxIterations
        LOOP
            tb_KeyID(ctr) := ctr;
            tb_KeyDate(ctr) :=
                (TO_DATE('12/31/2014', 'mm/dd/yyyy') -
DBMS_RANDOM.VALUE(1,3650));
            tb_KeyDesc(ctr) := LPAD(' ',DBMS_RANDOM.VALUE(1,32),
SUBSTR('abcdefghijklmnopqrstuvwxyz',DBMS_RANDOM.VALUE(1,26), 1));
            CASE
                MOD(ROUND(DBMS_RANDOM.VALUE(1,nMaxIterations),0),50)
```

```

WHEN 0 THEN tb_Keysts(ctr) :=
30;
WHEN 1 THEN tb_Keysts(ctr)
:=40;
WHEN 2 THEN tb_Keysts(ctr)
:=40;
WHEN 3 THEN tb_Keysts(ctr)
:=40;
WHEN 4 THEN tb_Keysts(ctr)
:=40;
WHEN 5 THEN tb_Keysts(ctr)
:=20;
WHEN 6 THEN tb_Keysts(ctr)
:=40;
WHEN 7 THEN tb_Keysts(ctr)
:=40;
WHEN 8 THEN tb_Keysts(ctr)
:=40;
WHEN 9 THEN tb_Keysts(ctr)
:=30;
WHEN 10 THEN tb_Keysts(ctr)
:=40;
WHEN 11 THEN tb_Keysts(ctr)
:=20;
WHEN 12 THEN tb_Keysts(ctr)
:=10;
WHEN 13 THEN tb_Keysts(ctr)
:=15;
WHEN 14 THEN tb_Keysts(ctr)
:=30;
ELSE tb_Keysts(ctr) :=50;
END CASE;
END LOOP;

-- Load table from collection items
FORALL ctr IN 1..nMaxIterations
    INSERT INTO ap.randomized_sorted(key_id, key_date,
key_desc, key_sts)
        VALUES (tb_KeyID(ctr), tb_KeyDate(ctr), tb_KeyDesc(ctr),
tb_Keysts(ctr));

COMMIT;

EXCEPTION
    WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('Fatal error detected!')
END;
/

```

To enforce a primary key constraint for this table's primary key column (KEY\_ID), a B-tree index, as shown in [Listing 18.2](#), is an excellent choice because each row has a

unique value in the column KEY\_ID.

Listing 18.2 Building a Unique Index for Demonstration Purposes

[Click here to view code image](#)

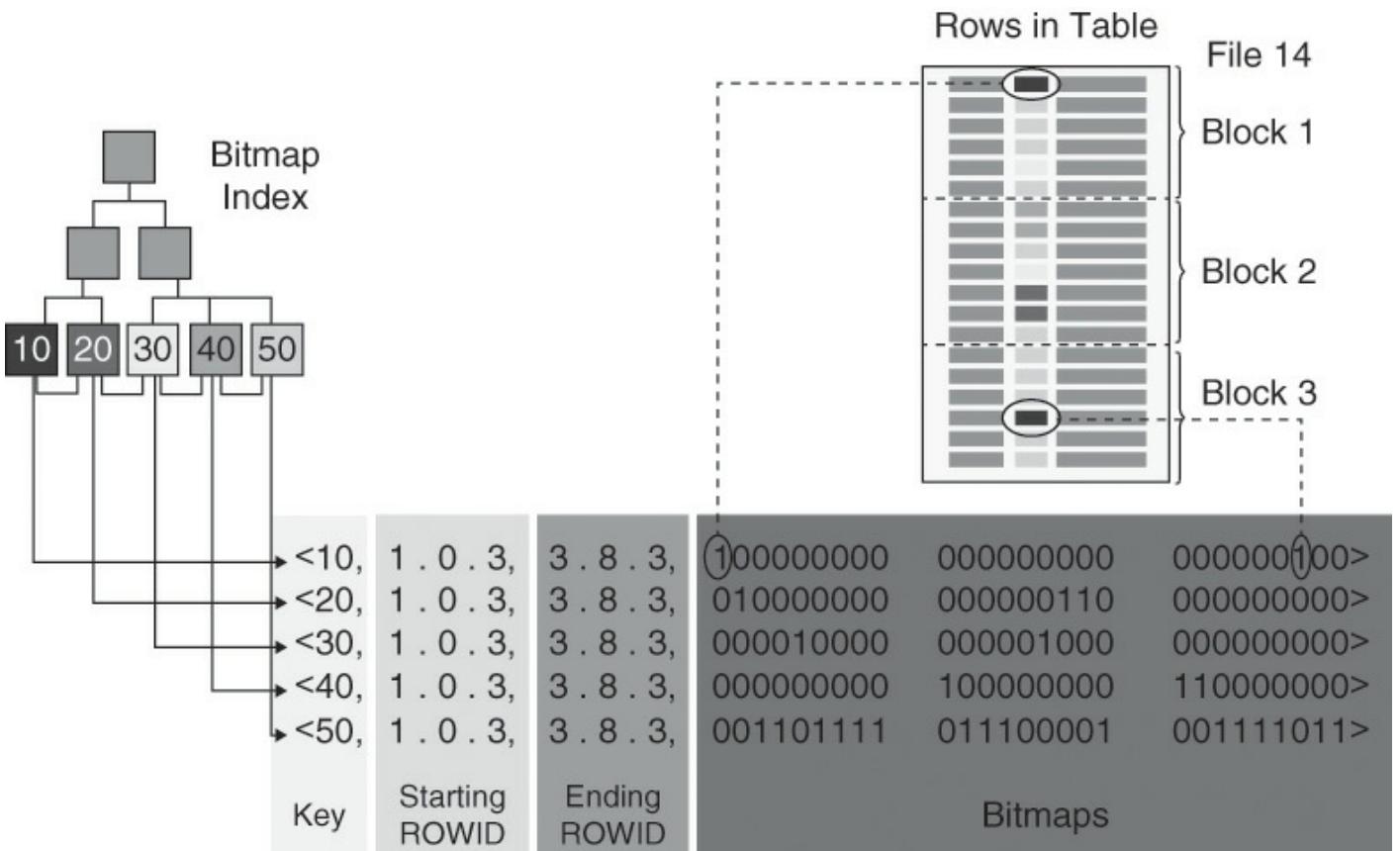
```
ALTER TABLE ap.randomized_sorted
  ADD CONSTRAINT randomized_sorted_pk
  PRIMARY KEY (key_id)
  USING INDEX (
    CREATE UNIQUE INDEX ap.randomized_sorted_pk
    ON ap.randomized_sorted(key_id)
    TABLESPACE ado_cold_idx
    NOLOGGING
    PARALLEL 4);

ALTER INDEX ap.randomized_sorted_pk LOGGING;
```

Note the use of *parallelism* as well as the NOLOGGING clause to avoid writing out any online redo entries during its creation so that this index can be built as quickly as possible. Don't forget to re-enable online redo log generation once the index has been created successfully.

Bitmap Indexes

Though they are typically used less frequently than B-tree indexes, the unique features of *bitmap indexes* are especially valuable for data warehousing application workloads. The easiest way to understand how a bitmap index works is to imagine all the ROWIDs of the corresponding table, laid on their side, along with a series of bitmaps—one for each unique value in the indexed column—arrayed below those ROWIDs. Each bit in a bitmap maps to a ROWID. In each bitmap, the bits mapping to those ROWIDs will be set, which have the value corresponding to the bitmap in the indexed column, as shown in [Figure 18.2](#).



Multiple rows of table data can be filtered via each set of bitmaps

Figure 18.2 Concepts behind the bitmap index

[Listing 18.3](#) shows an example of a bitmap index created on the KEY\_STS column in table AP.RANDOMIZED\_SORTED.

Listing 18.3 Constructing a Bitmap Index for Demonstration Purposes

[Click here to view code image](#)

```

CREATE BITMAP INDEX ap.randomized_sorted_sts_bix
  ON ap.randomized_sorted (key_sts)
  TABLESPACE ado_cold_idx
  NOLOGGING PARALLEL(4);

ALTER INDEX ap.randomized_sorted_sts_bix LOGGING;

```

Because they essentially reference multiple rows simultaneously, bitmap indexes tend to be especially useful for a database that's servicing "read-mostly" data primarily for data warehouse application workloads:

- A bitmap index works best for a column that comprises a small set of relatively static column values because filtering the rows that match selection criteria against that column requires searching the bitmaps only for the corresponding value.
- A table with a very large number of rows—say, 500,000 or more—tends to benefit from a bitmap index because multiple rows can be searched for matching column

values with significantly fewer reads of index blocks than with a B-tree index.

- When individual bitmap indexes are present on two or more columns, selection criteria against those columns can leverage bitmap XAND and XOR operations to quickly filter matching column values.
- Unlike B-tree indexes, a bitmap index can record the presence of a NULL value in its corresponding column; therefore, the IS NOT NULL operator can leverage a bitmap index without having to read all values in the underlying table.
- Bitmap indexes can take advantage of parallel DML during data loading, so they are perfect for data warehouses that are typically loaded during longer-running batch operations.

However, the same features that make the bitmap index so powerful for data warehouses are also its Achilles heel, especially for heavily updated data as well as online transaction processing (OLTP) application workloads:

- First and foremost, because they *reference* multiple rows simultaneously, bitmap index segments must be *updated* simultaneously too. Unfortunately, this means that the bitmap segments as well as the table rows they index *must* be locked while the index is reconstructed.
- Bitmap indexes are not optimal for enforcing a PRIMARY KEY or UNIQUE KEY referential integrity constraint.
- Likewise, a column that contains constantly changing values or whose domain grows constantly would be a poor choice for a bitmap index.
- A table with a relatively small number of rows—say, less than 500,000—will not likely benefit enough from a bitmap index to justify the maintenance costs, which are much higher compared to that of a B-tree index.
- Eventually, every bitmap entry must be *reconverted* into its corresponding ROWID. Although this takes virtually no time to complete, it obviously impacts performance when millions of entries need to undergo conversion.

Partitioned Indexes

When thinking about partitioned tables in an Oracle database, it's also important to understand the difference between *local partitioned* indexes and *global partitioned* indexes. Although these aren't really different index types—they can only be either B-tree or bitmap indexes—each flavor of partitioned index offers distinct advantages and drawbacks.

Local Partitioned Indexes

As its name suggests, a *local* partitioned index exists only within the scope of a single table partition and indexes only the column values within that partition. Since the data values to be retrieved exist only within a few partitions, local indexes can be used to filter, sort, and retrieve data only within those partitions whenever the Oracle Optimizer chooses to leverage *partition pruning*. Using a local partitioned index usually results in

much fewer physical reads to locate data.

Global Partitioned Indexes

Unlike a local partitioned index, a *global* partitioned index spans all existing partitions of a partitioned table. It is therefore useful in situations when the Oracle Optimizer determines that a majority of the table partitions must be scanned to retrieve pertinent data values, and thus the global index is a better choice than searching all the local partitioned indexes to locate data.

[Listing 18.4](#) shows the creation of table AP.RANDOMIZED\_PARTED, which comprises four partitions.

Listing 18.4 Creating an Example Partitioned Table

[Click here to view code image](#)

```
DROP TABLE ap.randomized_parted PURGE;
CREATE TABLE ap.randomized_parted (
    key_id      NUMBER(8)
    ,key_date   DATE
    ,key_desc   VARCHAR2(32)
    ,key_sts    NUMBER(2) NOT NULL
)
PARTITION BY RANGE(key_date) (
    PARTITION p1_frigid
        VALUES LESS THAN (TO_DATE('2010-01-01','yyyy-mm-dd'))
        TABLESPACE ado_cold_data
    ,PARTITION p2_cool
        VALUES LESS THAN (TO_DATE('2013-01-01','yyyy-mm-dd'))
        TABLESPACE ado_cool_data
    ,PARTITION p3_warm
        VALUES LESS THAN (TO_DATE('2014-01-01','yyyy-mm-dd'))
        TABLESPACE ado_warm_data
    ,PARTITION p4_hot
        VALUES LESS THAN (TO_DATE('2014-07-01','yyyy-mm-dd'))
        TABLESPACE ado_hot_data
    ,PARTITION p5_radiant
        VALUES LESS THAN (MAXVALUE)
        TABLESPACE ap_data
)
NOLOGGING
PARALLEL 4
;

INSERT /*+ APPEND NOLOGGING PARALLEL(4) */ INTO
ap.randomized_parted
SELECT * FROM ap.randomized_sorted;

COMMIT;
```

```

ALTER TABLE ap.randomized_parted
    DROP CONSTRAINT randomized_parted_pk;

DROP INDEX ap.randomized_parted_pk;

ALTER TABLE ap.randomized_parted
    ADD CONSTRAINT randomized_parted_pk
        PRIMARY KEY (key_id)
    USING INDEX (
        CREATE UNIQUE INDEX ap.randomized_parted_pk
            ON ap.randomized_parted(key_id)
            TABLESPACE ap_idx
            NOLOGGING
            PARALLEL 4
    );

```

Table created.

[Listing 18.5](#) illustrates how to create a series of local indexes on column KEY\_STS as well as a single global partitioned index on column KEY\_DATE.

Listing 18.5 Example of Local and Global Partitioned Indexes

[Click here to view code image](#)

```

-----
-- Create local partitioned indexes on KEY_STS
-----

DROP INDEX ap.randomized_parted_loc_sts;
CREATE INDEX ap.randomized_parted_loc_sts
    ON ap.randomized_parted (key_sts)
    STORAGE (INITIAL 10M)
    LOCAL (
        PARTITION lx1_frigid      TABLESPACE ado_cold_idx
        ,PARTITION lx2_cool       TABLESPACE ado_cool_idx
        ,PARTITION lx3_warm       TABLESPACE ado_warm_idx
        ,PARTITION lx4_hot        TABLESPACE ado_hot_idx
        ,PARTITION lx5_radiant    TABLESPACE ap_idx
    );

-----
-- Create a global partitioned index on KEY_DESC
-----

CREATE INDEX ap.randomized_parted_glb_desc
    ON ap.randomized_parted (key_desc)
    STORAGE (INITIAL 10M)
    GLOBAL;
```

Partial Indexes

Starting with Oracle Database Release 12.1.0.1, it's now possible to create indexes for only some of the partitions of a partitioned table. For example, consider table AP.RANDOMIZED\_PARTED, shown earlier in [Listing 18.5](#). There may be very little advantage to creating local indexes on column KEY\_STS in the oldest partitions because queries hardly ever reference these partitions; additionally, even when the oldest data is referenced, it's usually going to be accessed via a full table partition scan anyway because the majority of the data is likely to be in an active state. The new INDEXING {ON|OFF} clause of the CREATE TABLE and ALTER TABLE statements makes this possible, as shown in the reconstructed data definition language (DDL) for this table in [Listing 18.6](#).

Listing 18.6 Example of Partial Indexing Schemes in Oracle 12c Release 1 (12.1.0.1)

[Click here to view code image](#)

```
CREATE TABLE ap.randomized_parted (
    key_id      NUMBER(8)
    ,key_date   DATE
    ,key_desc   VARCHAR2(32)
    ,key_sts    NUMBER(2) NOT NULL
)
INDEXING OFF
PARTITION BY RANGE(key_date) (
    PARTITION p1_frigid
        VALUES LESS THAN (TO_DATE('2010-01-01','yyyy-mm-dd'))
        TABLESPACE ado_cold_data
    ,PARTITION p2_cool
        VALUES LESS THAN (TO_DATE('2013-01-01','yyyy-mm-dd'))
        TABLESPACE ado_cool_data
    ,PARTITION p3_warm
        VALUES LESS THAN (TO_DATE('2014-01-01','yyyy-mm-dd'))
        TABLESPACE ado_warm_data
    INDEXING ON
    ,PARTITION p4_hot
        VALUES LESS THAN (TO_DATE('2014-07-01','yyyy-mm-dd'))
        TABLESPACE ado_hot_data
    INDEXING ON
    ,PARTITION p5_radiant
        VALUES LESS THAN (MAXVALUE)
        TABLESPACE ap_data
    INDEXING ON
)
NOLOGGING
PARALLEL 4;
```

In this case, local indexes would not be created by default for any table partitions at

all unless the `INDEXING` attribute is set to `ON` for each individual partition. In addition, if a global index were to be created on the `KEY_STS` column for this table, by default that index would not include any values from the first two partitions.

Other Index Types

There are a few other index types and subvariants that are beyond the bounds of this discussion, but they definitely deserve mention, as you will almost certainly encounter them during your Oracle DBA career.

Index Organized Tables

Index-organized tables (IOTs) are truly indexes, but they have many of the same attributes of a standard heap-organized table. When a majority of a table's columns need to be indexed to enforce a `PRIMARY KEY` constraint, an IOT fulfills this need quite nicely.

Another advantage to an IOT is that it eliminates the required retrieval of rows from the table it would normally index, so it's not uncommon to see the physical I/O required to retrieve data dropping by a significant margin; IOTs are therefore not uncommon in OLTP applications. Data within IOTs can be indexed with secondary indexes as well.

Reverse-Key Indexes

When a table's `PRIMARY KEY` constraint is based on a single column that contains a monotonically increasing value—for example, `ORDER_ID` in the `OE.ORDERS` table (part of the standard Oracle sample schemas)—index entries for similar values tend to be located within just a few index blocks. While this arrangement does help the clustering factor of the index, it can have an extremely deleterious effect on OLTP application performance, especially in an Oracle Real Application Cluster (RAC) database.

A reverse-key index simply stores the values in *reverse byte order*—for example, 1001 is stored as 1001, but 1002 is stored as 2001, 1003 is stored as 3001, and so forth—so this tends to spread out the values among multiple index blocks. The end result is that two different user sessions will not contend with each other when attempting to add a new entry into the same index simultaneously during heavy OLTP application activity, thus alleviating a `TX` wait enqueue for that index.

Composite Indexes

If an index contains more than one column, it's termed a *composite* or *concatenated* index. If a query's selection criteria includes all or leading columns of the columns in a composite index, it's likely to speed the execution of that query. However, it's also possible to construct a composite index so that the columns with the lowest number of unique values are listed first in the index—for example, `COUNTRY`, `STATE`, and `CITY`—so that a query that specifies only values for `STATE` and `CITY` in its selection criteria could still take advantage of that index via a *skip-scan* operation, which essentially

bypasses the leading column in the index (COUNTRY) and accesses the secondary columns.

Compressed Indexes

The child table in a parent–child relationship typically requires a multicolumn PRIMARY KEY constraint to enforce referential integrity. For example, the OE.ORDER\_ITEMS table that's part of the standard Oracle sample schemas has a two-column primary key for columns ORDER\_ID and LINE\_ITEM\_ID. It's not uncommon for the child table to have a multiplicity of LINE\_ITEM\_ID values per ORDER\_ID; therefore, to save space in an index for this constraint, the index can be *compressed* so that the value for ORDER\_ID is stored just once as a prefix within the index page for all LINE\_ITEM\_ID values, as shown in [Listing 18.7](#).

Listing 18.7 Compressing an Index

[Click here to view code image](#)

```
ALTER TABLE oe.order_items DROP CONSTRAINT order_items_pk;
ALTER TABLE oe.order_items
  ADD CONSTRAINT order_items_pk
    PRIMARY KEY (order_id, line_item_id)
    USING INDEX (
      CREATE UNIQUE INDEX oe.order_items_pk_idx
        ON oe.order_items (order_id, line_item_id)
        TABLESPACE example
      COMPRESS);
```

Bitmap Join Indexes

The bitmap join index is a special subtype of bitmap index that is useful when the columns that are going to be used for joins within a subset of tables—for example, a fact table and several of its corresponding dimension tables—are already known. A bitmap join index essentially stores the ROWIDs of all the values that already matched between the fact and dimension tables (see [Listing 18.8](#)). The advantage of this flavor of bitmap index is that it's extremely quick to search for matched sets of column values, as it essentially offloads the processing of the joined values to the index itself.

Listing 18.8 Storing ROWIDs in a Bitmap Join Index

[Click here to view code image](#)

```
CREATE BITMAP INDEX oe.order_item_qty_bjx
  ON oe.order_items (quantity)
    FROM oe.order_items OI, oe.product_information P
    WHERE OI.product_id = P.product_id
  TABLESPACE example;
```

Multiple Indexes on Identical Columns

Prior to Oracle Database 12c Release 1, only one index could be created on the same column or combination of columns. But starting with Release 12.1.0.1, it's now possible to create multiple indexes on the same column or set of columns, so that the statements that create the new bitmap index shown in [Listing 18.3](#) would actually execute correctly without having to drop the existing B-tree index first. Using multiple indexes on identical columns offers the following possibilities:

- A *unique* index as well as a *nonunique* index can be created for the same set of columns.
- A B-tree and a bitmap index can be created on the same columns.
- Indexes with different partition schemes (for example, a local partitioned index and a global partitioned index) can co-exist for the same columns.
- Partitioned indexes are also permitted on the same columns, but with different partitioning methods (for example, one that is *hash-partitioned* and another that is *range-partitioned*).

Note that there are some restrictions on this new feature, however:

- A B-tree *non-clustered* index and a B-tree *clustered* index cannot be created on the same column.
- A B-tree index and an IOT cannot co-exist on the same column.
- When multiple indexes are created on the same column, only one of the indexes can be visible at a time, and the other indexes must have a status of `INVISIBLE`. However, it's possible to inform the optimizer that it is permitted to leverage the non-default index by setting the `OPTIMIZER_USE_INVISIBLE_INDEX` initialization parameter to `TRUE` at either the session or system level.

Index Performance Issues

Even when an Oracle DBA has done her homework and has chosen the proper type of index for the job, indexes can present some unique challenges for tuning application performance.

Index Statistics

The Oracle query optimizer leverages index statistics when it makes its decision about whether to use an index instead of a full table scan. If there are multiple indexes to choose from, the optimizer also uses these statistics to determine which index is most efficient. A simple query against the `DBA_INDEXES` data dictionary view reveals the most crucial statistics, as [Listing 18.9](#) shows.

Listing 18.9 Crucial Index Statistics

[Click here to view code image](#)

```

SET LINESIZE 170
SET PAGESIZE 20000
COL owner          FORMAT A08          HEADING "Owner"
COL index_owner    FORMAT A08          HEADING "Owner"
COL table_name     FORMAT A24          HEADING "Table Name"
COL index_name     FORMAT A30          HEADING "Index Name"
COL partition_name FORMAT A12          HEADING "Partition Name"
COL tsp_name       FORMAT A12          HEADING "Tablespace|Name"
COL status         FORMAT A08          HEADING "Status"
COL num_rows       FORMAT 999,999,999  HEADING "Row|Count"
COL distinct_keys  FORMAT 999,999,999  HEADING "Distinct|Keys"
COL clusfctr      FORMAT 99,999,999   HEADING "Clust|Factor"
COL blevel         FORMAT 99999        HEADING "Index|Depth"
COL leaf_blocks   FORMAT 9,999,999    HEADING "Leaf|Blocks"
COL avg_lbpk      FORMAT 999,999      HEADING "Avg Leaf|Blks/Key"
COL avg_dbpk      FORMAT 999,999      HEADING "Avg Data|Blks/Key"
TTITLE "Index Performance Metadata (from DBA_INDEXES)"
SELECT
    owner
    ,table_name
    ,index_name
    ,tablespace_name tsp_name
    ,status
    ,num_rows
    ,distinct_keys
    ,clustering_factor clusfctr
    ,blevel
    ,leaf_blocks
    ,avg_leaf_blocks_per_key avg_lbpk
    ,avg_data_blocks_per_key avg_dbpk
FROM dba_indexes
WHERE owner = 'AP'
ORDER BY 1,2,3
;
TTITLE OFF
Index Performance Metadata
DBA_INDEXES)

```

| Tablespace | | | | |
|-------------------|------------------|----------------------|---------|--------------|
| Avg Leaf Avg Data | Owner Table Name | Index | | |
| Name | Name | Status | Count | Keys |
| Blks/Key | Blks/Key | | | |
| ----- | ----- | ----- | ----- | ----- |
| ----- | ----- | ----- | ----- | ----- |
| AP EST_SALES | | EST_SALES_CITYST_IDX | | ADO_COLD_IDX |
| VALID 680,352 | | 29,768 | 526,867 | 2 1,124 |

| | | | | | |
|-------|---------------------|------------|-------------------------|--------------|--------------|
| AP | EST_SALES | | EST_SALES_CITY_IDX | | ADO_COLD_IDX |
| VALID | 680,352 | 18,792 | 526,845 | 2 | 1,948 |
| AP | EST_SALES | | EST_SALES_PK_IDX | | ADO_COLD_IDX |
| VALID | 680,352 | 680,352 | 74,988 | 2 | 1,519 |
| AP | EST_SALES | | EST_SALES_STATE_IDX | | ADO_COLD_IDX |
| VALID | 680,352 | 62 | 89,808 | 2 | 1,047 |
| AP | EST_SALES | | EST_SALES_TIMEZONE_IDX | | ADO_COLD_IDX |
| VALID | 649,680 | 26 | 82,138 | 2 | 1,001 |
| AP | EST_SALES | | EST_SALES_ZIPCITYST_IDX | | ADO_COLD_IDX |
| VALID | 680,352 | 42,522 | 680,352 | 2 | 1,193 |
| AP | EST_SALES | | EST_SALES_ZIPCODE_IDX | | ADO_COLD_IDX |
| VALID | 680,352 | 42,264 | 680,352 | 2 | 1,118 |
| AP | INVOICES | | INVOICES_CUST_IDX | AP_IDX | VALII |
| AP | INVOICES | | INVOICES_PK_IDX | AP_IDX | VALII |
| AP | INVOICE_ITEMS | | INVOICE_ITEMS_PK_IDX | AP_IDX | VALII |
| AP | INVOICE_ITEMS | | INVOICE_ITEMS_PROD_IDX | AP_IDX | VALII |
| AP | RANDOMIZED_PARTED | | RANDOMIZED_PARTED_PK | ADO_COLD_IDX | |
| VALID | 10,365,139 | 10,365,139 | 6,722,752 | 2 | 28,733 |
| AP | RANDOMIZED_SORTED | | RANDOMIZED_SORTED_PK | ADO_COLD_IDX | |
| VALID | 10,194,228 | 10,194,228 | 76,739 | 2 | 22,556 |
| AP | RANDOMIZED_UNSORTED | | RANDOMIZED_UNSORTED_PK | ADO_COLD_IDX | |
| VALID | 10,395,465 | 10,395,465 | 10,379,934 | 2 | 23,004 |
| AP | VENDORS | | VENDORS_PK_IDX | AP_IDX | VALII |

Here's a breakout of what five of the most crucial statistics reveal:

- **Depth:** An index's *depth* refers to the number of root versus node versus leaf pages it comprises. A value of zero (0) means there is only a root page, while any number greater than zero indicates how many levels have been constructed to accommodate all the values.
- **Leaf blocks:** Since this count reflects the total number of leaf blocks that make up the index, the optimizer uses this number to determine if a full table scan is cheaper than a full index scan.
- **Distinct keys:** This column reflects the number of unique values stored in the index; for an index that's enforcing either a PRIMARY KEY or UNIQUE constraint, the number of distinct keys will equal the number of rows in the table.
- **Average leaf blocks per key:** The average number of leaf blocks per distinct key value indicates the average number of leaf blocks in which each distinct value in the index appears. For an index that supports either a PRIMARY KEY or UNIQUE constraint, this value will be equal to one (1); for a nonunique index, a larger value could be there in case the number of entries per distinct key value is more than what can fit into one leaf block.
- **Average data blocks per key:** Likewise, this column records the average number of *data blocks* per distinct value in the index; it gives an approximate measure of how many rows can be retrieved from the corresponding table for one key value.
- **Clustering factor:** This is probably the most crucial index statistic to understand

because the optimizer uses it to determine how efficient the index is for retrieving all the necessary ROWIDs from a B-tree index in the fewest number of index blocks.

The Impact of a Low Clustering Factor

When data has been loaded into a table and the rows in the table are not sorted by the indexed column, then the index on the column will have a low clustering factor. For example, consider the table AP.RANDOMIZED\_UNSORTED, which has been loaded using data from table AP.RANDOMIZED\_SORTED, as shown in [Listing 18.10](#). Note that the ORDER BY clause in the statement that loads data into AP.RANDOMIZED\_SORTED forces the data to be sorted in descending order based on the values for the KEY\_DESC column, whereas the index has been created on the KEY\_ID column.

Listing 18.10 Example of Low Index Clustering Factor

[Click here to view code image](#)

```
DROP TABLE ap.randomized_unsorted PURGE;
CREATE TABLE ap.randomized_unsorted (
    key_id      NUMBER(8)
    ,key_date   DATE
    ,key_desc   VARCHAR2(32)
    ,key_sts    NUMBER(2) NOT NULL
)
TABLESPACE ado_cold_data
NOLOGGING
PCTFREE 0;

TRUNCATE TABLE ap.randomized_unsorted;

INSERT /*+ APPEND NOLOGGING PARALLEL(4) */ INTO
ap.randomized_unsorted
SELECT /*+ PARALLEL(4) */ * FROM ap.randomized_sorted
ORDER BY key_desc DESC;

COMMIT;

ALTER TABLE ap.randomized_unsorted
DROP CONSTRAINT randomized_unsorted_pk;

DROP INDEX ap.randomized_unsorted_pk;

ALTER TABLE ap.randomized_unsorted
ADD CONSTRAINT randomized_unsorted_pk
PRIMARY KEY (key_id)
USING INDEX (
```

```

CREATE UNIQUE INDEX ap.randomized_unsorted_pk
    ON ap.randomized_unsorted(key_id)
    TABLESPACE ado_cold_idx
    NOLOGGING
    PARALLEL 4
);
ALTER INDEX ap.randomized_unsorted_pk LOGGING;

```

When index AP.RANDOMIZED\_UNSORTED\_PK\_IDX is created on column KEY\_ID, the index's clustering factor is dramatically different than that of AP.RANDOMIZED\_SORTED\_PK\_IDX, as the query and output in [Listing 18.9](#) shows. As a result, whenever a query filters data using a range scan against KEY\_ID for this table, it will use dramatically larger amounts of physical I/O because the data values are not tightly clustered within a few index pages.

Note

An index's clustering factor is only meaningful to the query optimizer for B-tree indexes; it is meaningless for bitmap indexes because of their structure and how they are used.

Operational Considerations for Indexes

Sometimes, an unexpectedly poor performance from an index might have nothing to do with the index at all. Its ill effect on a decision support system (DSS) query's execution speed or an OLTP application workload's execution rate per transaction could occur because the Oracle DBA employed an inappropriate index type; he might have ignored best practices, or he could simply be encountering a performance "headwind" because of the application's inability to scale up as expected.

Impact of Choosing the Wrong Index Type

As we've already discussed, when implementing a new index, it's important to remember which index type is the most appropriate:

- Generally speaking, remember that bitmap indexes are most effective when they are used to index an extremely large number of rows for a column that has relatively few distinct values that very rarely change.
- Conversely, B-tree indexes will generally outperform a bitmap index when the column being indexed has an extremely large number of frequently updated distinct values, and especially when the domain of unique values is constantly expanding over time.

Impact of Forcing the Use of an Index via a Hint

Unfortunately, forcing an index via a hint happens more frequently than it should. For example, suppose an application's user complains about the performance of a query. An

overzealous developer or DBA reviews the query's execution plan to determine the cause of the perceived poor performance. When she discovers that the query is not using the supposedly "appropriate" index, she might choose to force the use of the appropriate index by modifying the application code to incorporate a +INDEX hint.

That strategy may solve the query's performance issue (for now), but in due course of time when that selected index ceases to be the most appropriate choice—say, after the table has grown to such a size that a table scan with several degrees of parallelism is actually a more efficient access path to data—the optimizer will have no choice but to continue to use the now less-than-desirable index.

Impact of Outmoded Initialization Parameter Settings

It was not uncommon in earlier releases of Oracle to override the default values for the two initialization parameters discussed in this section in an attempt to influence the cost-based optimizer to favor indexes over table scans. We've often seen that these settings continue to be propagated forward into future releases of the Oracle database long after they might have outlived their usefulness.

The OPTIMIZER\_INDEX\_CACHING initialization parameter controls the costing of an index probe in conjunction with a nested loop or an INLIST iterator. The range of values 0 to 100 for this parameter indicates percentage of index blocks available in the buffer cache, which modifies the optimizer's estimate of the cost of an index probe for nested loops and INLIST iterators. A value of 100 infers that 100 percent of the index blocks are likely to be found in the buffer cache, and the optimizer adjusts the cost of an index probe or nested loop accordingly.

For good reason, the default value of this parameter is zero (0), or in other words, index blocks are no more likely than table blocks to be found in the database buffer cache, which results in the default optimizer behavior. Unless you know precisely how often this is likely to occur, Oracle suggests using caution when setting this parameter to any value other than zero.

The OPTIMIZER\_INDEX\_COST\_ADJ initialization parameter tweaks optimizer behavior to be either less or more inclined to choose an index versus a table scan to retrieve data. The minimum value of one (1) is the lowest setting, indicating the optimizer should consider indexes about 100 times more costly than table scans; conversely, the maximum setting of 10000 forces the optimizer to consider that index scans are 100 times less costly than table scans.

The default for this parameter is 100, at which the optimizer evaluates index access paths at the regular cost. Any other value makes the optimizer evaluate the access path at that percentage of the regular cost. Again, unless you know your application workload intimately and can accurately predict the implications of this setting after extensive testing, Oracle suggests leaving this parameter at its default value.

Impact of Deleting (Almost) All Index Entries in a Block

If an application permits deletion of rows from its tables, after a mass deletion has

occurred, it is possible that many index pages may have only a few entries that correspond to active, non-deleted rows. Unfortunately, Oracle cannot reclaim space in an index page until all entries have been deleted; as a result, it is possible that many more index pages are being scanned than necessary to retrieve index row pieces during indexed lookups. It's therefore recommended in this state to either rebuild an index or, if the database release is at least Oracle 10g Release 2, to compact the index segment with the `ALTER INDEX <index name> COMPACT;` command.

Impacts of Index Overuse

Finally, when it comes to tuning application performance, a prudent Oracle DBA should use indexes like a gourmet chef uses saffron or some other rare and costly spice: with great care, and not in every dish! Put simply, not every poorly performing query will suddenly start performing faster just because a new index has been added, and *every* index adds the specter of additional physical I/O whenever DML affecting the indexed column is performed on the underlying table.

Hiding Unselective Indexes

If your current Oracle Database release is at least 11g Release 1, then you are fortunate to have a powerful tool to battle unnecessary indexes on your tool belt: the ability to hide an index from the Oracle Optimizer for consideration as a possible access path during statement parsing.

An existing index can be made invisible by simply altering its state to `INVISIBLE`, and a new index can even be created in an initially `INVISIBLE` state, as shown in [Listing 18.11](#). Even though the optimizer will not be allowed to use this index as an access path, all DML operations against this index will continue to be processed.

Listing 18.11 Making an Existing Index Invisible to the Optimizer

[Click here to view code image](#)

```
ALTER INDEX sh.zm_customers_marital_bix INVISIBLE;

DROP INDEX sh.zm_customers_marital_bix;
CREATE BITMAP INDEX sh.zm_customers_marital_bix
    ON sh.zm_customers (cust_marital_status)
    TABLESPACE example
    INVISIBLE;
```

Note that even if the index is specified directly in a `+INDEX` hint, the optimizer will ignore it as long as it has been marked `INVISIBLE` ... unless, that is, the `OPTIMIZER_USE_INVISIBLE_INDEX` initialization parameter is set to `TRUE` at the session level, as [Listing 18.12](#) shows.

Listing 18.12 Making an Invisible Index Temporarily Visible to the Optimizer

[Click here to view code image](#)

```
SELECT /*+ INDEX(sh.zm_customers_marital_bix) */  
    cust_state_province, COUNT(*)  
   FROM sh.zm_customers  
 WHERE cust_marital_status = 'widow'  
 GROUP BY cust_state_province  
 ORDER BY cust_state_province;
```

Plan hash value: 2056234527

| Id Operation | Name | Rows | Bytes | Cost |
|---------------------------------------|------|-------|-------|------|
| (%CPU) | Time | | | |
| 0 SELECT STATEMENT | | 145 | 2465 | |
| 424 (1) 00:00:01 | | | | |
| 1 SORT GROUP BY | | 145 | 2465 | |
| 424 (1) 00:00:01 | | | | |
| * 2 TABLE ACCESS FULL ZM_CUSTOMERS | 3461 | 58837 | | |
| 423 (1) 00:00:01 | | | | |

Predicate Information (identified by operation id):

2 - filter("CUST\_MARITAL\_STATUS"='widow')

```
ALTER SESSION SET optimizer_use_invisible_index = TRUE;
```

```
SELECT /* +INDEX(sh.zm_customers_marital_idx) */  
    cust_city, COUNT(*)  
   FROM sh.zm_customers  
 WHERE marital_status = 'S';
```

Plan hash value: 2120557835

| Id Operation | | | | | |
|---|------|-------|------|--------|----------|
| Name | Rows | Bytes | Cost | (%CPU) | Time |
| 0 SELECT STATEMENT | | | | | 58 |
| 986 19 (6) 00:00:01 | | | | | |
| 1 SORT GROUP BY | | | | | 58 986 |
| 19 (6) 00:00:01 | | | | | |
| 2 TABLE ACCESS BY INDEX ROWID BATCHED | | | | | |

```

ZM_CUSTOMERS          |    75 | 1275 |    18   (0) | 00:00:01 |
| 3 | BITMAP CONVERSION TO
ROWIDS               |
|* 4 | BITMAP INDEX SINGLE VALUE
ZM_CUSTOMERS_MARITAL_BIX |      |      |      |      |
-----
-----
Predicate Information (identified by operation id):
-----
4 - access("CUST_MARITAL_STATUS">'widow')

```

If the Oracle DBA determines that the index should indeed remain visible, it can simply be rendered visible to the optimizer again by issuing the `ALTER INDEX <index name> VISIBLE;` command.

Index Performance Issues in RAC Databases

RAC databases present their own unique challenges in regard to index performance, as discussed in the following sections.

Impact of Nonselective Indexes

Indexes that are *nonselective*—that is, that don't dramatically improve query performance—can have an especially deleterious effect on DML in RAC because they may be subject to *interinstance contention*. For example, consider a table with five indexes: one primary key index and four other indexes that were created in anticipation of particular query workloads. If those four additional indexes provide very little benefit for speeding query execution times, they may actually be hindering the RAC database's ability to quickly update these nonselective indexes. Even worse, if these indexes need to be accessed by queries while OLTP activity is occurring, it is likely that this concurrent query and OLTP activity will cause severe cache transfer contention between RAC database instances because both a *current* image and at least one *CR* image must be maintained for each of the index pages.

Impact of Monotonically Increasing Indexes

Indexes that grow *monotonically* because of *ever-increasing values* (for example, `OE.ORDERS.ORDER_ID`) often become hot spots in a RAC application workload. Consider a typical primary key strategy that simply increments the next key value by +1 each time a new entry is requested. In this situation, the corresponding primary key index will tend to cluster all of the like values in a very few index blocks; this clustering will result in frequent leaf block splits, the index will end up with an index tree structure that is relatively low in depth, and the index's root block will tend to be accessed much more frequently than other leaf pages as compared to the case when the index was not monotonic. In a RAC database, this strategy is likely to become a serious performance bottleneck for OLTP applications, especially if *multiple* instances are hosting the OLTP application: the instances will essentially play a constant game of tug-of-war as they

exchange holder status of the current primary index block.

One approach to alleviating this issue is to implement *global index hash partitioning*, a new feature in Oracle 10g. Other approaches include using a *natural key* based on existing database column values (that is, instead of a “dumb-numbering” scheme), or even using a reverse-key index as described previously. (Remember, reverse-key indexes cannot be used for a range scan.)

Oracle Sequences and Index Contention

If sequences are used to obtain a table’s next value for its primary key column, it’s important to verify that the sequence’s CACHE value is set much higher than the default of 20—high enough to ensure that two or more instances will never access the same index page at the same time—and also to specify the NOORDER directive. (Recall that the NOORDER clause means that if Session 1 requests a NEXTVAL from a sequence before Session 2 does, it doesn’t matter that Session 2’s value is higher than Session 1’s.)

This means that each instance in the cluster will get a much higher range of sequence values to choose from, eliminating index page contention as a factor; however, this approach might result in large gaps between sequenced values because each instance has reserved a widely disjoint set of values.

OLTP and Read-Mostly Application Workload Collisions

Finally, sometimes problems related to index difficulties are simply the symptom of a set of application operations colliding unexpectedly at the same time. Consider this situation: an OLTP application has recently added new rows to an indexed table on one RAC node, but the changes haven’t been committed yet. There is also a read-mostly workload (for example, batch reporting) executing on another RAC node that is actively attempting to read from the same table and indexes. Because the changes haven’t been committed on the first node, the second node has no choice but to request consistent read (CR) images be generated from undo data on the first node for *both* the tables and the indexes. If we now imagine that *both* nodes are servicing *both* the OLTP and the batch reporting workload, it’s easy to understand how quickly the RAC database’s interconnect will tend to spend most of its time processing requests for CR images.

If the Oracle DBA cannot get the application developers to commit changes more frequently, then it’s very likely that the CR images of the data and index blocks will take up most of the RAC database’s interconnect bandwidth. Likewise, if the indexes are significantly unselective, all of this extra work to maintain CR images for the index blocks will turn out to be essentially counterproductive.

Summary

An index may not be the panacea to cure a query’s performance ills, so don’t be afraid to strenuously question the need for a new index if the intent is merely to speed a query’s execution. Remember that an index may actually contribute significantly to the execution

time of DML against the index's base table or partition, so adding indexes, especially to often-updated data segments, may actually cause more trouble than they are worth.

Invest time to choose the right index type. A bitmap index may be excellent for a column with a small domain of indexed values—until a single brand-new value needs to be added to that column's domain, forcing a complete rebuild of all bitmap index segments to incorporate that value.

Remember that the *B* in B-tree stands for “balanced”—and that means that except for the most extreme circumstances, there is little need to rebuild a B-tree index because it remains balanced in almost all situations.

If deletions are performed against a table with large numbers of indexed columns, remember that space allocated to an index block cannot be reclaimed until all values within that block are marked as deleted, and it's possible that poor application performance may result because many more “almost empty” index blocks may have to be accessed.

The onset of engineered systems such as Exadata and the innovation of In-Memory Column Store (IMCS) in Oracle Database 12.1.0.2 often completely negate the need for indexes for analytical queries. So when in doubt of an index's benefit, simply consider making that index invisible—and *telling no one*. If there are no significant complaints about application performance with an invisible index, you have probably found an opportunity to reduce the demands of keeping that index updated by removing it entirely.

If you're fortunate enough to have made the transition to Oracle Database 12c Release 1 for your data warehouse, consider leveraging the new PARTIAL index feature to limit the creation of local and/or global index partitions against the table partitions that can least benefit from them while retaining index partitions for the table partitions that could most benefit from speedy data access.

Real Application Cluster (RAC) databases are often persnickety about indexes. Remember that applications with too many unselective indexes, improperly sequenced monotonically increasing indexes, or colliding OLTP versus read-mostly workloads can often trace their unexpectedly poor performance or lack of scalability directly to indexes.

19. Using SQLT to Boost Query Performance

Oracle DBAs can be divided into two broad groups: those of us who are lucky enough to have installed, configured, and leveraged the SQLT utility to help improve SQL query performance and those of us who have never even heard of the SQLT utility. Carlos Sierra created SQLT while working for Oracle many years ago, well before many of today's Oracle database performance analysis tools that are built into more current versions of the database kernel—SQL Tuning Advisor, SQL Access Advisor, SQL Performance Analyzer, and SQL Monitor—even existed.

Because it focuses on the best practices for tuning SQL statements effectively and with minimum effort, SQLT is also a great introduction for a “newbie” DBA to learn how to isolate, diagnose, and tune just about any SQL statement to improve database application performance. Best of all, SQLT captures and displays an abundance of information regarding the SQL statement identified by its `sql_id` because it generates a detailed report containing information about the related optimizer statistics, initialization parameters, object metadata, execution plans, and much more.

This chapter focuses on several topics, including installing SQLT, using the `XTRACT` and `EXECUTE` methods, and leveraging other SQLT methods. Finally, it provides a real-world example for using SQLT to improve query performance.

We only briefly review the necessary steps to install the SQLT utility; further, this chapter does not attempt to explain the details of how SQLT actually works, as that is beyond the scope of this chapter. For a deeper look into how to leverage the full capacity of SQLT for performance tuning, be sure to review the several excellent My Oracle Support (MOS) documents listed at the end of this chapter.

Installing SQLT

Installing SQLT is extremely simple and involves a handful of steps:

1. Download the `sqlt.zip` file based on the instructions in MOS Note 215187.1, “All about the SQLT diagnostic tool.”
2. Navigate to the selected installation directory and run the `sqcreate.sql` script as follows:

[Click here to view code image](#)

```
$> cd sqlt/install  
$> sqlplus / as sysdba  
SQL> START sqcreate.sql  
  
zip warning: name not matched: *_sq*.log  
  
zip error: Nothing to do! (SQLT_installation_logs_archive.zip)  
zip warning: name not matched: *_ta*.log
```

```
zip error: Nothing to do! (SQLT_installation_logs_archive.zip)

PL/SQL procedure successfully completed.

. . .
(many lines of output suppressed for sake of brevity)
. . .

SQUTLTEST completed.
adding: 150523163243_10_squtltest.log (deflated 59%)
```

SQLT users must be granted `SQLT_USER_ROLE` before using this tool.

SQCREATE completed. Installation completed successfully.

3. Provide the necessary information about the appropriate connect identifier, `SQLTXPLAIN` password, default tablespace, and temporary tablespace during the execution of `sqcreate.sql`.

For example, if SQLT is being installed within an Oracle E-Business Suite (EBS) environment, it will be necessary to provide the APPS username. This is optional, and it is the schema owner of the statement's `SQL_ID` that will be analyzed.

4. If queries are going to be executed by more than one database user, grant the `SQLT_USER_ROLE` to those users to permit those accounts to leverage SQLT.
5. When the installation script prompts, indicate whether the database is currently licensed for either the Oracle SQL (T)uning Pack, (D)agnostic Pack, or (N)either Diagnostic nor Tuning packs.

SQLT can also be installed *silently* by modifying the contents of `sqdefparams.sql` and then running script `sqcsilent.sql` as follows:

```
$> cd sqlt/install
$> sqlplus / as sysdba
SQL> START sqdefparams.sql
SQL> START sqcsilent.sql
```

Note that it's also possible to supply the desired parameters within a single command string, as follows:

[Click here to view code image](#)

```
$> cd sqlt/install
$> sqlplus / as sysdba
SQL> START sqcsilent2.sql '' sqlxplain USERS TEMP '' T
```

Finally, when an Oracle DBA is prohibited from installing anything additional on an existing database because of specific regulatory restrictions or tightened security arrangements, consider using the `sqlhc.sql` utility, which invokes the Oracle database *Health Check* script that can also collect information about a specific `sql_id`, albeit

with a limited diagnostic range.

Using the XTRACT Method

SQLT has numerous methods that can be leveraged, but this chapter focuses on just two key methods: XTRACT and XECUTE. The main difference between the two methods is that, as the name suggests, the XECUTE method actually *executes* the statement identified by `sql_id`, and the XTRACT method *does not execute* the statement. As we will see in the next section, the XECUTE script generates much more information about the statement.

The point here is to show how simple it is to create a report that contains all the information that we need to tune a specific query. The report created as output of these methods of SQLT contains easy-to-use, crucial information and also contains some suggestions about what to do in some specific cases to improve the statement's performance—for example, regathering stale statistics, defragmenting fragmented tables or indexes, and dealing with known bugs.

Here is an example of how with one simple command we can generate a complete SQLT report using the XTRACT method:

1. Make sure the database's `STATISTIC_LEVEL` initialization parameter has been set to `ALL`. This setting ensures that SQLT will collect even deeper information about the statement.
2. Connect to a SQL\*Plus session using an appropriate application database user. For example, in EBS, this would be the APPS user.
3. Run the SQLT script using either the `sql_id` or the SQL hash value of the statement that is experiencing performance problems. These identifiers are easily obtained—say, for example, either by an AWR report or an ASH report:

[Click here to view code image](#)

```
# cd sqlt
# sqlplus apps
SQL> START [path]sqltxecute.sql [path]scriptname
[sqlxplain_password]
SQL> START run/sqltxecute.sql input/sample/script1.sql
sqlxplain_password
```

At the end, a .ZIP file will be created containing all the information available about the statement corresponding to the supplied `sql_id` or SQL hash value. Opening the `sqlt_sXXX_main.html` file will display something like what is shown in [Figure 19.1](#).

215187.1 SQLT XTRACT 12.1.01 Report: sqlt\_s70747\_main.html

This report may include some content provided by the Oracle Diagnostic writer for Oracle Tuning Plans (as per issue #215187.1SQL Tuning Advisor 12.1.01). SQL Tuning Advisor 12.1.01 SQL Monitoring and/or Automatic Workload Repository (AWR). An error that using the numbered functionality requires a license for the corresponding pack. If you need to disable SQLT issues its own set of WMRN packages please execute one of the following command: SQL> EXEC DBMS\_MONITOR.DISABLE\_CURSOR\_MONITOR; or SQL> EXEC DBMS\_MONITOR.DISABLE\_AWR\_CURSOR\_MONITOR; or SQL> EXEC DBMS\_MONITOR.DISABLE\_WLR\_CURSOR\_MONITOR; or SQL> EXEC DBMS\_MONITOR.DISABLE\_WLR\_AWR\_CURSOR\_MONITOR;.

sql\_start 2014-01-30 15:27:48

Figure 19.1 A representation of output from XTRACT script execution

The resulting output contains an almost overwhelming amount of information divided into specific areas like *Global*, *Plans*, *Tables*, *Plan Control*, *Cursors*, *SQL Tuning Advisors*, *Objects*, and *SQL Execution*. Navigating between report sections is as easy as clicking on the desired HTML link in the report. Without this script, it could take significant time to gather this level of detailed information about a single statement, but the XTRACT method dramatically reduces that effort.

Using the XECUTE Method

As mentioned previously, the XECUTE method generates a much more detailed report because it actually executes the SQL statement that corresponds to the specified `sql_id`. Therefore, if the SQL statement takes more than one hour to complete during its execution, be aware that this may have a deleterious effect on the corresponding database environment, especially if the statement is known to consume large amounts of CPU or I/O bandwidth.

Executing the XECUTE method is almost as simple as executing the XTRACT method but requires slightly different steps as shown:

1. Prepare a separate file containing the SQL statement itself.
2. If the statement being analyzed uses bind variables, declare them in the same file as the SQL statement. The following example demonstrates how to accomplish this:

[Click here to view code image](#)

```
-- execute sqlt xecute as sh passing script name
-- cd sqlt
-- #sqlplus sh
-- SQL> start run/sqltxecute.sql input/sample/script1.sql
```

```
REM Optional ALTER SESSION commands
REM ~~~~~~
```

```

--ALTER SESSION SET statistics_level = ALL;

REM Optional Binds
REM ~~~~~

VAR b1 NUMBER;
EXEC :b1 := 10;

REM SQL statement to be executed
REM ~~~~~

SELECT /*+ GATHER_PLAN_STATISTICS MONITOR BIND_AWARE */
       /* ^unique_id */
       s1.channel_id,
       SUM(p.prod_list_price) price
  FROM products p,
       sales s1,
       sales s2
 WHERE s1.cust_id = :b1
   AND s1.prod_id = p.prod_id
   AND s1.time_id = s2.time_id
 GROUP BY
       s1.channel_id;
/
/

```

REM Notes:

REM 1. SQL must contain token: /\* ^unique\_id \*/

REM 2. Do not replace ^unique\_id with your own tag.

REM 3. SQL may contain CBO Hints, like:

REM /\*+ gather\_plan\_statistics monitor bind\_aware \*/

Note

The **sqlt.zip** file contains several examples of how to accomplish bind variable declarations.

For SQL statements that contain data manipulation language (DML) commands, the XECUTE method will create a SAVEPOINT prior to the statement's execution; at the end of the transaction, XECUTE issues a rollback to return the database's state prior to the SAVEPOINT.

3. Run the XECUTE script, passing the text file created that contains the SQL statement and its bind variables declared in it:

[Click here to view code image](#)

```

# cd sqlt
# sqlplus apps
SQL> START [path]sqltxecute.sql [path]scriptname

```

```
[sqltxplain_password]
16:43:56 APPS@BWEBSPR5 > START run/sqltxexecute.sql
input/input_pp.sql f2cb2w08
```

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.01

Parameter 1:

SCRIPT name which contains SQL and its binds (required)

. . .

When using the XECUTE method, you will be prompted for the SQLXPLAIN password that was provided during the installation of SQLT. Also note that if the XECUTE method is being invoked by an application user, that user account must have been GRANTED the SQLT\_USER\_ROLE.

4. Monitor the progress of the XECUTE method by querying the contents of view SQLTXADMIN.SQLT\$\$\_LOG\_V, as the following example demonstrates:

[Click here to view code image](#)

```
SQL> SELECT * FROM SQLTXADMIN.sqlt$$_log_v;

TIME      LINE
-----
-----
16:49:11 sqlt$a: tool version: 12.1.11
16:49:11 sqlt$a: script version: 12.1.11
16:49:11 sqlt$a: -> common_initialization
16:49:11 sqlt$a: ALTER SESSION SET NLS_NUMERIC_CHARACTERS = ".,"
16:49:11 sqlt$a: ALTER SESSION SET NLS_SORT = BINARY
. . .
16:49:37 sqlt$d: task name = "sqlt_s93766_mem"
16:49:37 sqlt$d: <- create_tuning_task_text
16:49:37 sqlt$d: -> collect_tuning_sets_mem
16:49:37 sqlt$d: sqlt$$_gv$sql_plan: 1518346075

181 rows selected.
```

SQL>

Once XECUTE has completed, it will produce a complete and detailed report, as shown in [Figure 19.2](#).

215187.1 SQLT XECUTE 12.1.11 Report: sqlt\_s93766\_main.html

| Global | Plans | Tables |
|---|--|--|
| <ul style="list-style-type: none"> • Observations • SQL Text • SQL Identification • Environment • CBO Environment • Fix Control • CBO System Statistics • DBMS_STATS Setup • Initialization Parameters • NLS Parameters • I/O Calibration • Tool Configuration Parameters | <ul style="list-style-type: none"> • Summary • Performance Statistics • Performance History (delta) • Performance History (total) • Execution Plans | <ul style="list-style-type: none"> • Tables • Statistics • Statistics Extensions • Statistics Versions • Modifications • Properties • Physical Properties • Constraints • Columns • Indexed Columns • Histograms • EBS Histograms • Partitions • Indexes |
| Cursor Sharing and Binds | Plan Control | Objects |
| <ul style="list-style-type: none"> • Cursor Sharing • Adaptive Cursor Sharing • Peeked Binds • Captured Binds | <ul style="list-style-type: none"> • Stored Outlines • SQL Patches • SQL Profiles • SQL Plan Baselines • SQL Plan Directives | <ul style="list-style-type: none"> • Objects • Dependencies • Fixed Objects • Fixed Object Columns • Nested Tables • Policies • Audit Policies • Tablespaces • Metadata |
| SQL Tuning Advisor | SQL Execution | |
| <ul style="list-style-type: none"> • STA Report • STA Script | <ul style="list-style-type: none"> • Active Session History • AWR Active Session History • SQL Statistics • SQL Detail ACTIVE Report • Monitor Statistics • Monitor ACTIVE Report • Monitor HTML Report • Monitor TEXT Report • Segment Statistics • Session Statistics • Session Events • Parallel Processing | |

Figure 19.2 Output from XECUTE script execution

Just as with the XTRACT method, the resulting XECUTE report output contains an almost overwhelming amount of information, including significant recommendations for improving the performance of the targeted SQL statement.

Leveraging Other SQL Methods

SQL offers several other methods that could be more valuable than XTRACT and XECUTE depending on the specific use case. A brief explanation for each of these methods follows:

- **XTRXEC method:** If a statement has bind variables and an Oracle DBA wants to identify the best variables to analyze in terms of which one(s) influenced the worst performance for the statement, she would use the XTRXEC method. This method will first run XTRACT to determine the best bind variables to use in the report; afterwards, the DBA can use the results from XTRXEC to analyze the statement with the XECUTE method to get the complete picture about the statement's execution with those bind variables.
- **XTRSBY method:** This SQL method makes it possible to analyze the statement by executing it on a physical standby database, thus limiting the impact on the primary production database's performance. Using XTRSBY requires the creation of a database link on the primary database so that SQL can write information into SQL tables on that database over the link.
- **XPLAIN method:** This method is essentially the same as executing an EXPLAIN

PLAN FOR command for the statement; however, be aware that it cannot view the statement's bind variables.

- **XPREXC method:** This method is a faster way to run the XECUTE method because it disables some of XECUTE's features; it's therefore faster and will save at least some resource utilization.
- **COMPARE method:** This method is useful if a SQL statement is executing in two different database environments, but it is running much faster on one database than on the other. COMPARE does require the importation of the SQLT repository from one of the database environments to the other environment so that the comparison can be executed; it's also possible to export and then import the SQLT repositories from two different database environments to a third environment just for the purposes of running the comparison.
- **TRCANLZR method:** This method essentially offers the same functionality as the Oracle Trace Analyzer utility; however, this method offers the capability to analyze several traces concurrently when the traces resulted from a parallel execution.
- **TRCAXTR method:** The TRCAXTR method essentially works the same as TRCANLZR, except that it will call the XTRACT method for the worst-performing SQL statement found in the Trace Analyzer output.
- **TRCASPLIT method:** If a trace is created using EVENT 10046, the TRCASPLIT method can be used to analyze the output. This method creates two different files: one with the actual lines generated by EVENT 10046 and one with the lines not generated by setting the event. This makes it much simpler to identify just the key output from that SQL trace method.
- **XTRSET method:** This method is excellent for locating the worst-executing SQL statement in an AWR report and then executing XTRACT for that particular statement.

A Real-World Example

Since the goal of this book is to help Oracle DBAs find and fix almost any kind of problem in their database environments, let's turn our attention to how we can leverage the main methods of SQLT in a real-world environment. The following example demonstrates how simple it is to quickly tune a poorly performing query in a database in just a few minutes without requiring the intervention of an Oracle performance DBA:

1. If the offending query was submitted by an application user, find its `sql_id` using the `GV$SQLTEXT` or `DBA_HIST_SQLTEXT` view:

[Click here to view code image](#)

```
select sql_id
  from DBA_HIST_SQLTEXT
 where sql_text like '%where DATE_VALUE > sysdate -180% ';
```

Alternatively, to retrieve the `sql_id` for a poorly performing query from an Automatic Workload Repository (AWR) snapshot period, run the AWR report first to isolate the query (or use the XTRSET method explained earlier).

- After obtaining the `sql_id`, run the XECUTE method for that `sql_id`, as follows:

[Click here to view code image](#)

```
$> START run/sqltxecute.sql input/input_pp.sql f2cb2w08
PL/SQL procedure successfully completed.

Elapsed: 00:00:00.01
```

- Once the XECUTE method completes, review the resulting output's *Plan Hash Values* report to find which possible execution plan hash value(s) may relate to the reported performance issues. As the report sample in [Figure 19.3](#) shows, there is one execution plan that offers superior performance.

| Execution Plans | | | | | | | | | | | | | | | |
|-----------------|-----------------|---------|-------|-------------------|-------------------|--------------|-------------|----------------|-----------------------|----------------|---------------------|--------------|---------|---------|-----------|
| # | Plan Hash Value | Value#? | Obj | Source | Plan Info | Plan Details | In Baseline | Optimizer Cost | Estimated Cardinality | Rows Processed | Plan Timestamp | Child Plan#? | Plan ID | Task ID | Attribute |
| 1 | 14238810 | 33786 | AWR | DBA_HIST_SQL_PLAN | | | | 21576 | 833 | 3156 | 2013-06-01T17:19:32 | | | | |
| 2 | 18485330 | 61679 | AWR | DBA_HIST_SQL_PLAN | | | | 25460 | 2612 | 377 | 2013-06-01T17:17:26 | | | | |
| 3 | 877352879 | 62811 | AWR | DBA_HIST_SQL_PLAN | | | | 20145 | 2012 | 12546 | 2013-05-31T13:11:06 | | | | |
| 4 | 162734829 | 24162 | 21102 | AWR | DBA_HIST_SQL_PLAN | | | 20042 | 2012 | 3647 | 2013-05-31T17:17:06 | | | | |
| 5 | 234957315 | 94154 | 21137 | MEM | DBMS_SQL_PLAN | | H | 30889 | 2012 | 922 | 2013-06-01T00:26:23 | 1 | | | |
| 6 | 334957315 | 94154 | 94154 | AWR | DBA_HIST_SQL_PLAN | | | 20886 | 2012 | 922 | 2013-06-01T00:26:23 | | | | |
| 7 | 324957315 | 94154 | 26439 | XPL | FLAN_TABLE | | | 20985 | 2012 | 922 | 2013-05-10T20:56:56 | 32 | | | |

Figure 19.3 Execution plan

- To repair this query's poor performance, create a SQL Profile to give the query optimizer the ability to choose the better plan (in Oracle Database 11g, recall that a SQL plan baseline could also be created):

[Click here to view code image](#)

```
-- This creates the script below that must be run to create the profile:
START coe_xfr_sql_profile.sql gqmv2hr133bjv 163726467

--This script finally creates the profile:
@coe_xfr_sql_profile_gqmv2hr133bjv_4271579545.sql
```

My Oracle Support Reference Documents

The following My Oracle Support (MOS) documents offer excellent reference information on how to best leverage the SQLT utility and its related feature set:

- **215187.1**: “All about the SQLT diagnostic tool”
- **1454160.1**: “FAQ: SQLT (SQLTXPLAIN) frequently asked questions”
- **1465741.1**: “How to use SQLT (SQLTXPLAIN) to create a testcase containing application data”
- **1470811.1**: “How to use SQLT (SQLTXPLAIN) to create a testcase without row data”

- **1614107.1**: “SQLT usage instructions”
 - **1614201.1**: “SQLT changes”
 - **1670677.1**: “FAQ: Common SQLT (SQLTXPLAIN) runtime/installation errors”
 - **1683772.1**: “How to collect standard diagnostic information using SQLT for SQL issues”
 - **1922234.1**: “SQLT main report: Usage suggestions”
 - **1951018.1**: “SQLT utility and partitioned tables”
-

Summary

This chapter showed how easy is to tune query performance using SQLT. The methods that SQLT employs were briefly explained, and a real-world test case demonstrated how to leverage this utility with just a few simple steps. Without SQLT, an Oracle DBA would most likely have to manually run several individual queries and pursue several erroneous false leads before isolating the actual root cause (or causes) of the statement’s poor performance.

20. Dealing with XA Distributed Transaction Issues

In today's complex Oracle database environments, it is not uncommon to encounter database application architectures that leverage Java components that connect to two or more different databases using connection pools and require processing a single transaction in a coordinated fashion. This multiconnection, multidatabase transaction processing method is typically known as a *distributed transaction*, but it is sometimes also termed a *global transaction* or *XA* (short for X/Open XA). The concept of distributed transactions is not particularly new—it was created back in 1991—but as Java has come to the fore as an application language in recent years, the use of distributed transactions has accelerated in many Oracle database environments.

This chapter therefore focuses on some well-known issues that can occasionally arise when an Oracle database is using distributed transactions. It demonstrates just how quickly an XA distributed transaction problem might arise in a complex database environment, how to identify and diagnose the problem, and of course how to quickly repair the issue to restore the corresponding database application to peak performance.

Note

Detailed information about the architecture, components, error handling, and optimization of distributed transactions can be found in the web-based Oracle Database documentation portal. The corresponding JDBC Developer's Guide for each Oracle database release is especially valuable for gaining a better understanding of the key concepts of distributed transactions.

Repairing Common Distributed Transaction Issues

Imagine an e-commerce database application environment comprising several different types of applications executing against numerous Oracle databases, including at least one whose data sources are leveraging distributed transaction processing. Perhaps a short-lived network interruption occurs; perhaps one of the databases that's participating in a distributed transaction environment crashes; or perhaps even the application software itself crashes.

In any of these situations, it is not unlikely that the corresponding distributed transaction may be left in an indeterminate state because the transaction may not be able to reconnect to the database with which it lost connectivity. In this situation, the transaction will become what is typically called a *lost distributed transaction*. Each distributed transaction generates a lock on a resource, and this resource is not released until the transaction coordinator receives either a COMMIT or ROLLBACK. Therefore, if *one* distributed transaction should hang, it is crucial to fix this pending transaction immediately; otherwise, it is possible that the *entire production environment* could hang.

as well.

Fortunately, solving this particular issue requires just a few simple steps:

1. Query the DBA\_2PC\_PENDING view to locate any pending distributed transactions:

[Click here to view code image](#)

```
SQL> SELECT COUNT(*) FROM dba_2pc_pending;
```

Of course, this SQL script can be encapsulated within a customized shell script, embedded within an application monitoring tool such as Oracle APP Manager, or even used to create a user-defined metric (UDM) in Oracle Enterprise Manager Grid Control 11g or Cloud Control 12c.

2. If a pending transaction is indeed detected, it must either be committed or rolled back. Use the following script to create a series of ready-to-run SQL commands that will roll back all pending transactions in your database returned by the prior query:

[Click here to view code image](#)

```
SQL> SET HEADING OFF
SQL> SELECT 'ROLLBACK FORCE '''||local_tran_id||''';' FROM
dba_2pc_pending;
```

3. In an extreme case, the distributed transaction may not disappear from the DBA\_2PC\_PENDING view even after committing or rolling back all pending transactions using the commands just generated. In this situation, purge the pending transaction using the PURGE\_LOST\_DB\_ENTRY procedure of package DBMS\_TRANSACTION:

[Click here to view code image](#)

```
SQL> SELECT 'EXECUTE
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('''||local_tran_id||''');COM
FROM dba_2pc_pending;
```

Querying the DBA\_2PC\_PENDING view will verify that no pending transactions exist anymore, thus proving that all in-doubt transactions have been repaired.

Repairing Ghost Distributed Transactions

In isolated, unexpected situations, a distributed transaction may simply be lost and become what is known as a *ghost transaction*. For example, a distributed transaction may appear to be pending in the DBA\_2PC\_\* views, but it is impossible to either commit or roll back the transaction, and it also cannot be purged. Another scenario may involve application users receiving an ORA-01591 error message as follows, but the specific transaction doesn't appear at all in any of the DBA\_2PC\_\* views:

[Click here to view code image](#)

```
ORA-01591: lock held by in-doubt distributed transaction 1.21.17
```

This could happen in cases where the databases involved in the distributed transaction cannot be accessed simultaneously. If the RECO background process responsible for recovering these transactions cannot handle this situation—perhaps because of a failure in network communication between the databases processing the transaction—it is possible that the transaction was unable to be added to the DBA\_2PC\_\* views.

Ghost transactions typically require some rather tricky commands to repair them. There are three special scenarios that require some relatively dangerous commands to repair them, and we discuss each of them in the following sections:

- There are entries in the DBA\_2PC\_\* views for a distributed transaction, but that transaction has *ceased to exist*.
- A distributed transaction is hanging, but there is no corresponding information for that transaction in the DBA\_2PC\_\* views.
- A COMMIT or ROLLBACK is performed against a distributed transaction that is in a PREPARED state, but the COMMIT or ROLLBACK command hangs.

Information Exists, but Transaction Missing

A transaction exists in the DBA\_2PC\_\* views, but that transaction no longer can be found in the corresponding database. In this situation—when the status of the transaction is COMMITTED, ROLLBACK FORCE, or COMMIT FORCED—the transaction(s) just need to be purged. However, if the transaction is in the PREPARED state and no entry is located in the transaction table for this transaction, then the following steps are required to solve this issue:

1. Identify the transaction in question:

[Click here to view code image](#)

```
SQL> SELECT local_tran_id, state FROM dba_2pc_pending;  
  
LOCAL_TRAN_ID          STATE  
-----  
1704.34.52393          prepared
```

2. The local\_tran\_id is <RBS#>.<SLOT#>.<WRAP#>, which means that the rollback segment number is **1704**. Locate the active transactions on this rollback segment using this query:

[Click here to view code image](#)

```
SQL> SELECT KTUXEUSN, KTUXESLT, KTUXESQN, /* Transaction ID */  
KTUXESTA Status,  
KTUXECFL Flags  
FROM x$ktuxe  
WHERE ktuxesta != 'INACTIVE'  
AND ktuxeusn = 1704; <== this is the rollback segment#
```

In this particular example, the query did not return any row, and so it is impossible to issue a COMMIT FORCE or ROLLBACK FORCE for this transaction:

[Click here to view code image](#)

```
SQL> rollback force '1704.34.52393';
ORA-02058: no prepared transaction found with ID 1704.34.52393
```

3. Unfortunately, the only way to clean up this transaction is to manually delete it from internal tables. Be sure to first issue the SET TRANSACTION command so that it uses the rollback segment of the SYSTEM tablespace:

[Click here to view code image](#)

```
SQL> SET TRANSACTION USE ROLLBACK SEGMENT SYSTEM;
```

4. Delete the transaction using the transaction ID (local\_tran\_id) directly. Since it was carefully deleted manually from the appropriate internal transaction tables, the transaction will cease to exist:

[Click here to view code image](#)

```
SQL> DELETE FROM sys.pending_trans$ WHERE local_tran_id =
  '1704.34.52393';
SQL> DELETE FROM sys.pending_sessions$ WHERE local_tran_id =
  '1704.34.52393';
SQL> DELETE FROM sys.pending_sub_sessions$ WHERE local_tran_id =
  '1704.34.52393';
SQL> COMMIT;
```

ORA-1591 Has No Corresponding Information

In this next scenario, a database application user session is receiving an ORA-1591 error message, but a search of the DBA\_2PC\_\* views reveals no such transaction exists.

An application user's attempt to change a row in a table receives the following error:

[Click here to view code image](#)

```
ORA-1591: lock held by in-doubt distributed transaction
'1704.34.52393'
```

Resolving this situation requires a slightly different approach:

1. Search the DBA\_2PC\* views for the local transaction ID. This search returns no rows, which indicates that the transaction seems to have disappeared:

[Click here to view code image](#)

```
SQL> SELECT *
      FROM dba_2pc_pending
     WHERE local_tran_id='1704.34.52393';
```

```
No rows returned
```

2. Use the following query to identify the transaction in its rollback segment; it is apparent that it is in the PREPARED state:

[Click here to view code image](#)

```
SQL> SELECT
    KTUXEUSN,
    KTUXESLT,
    KTUXESQN,
    KTUXESTA Status,
    KTUXECFL Flags
   FROM x$ktuxe
  WHERE ktuxesta != 'INACTIVE'
    AND ktixeusn= 1704;
```

| KTUXEUSN | KTUXESLT | KTUXESQN | STATUS | FLAGS |
|----------|----------|----------|----------|-----------------------|
| 1704 | 34 | 52393 | PREPARED | SCO COL REV DEA |

However, any attempt to commit or roll back this transaction returns an ORA-2058 error:

[Click here to view code image](#)

```
SQL> COMMIT FORCE '1704.34.52393';
ORA-02058: no prepared transaction found with ID 1704.34.52393
```

3. The ORA-02058 error occurs because there are no entries in the DBA\_2PC\_\* views with the specified local\_id. Insert a dummy row into the PENDING\_TRAN\$ and PENDING\_SESSION\$ tables to essentially re-establish the transaction with a status of PENDING, and then reattempt to commit this transaction:

[Click here to view code image](#)

```
-- Disable distributed recovery to prevent the RECO background
process
-- from handling this dummy row
SQL> Alter System Disable Distributed Recovery;
```

```
SQL> INSERT INTO pending_trans$ (
local_tran_id,
global_tran_fmt,
global_oracle_id,
state,
status,
session_vector,
reco_vector,
type#,
fail_time,
reco_time)
VALUES (
'1704.34.52393', /* ONLY THIS ROW SHOULD BE MODIFIED WITH YOUR
LOCAL TRAN ID*/
```

```

306206,
'XXXXXXXX.12345.1.2.3',
'prepared',
'P',
hextoraw( '00000001' ),
hextoraw( '00000000' ),
0,
sysdate,
sysdate ) ;

SQL> INSERT INTO pending_sessions$
VALUES (
'1704.34.52393',
1,
hextoraw('05004F003A1500000104'),
'C',
0,
30258592,
',
146
) ;

SQL> COMMIT;

SQL> COMMIT FORCE '1704.34.52393';

```

4. Should this solution fail, clean out these dummy pending transaction entries and then re-enable recovery of distributed transactions:

[Click here to view code image](#)

```

SQL> DELETE FROM pending_trans$ WHERE
local_tran_id='1704.34.52393';
SQL> DELETE FROM pending_sessions$ WHERE
local_tran_id='1704.34.52393';
SQL> COMMIT;
SQL> ALTER SYSTEM ENABLE DISTRIBUTED RECOVERY;

```

5. Use the following command to purge the lost distributed transaction:

[Click here to view code image](#)

```
SQL> EXEC DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('69.24.6020053');
```

Transaction Hangs after COMMIT or ROLLBACK

In this final situation, a distributed transaction is in the PREPARED state; however, any attempt to COMMIT or ROLLBACK this transaction causes the command to hang, and the **free global transaction table entry** wait event appears in dynamic view V\$SESSION\_WAIT. Issuing the following commands will typically resolve this problem:

- 1.** Check the transaction ID and its status; if it is indeed in a PREPARED state, it fits this scenario:

[Click here to view code image](#)

```
SQL> SELECT local_tran_id, state FROM dba_2pc_pending;
```

| LOCAL_TRAN_ID | STATE |
|---------------|----------|
| 7.12.102 | prepared |

- 2.** Verify the x\$ktuxe table to see if the transaction really exists in database:

[Click here to view code image](#)

```
SQL> SELECT
  KTUXEUSN,
  KTUXESLT,
  KTUXESQN,
  KTUXESTA Status,
  KTUXECFL Flags
  FROM x$ktuxe
 WHERE ktuxesta != 'INACTIVE'
      AND ktuxeusn= 7; --HERE WE MUST CHANGE TO THE UNDO BLOCK. IN
THIS CASE IT'S NUMBER 7
```

| KTUXEUSN | KTUXESLT | KTUXESQN | STATUS | FLAGS |
|----------|----------|----------|----------|-----------------------|
| 7 | 12 | 102 | PREPARED | SCO COL REV DEA |

- 3.** Try to COMMIT or ROLLBACK this transaction; however, issuing either of these commands results in a hung session:

[Click here to view code image](#)

```
SQL> rollback force '7.12.102';
```

```
. . .
{ Session HANGs }
```

```
SQL> commit force '7.12.102';
```

```
. . .
{ Session HANGs }
```

- 4.** Review the contents of dynamic view GV\$SESSION\_WAIT for this session. It will reveal that the session is now waiting on the **free global transaction table entry** event:

[Click here to view code image](#)

```
SQL> SELECT event
```

```
FROM gv$session_wait
WHERE event LIKE '%free%global%entry%';
```

5. Use the following query to isolate the transaction causing the issue:

[Click here to view code image](#)

```
SQL> SELECT a.sql_text, s.osuser, s.username,s.sid
      FROM v$transaction t, v$session s, v$sqlarea a
     WHERE s.taddr = t.addr
       AND a.address = s.prev_sql_addr
       AND t.xidusn = 7
       AND t.xidslot =12
       AND t.xidsqn = 102;
```

6. Attempt to purge this transaction using procedure PURGE\_LOST\_DB\_ENTRY of package DBMS\_TRANSACTION. This results in a different and unexpected error:

[Click here to view code image](#)

```
SQL> EXECUTE
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('7.12.102');COMMIT;
*
ERROR at line 1:
ORA-06510: PL/SQL: unhandled user-defined exception
ORA-06512: at "SYS.DBMS_TRANSACTION", line 94
ORA-06512: at line 1
```

7. Here is where things get a little complicated! To fix this issue, first delete the transaction from internal Oracle database tables:

[Click here to view code image](#)

```
SQL> DELETE FROM sys.pending_sub_sessions$ WHERE local_tran_id =
='7.12.102';
SQL> DELETE FROM sys.pending_sessions$ WHERE local_tran_id =
'7.12.102';
SQL> DELETE FROM sys.pending_trans$ WHERE local_tran_id =
'7.12.102';
SQL> COMMIT;
```

8. Once the transaction has been successfully deleted, insert a fake record into PENDING\_TRANS, as done in the previous scenario:

[Click here to view code image](#)

```
--Disable distributed recovery to not let the RECO background
process work on this dummy row
SQL> ALTER SYSTEM DISABLE DISTRIBUTED RECOVERY;

SQL> INSERT INTO pending_trans$ (
local_tran_id,
global_tran_fmt,
global_oracle_id,
```

```

state,
status,
session_vector,
reco_vector,
type#,
fail_time,
reco_time)
VALUES (
'7.12.102', /* ONLY THIS ROWS SHOULD BE MODIFIED WITH YOUR LOCAL
TRAN ID*/
306206,
'XXXXXXXX.12345.1.2.3',
'prepared',
'P',
hextoraw( '00000001' ),
hextoraw( '00000000' ),
0,
sysdate,
sysdate );

SQL> INSERT INTO pending_sessions$
VALUES (
'7.12.102',
1,
1,
hextoraw('05004F003A1500000104'),
'C',
0,
30258592,
 '',
146
);

SQL> COMMIT;

```

9. After inserting these fake records, commit the transaction:

```
SQL> COMMIT FORCE '7.12.102';
```

10. Finally, purge the transaction from the database:

[Click here to view code image](#)

```
SQL> EXEC DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('7.12.102');
```

Monitoring Distributed Transactions

Some useful queries that help to quickly locate and assist in the repair of distributed transaction problems are shown in [Listings 20.1, 20.2, 20.3, 20.4](#), and [20.5](#).

Listing 20.1 Locating Pending Transactions

```

SET LINES 200
COL global_tran_id FORMAT A40
COL host FORMAT A40
SELECT
    local_tran_id
    ,global_tran_id
    ,state
    ,mixed
    ,host
    ,commit#
FROM dba_2pc_pending;

```

```

SET LINES 200
COL GLOBAL_TRAN_ID FORMAT A26
COL HOST FORMAT A30
COL OS_TERMINAL FORMAT A30
COL OS_USER FORMAT A20
COL TRAN_COMMENT FORMAT A10
SELECT
    local_tran_id
    ,global_tran_id
    ,state
    ,mixed
    ,advice
    ,tran_comment
    ,os_user
    ,os_terminal
    ,host
    ,db_user
FROM dba_2pc_pending;

```

Listing 20.2 Isolating DML for a Distributed Transaction

[Click here to view code image](#)

```

SELECT a.sql_text, s.osuser, s.username
  FROM v$transaction t, v$session s, v$sqlarea a
 WHERE s.taddr = t.addr
   AND a.address = s.prev_sql_addr
   AND t.xidusn = (select local_tran_id from dba_2pc_pending)
   AND t.xidslot = <second-part-of -transaction-ID>
   AND t.xidsqn = <third-part-of -transaction-ID>;

```

Listing 20.3 Constructing ROLLBACK FORCE Commands for All Distributed Transactions

[Click here to view code image](#)

```
SET HEADING OFF
SELECT 'ROLLBACK FORCE '''||local_tran_id||''';'
   FROM dba_2pc_pending;
```

Listing 20.4 Constructing Purge Commands for All Distributed Transactions

[Click here to view code image](#)

```
SELECT
  'EXECUTE
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('''||local_tran_id||''');COMMIT'
  FROM dba_2pc_pending;
```

Listing 20.5 Verifying User and SQL Text for a Single Distributed Transaction

[Click here to view code image](#)

```
SELECT a.sql_text, s.osuser, s.username
  FROM v$transaction t, v$session s, v$sqlarea a
 WHERE s.taddr = t.addr
   AND a.address = s.prev_sql_addr
   AND t.xidusn = 3335
   AND t.xidslot =14
   AND t.xidsqn = 632719;
```

Summary

This chapter illustrated some of the most common critical problems related to distributed transactions and how to resolve those problems, especially for database releases prior to Oracle Database 12c. (Oracle 12c offers new features that can help mitigate some of the challenges of distributed transactions.) It addressed such problems as ghost transactions, missing transactions, and hanging transactions and offered step-by-step resolutions. Finally, this chapter presented queries that can help you monitor, locate, and repair distributed transaction problems.

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 VLAs and XLAs, [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 also* [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](#)
 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 11g 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](#)
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](#)
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](#)
ORArchk 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 also* [Troubleshooting RAC databases](#).
 SSD writes, [392–393](#)
Performance bottlenecks
 isolating, [47–49](#)
 solving with SSDs. *See also* [SSDs \(solid-state drives\)](#).
Performance issues
 ACS, example, [76–81](#)
 indexes. *See also* [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 also* [VLDBs \(very large databases\)](#), [performance tuning](#); [XLDBs](#)

(extremely large databases), performance tuning queries. *See SQL 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](#)

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 also* [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 also* [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 11g Release 11.2.0.3.0, [123–127](#)
 Oracle 12c 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 12c, [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 12c, [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](#)

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 also* [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 also* [SGA \(system global area\)](#).
System statistics. *See also* [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 also* [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](#)

 ORAck 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](#)
`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](#)
 \_HIGHLIMIT\_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](#)



REGISTER YOUR PRODUCT at informit.com/register Access Additional Benefits and SAVE 35% on Your Next Purchase

- Download available product updates.
- Access bonus material when applicable.
- Receive exclusive offers on new editions and related products.
(Just check the box to hear from us when setting up your account.)
- Get a coupon for 35% for your next purchase, valid for 30 days. Your code will be available in your InformIT cart. (You will also find it in the Manage Codes section of your account page.)

Registration benefits vary by product. Benefits will be listed on your account page under Registered Products.

InformIT.com—The Trusted Technology Learning Source

InformIT is the online home of information technology brands at Pearson, the world's foremost education company. At InformIT.com you can

- Shop our books, eBooks, software, and video training.
- Take advantage of our special offers and promotions (informit.com/promotions).
- Sign up for special offers and content newsletters (informit.com/newsletters).
- Read free articles and blogs by information technology experts.
- Access thousands of free chapters and video lessons.

Connect with InformIT—Visit informit.com/community

Learn about InformIT community events and programs.



informIT.com
the trusted technology learning source

Addison-Wesley • Cisco Press • IBM Press • Microsoft Press • Pearson IT Certification • Prentice Hall • Que • Sams • VMware Press

ALWAYS LEARNING

PEARSON

Code Snippets

```
select
  distinct name
from
  v$event_name
where
  name like '%enq%'
order by 1;
```

```
enq: AB - ABMR process initialized
enq: AB - ABMR process start/stop
enq: AC - acquiring partition id
enq: AD - allocate AU
enq: AD - deallocate AU
enq: AD - relocate AU
enq: AE - lock
...
```

| Event | Waits | Time (s) | (ms) | Time | Wait Class |
|-----------------------------|-----------|----------|------|------|-----------------|
| eng:HW contention | 249,725 | 3,289 | 13 | 90.0 | User I/O direct |
| path write | 168,486 | 103 | 1 | 2.8 | User I/O DB CPU |
| PX qref latch | 6,392,581 | 40 | 0 | 1.1 | Other |
| PX Deq: Slave Session Stats | 18 | 1 | 51 | .0 | Other |

```
SELECT sid, event
  FROM gv$session_wait
 WHERE event LIKE '%contention%';
```

| SID | EVENT |
|-------|----------------------|
| 9426 | eng: HW - contention |
| 13050 | eng: HW - contention |

```
SELECT
    DBMS开发利用.DATA_BLOCK_ADDRESS_FILE(id2) file#
    DBMS开发利用.DATA_BLOCK_ADDRESS_BLOCK(id2) block#
FROM gv$lock
WHERE type = 'HW';
```

| FILE# | BLOCK# |
|-------|--------|
| ----- | ----- |
| 19 | 195 |

```
SELECT
    owner,
    segment_type,
    segment_name
FROM
    dba_extents
WHERE
    file_id = 19
AND
    195 between block_id
AND
    block_id + blocks - 1;
```

| OWNER | SEGMENT_TYPE | SEGMENT_NAME |
|---------|--------------|-----------------------------|
| ORABPEL | LOBSEGMENT | SYS_LOB0000181226C00029\$\$ |

```
SELECT
  owner,
  table_name,
  segment_name
  FROM
  dba_lobs
WHERE
  segment_name= 'SYS_LOB0000181226C00029$$' ;
```

| OWNER | TABLE_NAME | SEGMENT_NAME |
|--------------|--------------------------|-----------------------------|
| ACOM_BPEL_AQ | INSERT_SITE_ORDER BI_TBL | SYS_LOB0000181226C00029\$\$ |

```
SELECT
  DISTINCT bytes
  FROM dba_extents
 WHERE segment_name = 'SYS_LOB0000181226C00029$$'
   AND owner = 'ORABPEL';
```

BYTES

| BYTES |
|----------------|
| ----- |
| 1048576 |
| 8388608 |
| 65536 |

```
ALTER TABLE orabpel.insert_site_order_bi_tbl
MODIFY LOB ('SYS_LOB0000055018C00004$$')
(ALLOCATE EXTENT (SIZE 8M));
```

Top 5 Timed Events

| Event | Waits | Time (s) | Avg | Total | Call % | Wait Class |
|-------------------------------|---------------|--------------|-------------|------------|----------------------|------------|
| | | | Wait (ms) | | | |
| db file sequential read | 2,580,474 | 35,544 | 14 | 77.4 | User I/O | |
| SQL*Net more data from client | 659,140 | 5,513 | 8 ** | 12.0 | Network | |
| CPU time | | 4,540 | 9.9 | | | |
| enq: HW - contention | 88,910 | 2,890 | 33 * | 6.3 | Configuration | |
| log file sync | 777,146 | 1,688 | 2 | 3.7 | Commit | |

Enqueue Activity Snaps: 1234-1235

-> only enqueues with waits are shown

-> Enqueue stats gathered prior to 10g should not be compared with 10g data

-> ordered by Wait Time desc, Waits desc

| Enq Type | Reg | Gets
Succ | Gets
Failed | Waits | Wait
Time (s) | Avg Wait
Time (ms) |
|---|---------|--------------|----------------|--------|------------------|-----------------------|
| HW-Segment | | | | | | |
| High Water Mark | 93,860 | 93,862 | 0 | 88,226 | 2,961 | 33.56 * |
| TX-Transaction
(row lock contention) | 272 | 272 | 0 | 209 | 570 | 2,729.44 ** |
| TX-Transaction
(index contention) | 4,564 | 4,564 | 0 | 4,144 | 34 | 8.16 |
| TX-Transaction | 793,989 | 794,042 | 0 | 97 | 0 | 4.08 |

-> s -second
 -> ms - millisecond - 1000th of a second
 -> ordered by wait time desc, waits desc (idle events last)

| Event | Waits | % Time
-outs | Total Wait
Time (s) | Avg
Wait (ms) | Waits
/txn |
|--|---------------|-----------------|------------------------|------------------|---------------|
| db file sequential read | 2,580,474 | .0 | 35,544 | 14 | 3.3 |
| SQL*Net more data from client ** | 659,140 | .0 | 5,513 | 8 | 0.9 |
| enq: HW - contention | 88,910 | .0 | 2,890 | 33* | 0.1 |
| log file sync | 777,146 | .0 | 1,688 | 2 | 1.0 |
| read by other session | 103,140 | .0 | 929 | 9 | 0.1 |
| SQL*Net break/reset to client | 114,782 | .0 | 813 | 7 | 0.1 |
| enq: TX - row lock contention *** | 380 | 43.4*** | 557 | 1466* | 0.0 |
| log file parallel write | 552,663 | .0 | 394 | 1 | 0.7 |
| latch: cache buffers chains | 55,203 | .0 | 382 | 7 | 0.1 |

```
SELECT
    sql_id,
    event,
    event_id,
    time_waited,
    current_obj#,
    current_file#,
    current_block#
FROM dba_hist_active_sess_history
WHERE snap_id BETWEEN 1072 AND 1076
AND event LIKE '%HW%CONTENTION%'
AND time_waited > 0
AND current_obj# <> -1
ORDER BY time_waited, event, sql_id;
```

```
SELECT
  owner,
  object_name,
  object_type
FROM dba_objects
WHERE object_id = [current_obj# of query above];
```

```

SQL> col object_name format a30
SQL> col program format a30
SQL> col event format a30
SQL> SELECT DISTINCT CURRENT_OBJ#,o.object_name,o.owner,o.
object_type,CURRENT_BLOCK#,SESSION_STATE,SQL_ID,EVENT
from v$active_session_history a, dba_objects o
where a.current_obj# = o.object_id
and a.event like 'enq%HW%';

```

| CUR_OBJ# | OBJ_NAME | OWN | OBJECT_TYPE | CUR_BLOCK# | SESS | SQL_ID | EVENT |
|----------|----------------------------|-----|-------------|------------|---------|---------------|----------------------|
| 235738 | MLOG\$_ENI_OLTP_ ITEM_STAR | ENI | TABLE | 100747 | WAITING | 0ghshjjvf86bg | enq: HW - contention |
| 612464 | HIST_PEDIDOS | B2W | TABLE | 394731 | WAITING | 9phv0npccjqa2 | enq: HW - contention |

```
SELECT COUNT(*) , bytes/1024/1024 "MB"
  FROM dba_extents
 WHERE segment_name = 'HIST_PEDIDOS' AND owner='B2W'
 GROUP BY bytes;
```

| COUNT(*) | BYTES |
|----------|--------|
| ----- | ----- |
| 1969801 | 131072 |

```
SELECT
  'alter table'||table_owner||'.'||table_name||' modify partition'||partition_name||' lob
('||column_name ||') (allocate extent (size 131072));'
FROM dba_lob_partitions
WHERE table_name = 'HIST_PEDIDOS'
  AND partition_name like '%2014%';
```

```
CREATE TABLE test1 (col1 CLOB,col2 number)
LOB(col1) STORE AS SECUREFILE(CACHE)
tablespace TS_GG_DATA;
```

```
SYS@ORCL AS SYSDBA> SELECT COUNT(*), segment_name, segment_type
  FROM dba_extents
 WHERE segment_name = 'TEST1'
 GROUP BY segment_name, segment_type;
```

| COUNT(*) | SEGMENT_NAME | SEGMENT_TYPE |
|----------|--------------|--------------|
| 24 | TEST1 | TABLE |

```
CREATE TABLE test2 (col1 CLOB,col2 number)
LOB(col1) STORE AS BASICFILE
tablespace TS_GG_DATA;
```

```
set lines 200
col column_name for a30
SELECT
  table_name,
  column_name,
  segment_name,
  securefile
FROM dba_lobs
WHERE table_name like 'TEST%';
```

| TABLE_NAME | COLUMN_NAME | SEGMENT_NAME | SECURE |
|------------|-------------|-----------------------------|--------|
| TEST1 | COL | SYS_LOB0000088862C00001\$\$ | YES |
| TEST2 | COL1 | SYS_LOB0000088867C00001\$\$ | NO |

```
SET TIME ON
SET TIMING ON
TRUNCATE TABLE test1;
TRUNCATE TABLE test2;

-- Load TEST1
TT@ORCL > BEGIN
  FOR v_Count_1 IN 1..1000000 LOOP
    INSERT INTO TEST1(col1) VALUES ('testInsertColLOBTest2'||v_Count_1);
    commit;
  END LOOP;
END;
/

```

PL/SQL procedure successfully completed.

Elapsed: 00:03:00.40

```
-- Load TEST2
```

```
TT@ORCL > BEGIN
  FOR v_Count_1 IN 1..1000000 LOOP
    INSERT INTO TEST2(col1) VALUES ('testInsertColLOBTest2'||v_Count_1);
    commit;
  END LOOP;
END;
/

```

22:00:55 2 22:00:55 3 22:00:55 4 22:00:55 5 22:00:55 6 22:00:55 7

PL/SQL procedure successfully completed.

Elapsed: 00:09:11.61

```
TT@ORCL > BEGIN
  FOR v_Count_2 IN 1..1000 LOOP
    update TEST2 set col1 = 'testInsertColLOBTest2'||v_count_2 where rownum <100;
    commit;
  END LOOP;
END;
/
```

PL/SQL procedure successfully completed.

Elapsed: 00:00:03.93

```
TT@ORCL > BEGIN
  FOR v_Count_2 IN 1..1000 LOOP
    update TEST1 set col1 = 'testInsertColLOBTest2'||v_count_2 where rownum <100;
    commit;
  END LOOP;
END;
/
```

PL/SQL procedure successfully completed.

Elapsed: 00:00:03.62

```
-- For Oracle Database 10.2 and above:  
ALTER TABLE <table name>  
    MODIFY LOB (<lob column name>) (SHRINK SPACE [CASCADE]);  
  
-- For Oracle Database 10.1 and below:  
ALTER TABLE <table name>  
    MOVE LOB (<lob column name>) STORE AS (TABLESPACE <tablespace name>);
```

```
TT@ORCL > ALTER TABLE test2 MOVE LOB (col1) STORE AS SECUREFILE (TABLESPACE users);
```

```
Table altered.
```

```
Elapsed: 00:00:23.54
```

```
-- Create the migrating user (if it doesn't yet exist)...
grant dba to tt identified by tt123;

-- ... or grant the existing user account the necessary specific privileges
-- so it can use DBMS_REDEFINITION
grant execute on dbms_redefinition to tt;
grant alter any table to tt;
grant drop any table to tt;
grant lock any table to tt;
grant create any table to tt;
grant select any table to tt;
grant create session to tt;
```

```
CREATE TABLE test3 (
  col1 NUMBER PRIMARY KEY,
  col2 CLOB
); 23:03:10    2  23:03:10    3  23:03:10    4
```

Table created.

Elapsed: 00:00:00.11

```
TT@ORCL > BEGIN
  FOR v_Count_2 IN 1..10000 LOOP
    INSERT INTO TEST3(col1,col2) VALUES (v_Count_2,'testInsertColLOBTest3'||v_Count_2);
    commit;
  END LOOP;
END;
/
PL/SQL procedure successfully completed.
```

Elapsed: 00:00:01.82

```
TT@ORCL > CREATE TABLE test4 (
  col1 NUMBER NOT NULL,
  col2 CLOB
) LOB(col2) STORE AS SECUREFILE (NOCACHE);
```

Table created.

Elapsed: 00:00:00.05

```
23:07:09 TT@ORCL > DECLARE
  col_mapping VARCHAR2(1000);
BEGIN
  col_mapping := 'col1 col1, || '|| 'col2 col2';
  DBMS_REDEFINITION.START_REDEF_TABLE('tt', 'test3', 'test4', col_mapping);
END;
/
```

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.97

```
23:07:42 TT@ORCL > DECLARE
  error_count pls_integer := 0;
BEGIN
  DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS(
    uname=> 'tt',
    orig_table=> 'test3',
    int_table=>  'test4',
    copy_indexes=> DBMS_REDEFINITION.CONS_ORIG_PARAMS,
    copy_triggers=> TRUE,
    copy_constraints=> TRUE,
    copy_privileges=> TRUE,
    copy_statistics=> FALSE,
    num_errors=> error_count);
  DBMS_OUTPUT.PUT_LINE('errors := ' || TO_CHAR(error_count));
END;
/
```

PL/SQL procedure successfully completed.

Elapsed: 00:00:05.89

```
23:08:10 TT@ORCL > EXEC DBMS_REDEFINITION.FINISH_REDEF_TABLE('tt', 'test3', 'test4');

PL/SQL procedure successfully completed.

Elapsed: 00:00:02.52
```

```
23:18:33 TT@ORCL > select owner,table_name,column_name,securefile  
      from dba_lobs where table_name='TEST3';
```

| OWNER | TABLE_NAME | COLUMN_NAME | SEC |
|-------|------------|-------------|------------|
| TT | TEST3 | COL2 | YES |

Elapsed: 00:00:00.05

```
23:18:53 TT@ORCL >
```

```
SYS@ORCL AS SYSDBA> create table tt.test_pctfree (col1 clob) PCTFREE 40;
```

```
Table created.
```

```
SYS@ORCL AS SYSDBA> BEGIN
  FOR v_Count_2 IN 1..100000 LOOP
    INSERT INTO tt.test_pctfree
      VALUES ('testInsertColLOBTest2'||v_count_2);
    COMMIT;
  END LOOP;
END;
/
```

```
SYS@ORCL AS SYSDBA> select count(*) from tt.test_pctfree;
```

| COUNT(*) |
|----------|
| 100000 |

```
SYS@ORCL AS SYSDBA> EXEC DBMS_STATS.GATHER_TABLE_STATS ('TT', 'TEST_PCTFREE');
```

```
PL/SQL procedure successfully completed.
```

```
SYS@ORCL AS SYSDBA>
select
    blocks,
    avg_space,
    pct_free
from
    dba_tables
where
    table_name='TEST_PCTFREE';
```

| BLOCKS | AVG_SPACE | PCT_FREE |
|-------------|-------------|-----------|
| ----- | ----- | ----- |
| 1504 | 3362 | 40 |

```
SYS@ORCL AS SYSDBA> truncate table tt.test_pctfree;
```

Table truncated.

Elapsed: 00:00:00.07

```
SYS@ORCL AS SYSDBA> alter table tt.test_pctfree pctfree 0;
```

Table altered.

Elapsed: 00:00:00.01

```
SYS@ORCL AS SYSDBA> BEGIN
  FOR v_Count_2 IN 1..100000 LOOP
    INSERT INTO tt.test_pctfree VALUES ('testInsertColLOBTest2'||v_count_2);
    COMMIT;
  END LOOP;
END;
/
```

PL/SQL procedure successfully completed.

Elapsed: 00:00:14.13

```
SYS@ORCL AS SYSDBA> EXEC DBMS_STATS.GATHER_TABLE_STATS ('TT', 'TEST_PCTFREE');
```

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.22

```
SYS@ORCL AS SYSDBA>
```

```
SYS@ORCL AS SYSDBA> select
  blocks,
  avg_space,
  pct_free
from
  dba_tables
where
  table_name='TEST_PCTFREE';
```

| 2
BLOCKS | 3 | 4
AVG_SPACE | 5 | 6 | 7 | 8
PCT_FREE |
|-------------|---|----------------|---|-------|---|---------------|
| ----- | | ----- | | ----- | | ----- |
| | | 874 | | 303 | | 0 |

Elapsed: 00:00:00.00

```
SQL> alter session set "_kdli_sio_fileopen"='nodsync';
```

```
SQL> SELECT DBMS_UNDO_ADV.getLongest_query(SYSDATE-1/24, SYSDATE)
  AS best_undo_time FROM dual;
```

```
BEST_UNDO_TIME
```

```
-----  
845
```

```
SQL> SELECT DBMS_UNDO_ADV.REQUIRED_RETENTION(SYSDATE-30, SYSDATE)
AS reqd_retn FROM dual;
```

```
REQD_RETN
```

```
-----  
1699
```

```
SQL> SET HEADING OFF
SELECT 'commit force '''||local_tran_id||''';' FROM dba_2pc_pending;
SQL> SQL>
commit force '151.23.987365';
commit force '155.29.1615583';
commit force '231.10.1069716';
commit force '237.18.648972';
commit force '238.15.811599';
commit force '36.5.1329177';
commit force '393.41.746115';
commit force '4733.28.915649';
commit force '613.17.686683';
```

9 rows selected.

```
SQL> commit force '151.23.987365';
commit force '155.29.1615583';
commit force '231.10.1069716';
commit force '237.18.648972';
commit force '238.15.811599';
commit force '36.5.1329177';
commit force '393.41.746115';
commit force '4733.28.915649';
commit force '613.17.686683';
Commit complete.
Commit complete.
.
.
.
Commit complete.
```

```
SQL> SELECT 'Execute DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY(''':''||local_tran_id||'');commit;'  
  FROM dba_2pc_pending;  
  
Execute DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('151.23.987365');commit;  
Execute DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('155.29.1615583');commit;  
Execute DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('231.10.1069716');commit;  
Execute DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('237.18.648972');commit;  
Execute DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('238.15.811599');commit;  
Execute DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('36.5.1329177');commit;  
Execute DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('393.41.746115');commit;  
Execute DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('4733.28.915649');commit;  
Execute DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('613.17.686683');commit;  
  
9 rows selected.
```

```
SQL> SELECT
  'ALTER ROLLBACK SEGMENT "'||segment_name||'" ONLINE;'
  FROM dba_rollback_segs
 WHERE status LIKE 'PARTLY_AVAILABLE';
```

```
SQL> ALTER ROLLBACK SEGMENT "_SYSSMU168$" OFFLINE;  
  
SQL> SELECT status  
  FROM dba_rollback_segs  
 WHERE SEGMENT_NAME = '_SYSSMU168$';
```

```
SQL> DROP ROLLBACK SEGMENT "_SYSSMU168$";  
SQL> CREATE ROLLBACK SEGMENT '_SYSSMU168$' TABLESPACE UNDOTBS1;
```

```
ALTER SESSION SET EVENTS 'immediate trace name HANGANALYZE level 3';
```

```
ALTER SESSION SET EVENTS 'immediate trace name HANGANALYZE level 3';
```

```
ALTER SESSION SET EVENTS 'immediate trace name HANGANALYZE level 3';
```

```
ALTER SESSION SET EVENTS 'immediate trace name SYSTEMSTATE level 266';
```

```
ALTER SESSION SET EVENTS 'immediate trace name SYSTEMSTATE level 266';
```

```
ALTER SESSION SET EVENTS 'immediate trace name SYSTEMSTATE level 266';
```

```
ALTER SYSTEM SET EVENT "10015 trace name context forever, level 10";
```

```
SQL> SELECT owner, object_name, object_type, status
  FROM dba_objects
 WHERE object_id = <object# from trace file>;
```

```
[root@ex01dbadm01 tmp]# cd
[root@ex01dbadm01 ~]# mkdir ORAchk
[root@ex01dbadm01 ~]# cd ORAchk/
[root@ex01dbadm01 ORAchk]# unzip /tmp/orachk.zip
Archive: /tmp/orachk.zip
  inflating: UserGuide.txt
  inflating: collections.dat
  inflating: rules.dat
  ...
  inflating: .cgrep/lcgrep6
  inflating: .cgrep/profile_only.dat
  inflating: .cgrep/auto_upgrade_check.pl
  inflating: .cgrep/diff_collections.pl
  inflating: CollectionManager_App.sql
  inflating: orachk
```

```
[root@ex01dbadm01 ORAChk]# ./orachk
This version of orachk was released on 09-Oct-2014 and its older than 120 days. No new version
of orachk is available in RAT_UPGRADE_LOC. It is highly recommended that you download the
latest version of orachk from my oracle support to ensure the highest level of accuracy of the
data contained within the report.
```

```
Do you want to continue running this version? [y/n] [y]
```

```
CRS stack is running and CRS_HOME is not set. Do you want to set CRS_HOME to /u01/app/11.2.0.4/
grid? [y/n] [y]
```

```
...
```

```
[root@ex01dbadm01 ORAchk]#  
./orachk -set "AUTORUN_SCHEDULE=3 1 * *;NOTIFICATION_EMAIL=paulo.portugal@f2c.com.br"  
  
Created AUTORUN_SCHEDULE for ID[orachk.default]  
  
Created NOTIFICATION_EMAIL for ID[orachk.default]  
  
[root@ex01dbadm01 ORAchk]#
```

```
AUTORUN_SCHEDULE * * * *      :- Automatic run at specific time in daemon mode.  
- - - -  
? ? ? ?  
? ? ? +----- day of week (0 - 6) (0 to 6 are Sunday to Saturday)  
? ? +----- month (1 - 12)  
? +----- day of month (1 - 31)  
+----- hour (0 - 23)
```

```
[root@ex01dbadm01 ORAchk]# ./orachk -get all  
  
ID: orachk.default  
-----  
AUTORUN_SCHEDULE = 3 1 * *  
NOTIFICATION_EMAIL = paulo.portugal@f2c.com.br  
[root@ex01dbadm01 ORAchk]#
```

...
RECOMMENDATION 1: Schema, 84.4% benefit (17609 seconds)
ACTION: Consider partitioning the INDEX "MID\_B2W\_ADMIN.STM\_LOG\_DATA\_IDX"
with object id 131712 in a manner that will evenly distribute
concurrent DML across multiple partitions.
RELEVANT OBJECT: database object with id 131712
RATIONALE: The INSERT statement with SQL\_ID "fv4un8f4w6zg8" was
significantly affected by "buffer busy" waits.
RELEVANT OBJECT: SQL statement with SQL\_ID fv4un8f4w6zg8
insert into STM\_LOG (NM\_LOGIN, DS\_ROLES, DS\_OPERATION, DT\_CRIACAO,
CD\_MARCA, CD\_LOG) values (:1, :2, :3, :4, :5, :6)
...

gc cr block receive time=

Time to send message to a remote LMS process by FG

- + Time taken by LMS to build block (statistics: gc cr block build time)
- + LMS wait for LGWR latency (statistics:gc cr block flush time)
- + LMS send time (Statistics: gc cr block send time)
- + Wire latency

Top User EventsDB/Inst: BWMDPR/BWMDPR1 (Feb 25 17:40 to 17:55)

| Event | Event Class | % Activity | Avg Active Sessions |
|-------------------------|-------------|------------|---------------------|
| CPU + Wait for CPU | CPU | 51.49 | 2.49 |
| db file sequential read | User I/O | 13.17 | 0.64 |
| gc buffer busy | Cluster | 5.63 | 0.27 |
| direct path read | User I/O | 3.61 | 0.17 |
| db file scattered read | User I/O | 3.17 | 0.15 |

Top Blocking Sessions DB/Inst: BWMDPR/BWMDPR1 (Feb 25 17:40 to 17:55)
-> Blocking session activity percentages are calculated with respect to
waits on enqueue, latches and "buffer busy" only
-> '% Activity' represents the load on the database caused by
a particular blocking session
-> '# Samples Active' shows the number of ASH samples in which the
blocking session was found active.
-> 'XIDs' shows the number of distinct transaction IDs sampled in ASH
when the blocking session was found active.

| Blocking Sid % Activity Event Caused | | % Event | |
|--------------------------------------|----------------------------|------------------|------|
| User | Program | # Samples Active | XIDs |
| 5074, 38287 | 1.15 gc buffer busy | 0.80 | |
| MID102_B2W_WL_APP | | 165/901 [18%] | 0 |
| 4375, 33093 | 1.13 read by other session | 0.85 | |

Top DB Objects DB/Inst: BWMDPR/BWMDPR1 (Feb 25 17:40 to 17:55)
-> With respect to Application, Cluster, User I/O and buffer busy waits only.

| Object ID & Activity Event | % Event |
|--|---------------------------|
| Object Name (Type) | Tablespace |
| 131712 4.37 gc buffer busy | 2.62 |
| MID_B2W_ADMIN.STM_LOG_DATA_IDX (INDEX) | TS_MID_B2W_ADMIN_INDEX_M_ |
| gc current block busy | 1.36 |

```

SELECT * FROM (
  SELECT
    h.event "Wait Event",
    SUM(h.wait_time + h.time_waited)/1000000 "Total Wait Time"
  FROM v$active_session_history h,
       v$event_name e
  WHERE h.sample_time < (SELECT MAX(sample_time)
                           FROM v$active_session_history)
        AND h.sample_time > (SELECT MAX(sample_time) - 1/24
                               FROM v$active_session_history)
        AND h.event_id = e.event_id
        AND e.wait_class <>'IDLE'
  GROUP BY h.event
  ORDER BY 2 DESC)
  WHERE ROWNUM <10;

```

| Wait Event | Total Wait Time |
|-------------------------------|-----------------|
| gc buffer busy | 40.23931 |
| enq: TX - row lock contention | 32.385347 |
| enq: TX - index contention | 28.62571 |
| gc current block busy | 25.963209 |
| db file sequential read | 14.387571 |
| LNS wait on SENDREQ | 13.18233 |
| gc cr multi block request | 12.478076 |
| reliable message | 5.038086 |
| cr request retry | 4.887495 |

```

COL object_name FORMAT A30
COL program      FORMAT A30
COL event        FORMAT A30
SELECT DISTINCT
    current_obj#,
    o.object_name,
    o.owner,
    o.object_type,
    current_file#,
    session_state,
    sql_id,
    event
FROM v$active_session_history a,
     dba_objects o
WHERE a.current_obj# = o.object_id
  AND a.event LIKE '%gc buffer busy%';

```

| CURR_
OBJ# | OBJECT_
NAME | OWNER | OBJ_
TYPE | CURR_
FILE# | SESSION | SQL_ID | EVENT |
|---------------|-----------------------|----------------|--------------|----------------|---------|---------------|----------------|
| 104830 | STM_DETALHE_LOG_VALOR | MID_B2W _ADMIN | INDEX | 445 | WAITING | 39j77bgam9506 | gc buffer busy |
| 79829 | STM_ITEM | MID_B2W _ADMIN | TABLE | 140 | WAITING | 0ba26mnwcv7x2 | gc buffer busy |
| 55681 | STM_ITEM GROUP | MID_B2W _ADMIN | TABLE | 141 | WAITING | 9apqp7fwnqsy7 | gc buffer busy |
| 131712 | STM_LOG_DATA_IDX | MID_B2W _ADMIN | INDEX | 445 | WAITING | fv4un8f4w6zg8 | gc buffer busy |

```
SQL> select sql_text from v$sqltext where sql_id='39j77bgam9506' order by piece;
```

```
SQL_TEXT
```

```
-----  
insert into STM_DETALHE_LOG (CD_LOG, DS_ATRIBUTO, DS_VALOR, TP_D  
ETALHE_LOG, CD_DETALHE_LOG) values (:1, :2, :3, :4, :5)
```

```
col min  for a30
col max  for a30
SQL> SELECT
      MIN(begin_interval_time) min,
      MAX(end_interval_time) max
    FROM dba_hist_snapshot
   WHERE snap_id BETWEEN 54657 AND 54658;
```

| MIN | MAX |
|---------------------------|---------------------------|
| ----- | ----- |
| 28-FEB-15 09.00.15.104 AM | 28-FEB-15 11.00.04.693 AM |

```
SQL> SELECT
  wait_class_id,
  wait_class,
  COUNT(*) cnt
  FROM dba_hist_active_sess_history
 WHERE snap_id BETWEEN 54657 AND 54659
 GROUP BY wait_class_id, wait_class
 ORDER BY 3;
```

| WAIT_CLASS_ID | WAIT_CLASS | CNT |
|---------------|--------------------|------|
| 4166625743 | Administrative | 23 |
| 3290255840 | Configuration | 23 |
| 3386400367 | Commit | 111 |
| 4217450380 | Application | 147 |
| 2000153315 | Network | 233 |
| 4108307767 | System I/O | 236 |
| 3875070507 | Other | 544 |
| 1893977003 | Cluster | 633 |
| 1740759767 | User I/O | 1019 |
| 3871361733 | Concurrency | 3402 |

11 rows selected.

```
SELECT
    event_id,
    event,
    COUNT(*) cnt
  FROM dba_hist_active_sess_history
 WHERE snap_id BETWEEN 54657 AND 54659
   AND wait_class_id = 3871361733
 GROUP BY event_id, event
 ORDER BY 3;
```

| EVENT_ID | EVENT | CNT |
|------------|------------------------|------|
| ... | | |
| 2277737081 | gc current grant busy | 147 |
| 3046984244 | gc cr block 3-way | 194 |
| 737661873 | gc cr block 2-way | 318 |
| 111015833 | gc current block 2-way | 451 |
| 2701629120 | gc current block busy | 473 |
| 1478861578 | gc buffer busy | 1333 |

24 rows selected.

```
SELECT
    sql_id,
    COUNT(*) cnt
        FROM dba_hist_active_sess_history
    WHERE snap_id BETWEEN 54657 AND 54659
        AND event_id = 1478861578
    GROUP BY sql_id
    HAVING COUNT(*)>1
    ORDER BY 2;
```

| SQL_ID | CNT |
|----------------------|-----|
| ... | |
| 0s34c5d0n7577 | 48 |
| 5bwdfzr1s4cx0 | 70 |
| Gppjjfhgxbxx | 94 |
| fv4un8f4w6zg8 | 690 |
| 52 rows selected. | |

```
SQL> SELECT
      DISTINCT sql_text
        FROM gv$sqltext
       WHERE sql_id = 'fv4un8f4w6zg8';

SQL_TEXT
-----
insert into STM_LOG (NM_LOGIN, DS_ROLES, DS_OPERATION, DT_CRIACAO, CD_MARCA, CD_LOG) values (:1, :2, :3, :4, :5, :6)
```

```

SQL> select * from v$version;

BANNER
-----
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production
PL/SQL Release 12.1.0.1.0 - Production
CORE    12.1.0.1.0      Production
TNS for 64-bit Windows: Version 12.1.0.1.0 - Production
NLSRTL Version 12.1.0.1.0 - Production

SQL> create table t_acs(n1  number, n2 number);

SQL> BEGIN
  for j in 1..1200150 loop
    if j = 1 then
      insert into t_acs values (j, 1);
    elsif j>1 and j<=101 then
      insert into t_acs values(j, 100);
    elsif j>101 and j<=1101 then
      insert into t_acs values (j, 1000);
    elsif j>10001 and j<= 110001 then
      insert into t_acs values(j,10000);
    else
      insert into t_acs values(j, 1000000);
    end if;
  end loop;
  commit;
END;
/
SQL> create index t_acs_i1 on t_acs(n2);

SQL> BEGIN
  dbms_stats.gather_table_stats
    (
      user
      , 't_acs'
      , method_opt => 'for all columns size 1'
      , cascade => true
      , estimate_percent => dbms_stats.auto_sample_size
    );
END;
/

```

```
SQL> select n2, count(1) from t_acs group by n2 order by 2;
```

| N2 | COUNT(1) |
|---------|----------|
| 1 | 1 |
| 100 | 100 |
| 1000 | 1000 |
| 10000 | 100000 |
| 1000000 | 1099049 |

```

SQL> var ln2 number;
SQL> exec :ln2 := 100;

SQL> select count(1) from t_acs where n2 <= :ln2;

      COUNT(1)
-----
          101

SQL> select * from table(dbms_xplan.display_cursor);

-----  

| Id | Operation           | Name | Rows | Bytes |
|----|----|----|----|----|
| 0 | SELECT STATEMENT    |       |       |       |
| 1 |   SORT AGGREGATE    |       | 1    | 3     |
|* 2 |   TABLE ACCESS FULL | T_ACS | 397K| 1165K |
-----  
  

Predicate Information (identified by operation id):
-----  

  2 - filter("N2" <=:LN2)

SQL> select
        sql_id
      ,child_number
      ,is_bind_sensitive
  from
        v$sql
 where
        sql_id = 'ct0yv82p15jdw';

SQL_ID      CHILD_NUMBER IS_BIND_SENSITIVE
-----  -----
ct0yv82p15jdw          0  Y

```

```

SQL> alter system flush shared_pool;

SQL> exec dbms_stats.delete_table_stats(user,'t_acs');

SQL> select count(1) from t_acs where n2 <= :ln2;

SQL> select * from table(dbms_xplan.display_cursor);

SQL_ID  ct0yv82p15jdw, child number 0
-----
| Id  | Operation          | Name      | Rows  | Bytes |
|-----|
|   0 | SELECT STATEMENT   |           |        |        |
|   1 |   SORT AGGREGATE   |           |     1  |    13  |
| * 2 |     INDEX RANGE SCAN| T_ACS_I1  |  101  | 1313  |
|-----|
Predicate Information (identified by operation id):
-----
2 - access ("N2" <=:LN2)

```

Note

- dynamic sampling used for this statement (level=2)

```

SQL> select
      sql_id
    ,child_number
    ,is_bind_sensitive
  from
    v$sql
  where
    sql_id = 'ct0yv82p15jdw';

```

| SQL_ID | CHILD_NUMBER | IS_BIND_SENSITIVE |
|---------------|--------------|-------------------|
| ct0yv82p15jdw | 0 | N |

```
SQL> BEGIN
      dbms_stats.gather_table_stats
        (user
         , 't_acs'
         , method_opt => 'for all columns size auto'
         , cascade => true
         , estimate_percent => dbms_stats.auto_sample_size
        );
END;
/

```

```
SQL> SELECT
      column_name,
      histogram
    FROM user_tab_col_statistics
   WHERE table_name = 'T_ACS'
     AND column_name = 'N2';
```

| COLUMN_NAME | HISTOGRAM |
|-------------|-----------|
| N2 | FREQUENCY |

```
SQL> select count(1) from t_acs where n2 = :ln2;
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

```
SQL_ID  f2pmwazy1rnfd, child number 0
-----
```

| Id | Operation | Name | Rows | Bytes |
|-----|------------------|----------|------|-------|
| 0 | SELECT STATEMENT | | | |
| 1 | SORT AGGREGATE | | 1 | 3 |
| * 2 | INDEX RANGE SCAN | T_ACS_I1 | 1372 | 4116 |

Predicate Information (identified by operation id):

2 - access("N2"=:LN2)

```
SQL> select
      sql_id
    ,child_number
    ,is_bind_sensitive
  from
    v$sql
 where
  sql_id = 'f2pmwazy1rnfd';
```

| SQL_ID | CHILD_NUMBER | IS_BIND_SENSITIVE |
|---------------|--------------|-------------------|
| f2pmwazy1rnfd | 0 | Y |

```
SQL> create table t_acs_part
      (n1 number, n2 number)
partition by range (n2)
(partition p1 values less than (100)
,partition p2 values less than (1000)
,partition p3 values less than (10000)
,partition p4 values less than (100000)
,partition p5 values less than (1000000)
,partition p6 values less than (10000000)
);
```

```
SQL> BEGIN
      for j in 1..1200150 loop
        if j = 1 then
          insert into t_acs_part values (j, 1);
        elsif j>1 and j<=101 then
          insert into t_acs_part values(j, 100);
        elsif j>101 and j<=1101 then
          insert into t_acs_part values (j, 1000);
        elsif j>10001 and j<= 110001 then
          insert into t_acs_part values(j,10000);
        else
          insert into t_acs_part values(j, 1000000);
        end if;
      end loop;
      commit;
```

```
END;
/
SQL> BEGIN
      dbms_stats.gather_table_stats
        (user
         , 't_acs_part'
         ,method_opt => 'for all columns size 1'
         ,cascade => true
         ,estimate_percent => dbms_stats.auto_sample_size
        );
END;
/
SQL> SELECT
      column_name,
      histogram
    FROM user_tab_col_statistics
   WHERE table_name = 'T_ACS_PART'
     AND column_name  = 'N2';

COLUMN_NAME  HISTOGRAM
-----
N2          NONE
```

```
SQL> select count(1) from t_acs_part where n2 = :ln2;
```

```
COUNT(1)
```

```
-----  
100
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

```
SQL_ID byztzuffb65n9, child number 0
```

| Id | Operation | Name | Rows | Pstart | Pstop |
|----|------------------------|-----------|------|--------|-------|
| 0 | SELECT STATEMENT | | | | |
| 1 | SORT AGGREGATE | | 1 | | |
| 2 | PARTITION RANGE SINGLE | | 100 | KEY | KEY |
| * | 3 TABLE ACCESS FULL | T_ACSPART | 100 | KEY | KEY |

```
Predicate Information (identified by operation id):
```

```
3 - filter("N2"=:LN2)
```

```
SQL> select  
      sql_id  
    , child_number  
    , is_bind_sensitive  
  from  
    v$sql  
 where  
   sql_id = 'byztzuffb65n9';
```

```
SQL_ID CHILD_NUMBER IS_BIND_SENSITIVE
```

```
byztzuffb65n9 0 Y
```

```
SQL> select n2, count(1) from t_acs group by n2 order by 2;
```

| N2 | COUNT(1) |
|---------|----------|
| 1 | 1 |
| 100 | 100 |
| 1000 | 1000 |
| 10000 | 100000 |
| 1000000 | 1099049 |

```
SQL> alter system flush shared_pool;
SQL> exec :ln2 := 100
SQL> select count(1) from t_acs where n2 = :ln2;
COUNT(1)
-----
100

SQL> select * from table(dbms_xplan.display_cursor);
SQL_ID  f2pmwazy1rnfd, child number 0
-----
| Id  | Operation          | Name      | Rows  | Bytes |
|---|---|---|---|---|
| 0  | SELECT STATEMENT   |           |       |       |
| 1  |   SORT AGGREGATE   |           | 1    | 3     |
|* 2 |   INDEX RANGE SCAN | T_ACS_I1  | 1372 | 4116 |
-----
```

Predicate Information (identified by operation id):

```
-----  
2 - access("N2"=:LN2)
```

```
SQL> exec :ln2 := 1000000
```

```
SQL> select count(1) from t_acs where n2 = :ln2;
```

```
COUNT(1)
```

```
-----  
1099049
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

```
SQL_ID  f2pmwazylrnfd, child number 0
```

| Id | Operation | Name | Rows | Bytes |
|----|------------------|----------|------|-------|
| 0 | SELECT STATEMENT | | | |
| 1 | SORT AGGREGATE | | 1 | 3 |
| * | INDEX RANGE SCAN | T_ACS_I1 | 1372 | 4116 |

```
Predicate Information (identified by operation id):
```

```
-----  
2 - access ("N2"=:LN2)
```

```
SQL> select
      sql_id
    ,child_number
  ,is_bind_sensitive
  ,is_bind_aware
  ,is_shareable
  from
    v$sql
 where
  sql_id = 'f2pmwazy1rnfd';
```

| SQL_ID | CHILD_NUMBER | IS_BIND_SENSITIVE | IS_BIND_AWARE | IS_SHAREABLE |
|---------------|--------------|-------------------|---------------|--------------|
| f2pmwazy1rnfd | 0 | Y | N | Y |

```
SQL> select count(1) from t_acs where n2 = :ln2;
```

```
COUNT(1)
```

```
-----  
1099049
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

```
SQL_ID  f2pmwazy1rnfd, child number 1
```

| Id | Operation | Name | Rows | Bytes |
|----|-------------------|-------|-------|-------|
| 0 | SELECT STATEMENT | | | |
| 1 | SORT AGGREGATE | | 1 | 3 |
| * | TABLE ACCESS FULL | T_AC斯 | 1096K | 3212K |

```
Predicate Information (identified by operation id):
```

```
-----  
2 - filter("N2"=:LN2)
```

```
SQL> select
      sql_id
    , child_number
    , is_bind_sensitive
    , is_bind_aware
    , is_shareable
  from
    v$sql
 where
  sql_id = 'f2pmwazy1rnfd';
```

| SQL_ID | CHILD_NUMBER | IS_BIND_SENSITIVE | IS_BIND_AWARE | IS_SHAREABLE |
|---------------|--------------|-------------------|---------------|--------------|
| f2pmwazy1rnfd | 0 | Y | N | N |
| f2pmwazy1rnfd | 1 | Y | Y | Y |

```

SQL> exec :ln2 := 1000

SQL> select count(1) from t_acs where n2 = :ln2;

COUNT(1)
-----
1000

SQL> select * from table(dbms_xplan.display_cursor);

SQL_ID  f2pmwazy1rnfd, child number 2
-----
| Id  | Operation          | Name    | Rows  | Bytes |
|---|---|---|---|---|
| 0  | SELECT STATEMENT   |          |       |         |
| 1  |   SORT AGGREGATE   |          |       1 |       3 |
|* 2  |   INDEX RANGE SCAN | T_ACS_I1 | 2747 | 8241  |
-----


Predicate Information (identified by operation id):
-----
2 - access("N2"=:LN2)

SQL> select
      sql_id
    ,child_number
    ,is_bind_sensitive
    ,is_bind_aware
    ,is_shareable
  from
    v$sql
 where
   sql_id = 'f2pmwazy1rnfd';

SQL_ID      CHILD_NUMBER IS_BIND_SENSITIVE IS_BIND_AWARE      IS_SHAREABLE
-----      -----          -----          -----          -----
f2pmwazy1rnfd        0 Y             N             N
f2pmwazy1rnfd        1 Y             Y             Y
f2pmwazy1rnfd        2 Y             Y             Y

```

```

SQL> create table t1
  as select
        rownum n1
      ,trunc((rownum -1)/3) n2
  from dual
 connect by level <=1e3;

SQL> exec dbms_stats.gather_table_stats(user, 't1');

SQL> select /*+ bind_aware */
  count(*)
  from t1
 where n2 = :ln2;

COUNT(*)
-----
0
SQL> select * from table(dbms_xplan.display_cursor);

SQL_ID 5gz8nu7ru5gh0, child number 0
-----
| Id  | Operation          | Name | Rows | Bytes |
|---|---|---|---|---|
| 0  | SELECT STATEMENT   |       |       |       |
| 1  |   SORT AGGREGATE   |       |     1 |       4 |
|* 2  |   TABLE ACCESS FULL| T1   |     1 |       4 |
-----


Predicate Information (identified by operation id):
-----
2 - filter("N2"=:LN2)

SQL> SQL> select
      sql_id
    ,child_number
    ,is_bind_sensitive
    ,is_bind_aware
    ,is_shareable
  from
    v$sql
 where
  sql_id = '5gz8nu7ru5gh0';

SQL_ID      CHILD_NUMBER IS_BIND_SENSITIVE IS_BIND_AWARE      IS_SHAREABLE
-----      -----          -----          -----          -----
5gz8nu7ru5gh0          0      Y                  Y              Y

```

```
SQL> select n2, count(1) from t_acs group by n2 order by 2;
```

| N2 | COUNT(1) |
|---------|----------|
| 1 | 1 |
| 100 | 100 |
| 1000 | 1000 |
| 10000 | 100000 |
| 1000000 | 1099049 |

```
SQL> desc v$sql_cs_statistics
```

| | Name | Null? | Type |
|----|-----------------------|-------|--------------|
| 1 | ADDRESS | | RAW(8) |
| 2 | HASH_VALUE | | NUMBER |
| 3 | SQL_ID | | VARCHAR2(13) |
| 4 | CHILD_NUMBER | | NUMBER |
| 5 | BIND_SET_HASH_VALUE | | NUMBER |
| 6 | PEEKED | | VARCHAR2(1) |
| 7 | EXECUTIONS | | NUMBER |
| 8 | ROWS_PROCESSED | | NUMBER |
| 9 | BUFFER_GETS | | NUMBER |
| 10 | CPU_TIME | | NUMBER |
| 11 | CON_ID | | NUMBER |

```
SQL> desc v$sql_cs_histogram
```

| | Name | Null? | Type |
|---|--------------|-------|--------------|
| 1 | ADDRESS | | RAW(8) |
| 2 | HASH_VALUE | | NUMBER |
| 3 | SQL_ID | | VARCHAR2(13) |
| 4 | CHILD_NUMBER | | NUMBER |
| 5 | BUCKET_ID | | NUMBER |
| 6 | COUNT | | NUMBER |
| 7 | CON_ID | | NUMBER |

```
SQL> desc v$sql_cs_selectivity
```

| | Name | Null? | Type |
|---|--------------|-------|--------------|
| 1 | ADDRESS | | RAW(8) |
| 2 | HASH_VALUE | | NUMBER |
| 3 | SQL_ID | | VARCHAR2(13) |
| 4 | CHILD_NUMBER | | NUMBER |
| 5 | PREDICATE | | VARCHAR2(40) |
| 6 | RANGE_ID | | NUMBER |
| 7 | LOW | | VARCHAR2(10) |
| 8 | HIGH | | VARCHAR2(10) |
| 9 | CON_ID | | NUMBER |

```
SQL> alter system flush shared_pool;  
  
SQL> exec :ln2 := 100;  
  
SQL> select count(1) from t_acs where n2 = :ln2;  
COUNT(1)  
-----  
      100
```

```
SQL> select
      child_number
    , executions
    , rows_processed
  from v$sql_cs_statistics
 where sql_id = 'f2pmwazy1rnfd' ;
```

```
no rows selected
```

```
SQL> select
      child_number
    , bucket_id
    , count
  from
    v$sql_cs_histogram
 where  sql_id = 'f2pmwazy1rnfd' ;

CHILD_NUMBER  BUCKET_ID      COUNT
-----  -----  -----
          0          0      1 → incremented because processed rows <1000
          0          1          0
          0          2          0
```

```
SQL> select count(1) from t_acs where n2 = :ln2;
SQL> select count(1) from t_acs where n2 = :ln2;
SQL> select
      child_number
     ,bucket_id
     ,count
   from
     v$sql_cs_histogram
  where  sql_id = 'f2pmwazy1rnfd' ;

CHILD_NUMBER  BUCKET_ID      COUNT
-----  -----  -----
          0          0      3 → incremented because processed rows <1000
          0          1          0
          0          2          0
```

```

SQL> exec :ln2 := 10000;

SQL> select count(1) from t_acs where n2 = :ln2;
   COUNT(1)
-----
100000

SQL> select
      child_number
     ,bucket_id
     ,count
  from
    v$sql_cs_histogram
 where  sql_id = 'f2pmwazy1rnfd' ;

CHILD_NUMBER  BUCKET_ID      COUNT
-----  -----  -----
      0          0            3
      0          1  1 → incremented because 1000<=processed rows <1e6
      0          2            0

```

```
SQL> exec :ln2 := 1000000;
SQL> select count(1) from t_acs where n2 = :ln2;
      COUNT(1)
-----
      1099049

SQL> select
      child_number
      ,bucket_id
      ,count
  from
    v$sql_cs_histogram
 where  sql_id = 'f2pmwazy1rnfd' ;
      CHILD_NUMBER   BUCKET_ID       COUNT
-----  -----  -----
          0           0            3
          0           1            1
          0           2            1 → incremented because processed rows >=1e6
```

```

SQL> alter system flush shared_pool;

SQL> exec :ln2 := 100

SQL> select count(1) from t_acs where n2 = :ln2;

COUNT(1)
-----
100

SQL> -- repeat 4 times
SQL> select
      child_number
      ,bucket_id
      ,count
  from
      v$sql_cs_histogram
  where  sql_id = 'f2pmwazy1rnfd' ;

CHILD_NUMBER   BUCKET_ID       COUNT
-----  -----  -----
0             0           5 → incremented 5 times
0             1           0
0             2           0

SQL> exec :ln2 := 10000

SQL> select count(1) from t_acs where n2 = :ln2;

COUNT(1)
-----
100000

```

```
SQL> -- repeat 4 times
```

```
SQL> select
      child_number
      ,bucket_id
      ,count
  from
      v$sql_cs_histogram
 where  sql_id = 'f2pmwazylrnfd' ;
```

| CHILD_NUMBER | BUCKET_ID | COUNT |
|--------------|-----------|-------------------------|
| 0 | 0 | 5 |
| 0 | 1 | 5 → incremented 5 times |
| 0 | 2 | 0 |

```
SQL> select count(1) from t_acs where n2 = :ln2;
```

```
COUNT(1)
```

```
-----  
100000
```

```
SQL> select
      child_number
    , bucket_id
    , count
  from
    v$sql_cs_histogram
 where  sql_id = 'f2pmwazy1rnfd' ;
```

| CHILD_NUMBER | BUCKET_ID | COUNT |
|--------------|-----------|------------------------------|
| 1 | 0 | 0 |
| 1 | 1 | 1 |
| 1 | 2 | 0 |
| 0 | 0 | 5 → COUNT of BUCKET_ID 0 = 5 |
| 0 | 1 | 5 → COUNT of BUCKET_ID 1 = 5 |
| 0 | 2 | 0 |

```
SQL> select
      sql_id
    , child_number
    , is_bind_sensitive
    , is_bind_aware
    , is_shareable
  from
    v$sql
 where
  sql_id = 'f2pmwazy1rnfd';
```

| SQL_ID | CHILD_NUMBER | IS_BIND_SENSITIVE | IS_BIND_AWARE | IS_SHAREABLE |
|----------------------|--------------|-------------------|---------------|--------------|
| f2pmwazy1rnfd | 0 | Y | N | N |
| f2pmwazy1rnfd | 1 | Y | Y | Y |

```

SQL> alter system flush shared_pool;

SQL> exec :ln2 := 100

SQL> select count(1) from t_acs where n2 = :ln2;

COUNT(1)
-----
100
SQL> repeat this 8 times

SQL> select
      child_number
    , bucket_id
    , count
  from
    v$sql_cs_histogram
  where sql_id = 'f2pmwazy1rnfd' ;

CHILD_NUMBER  BUCKET_ID      COUNT
-----  -----
0            0          9 → incremented 9 times = 9 executions
0            1          0
0            2          0

-- change the bind variable value
SQL> exec :ln2 := 1000000

SQL> select count(1) from t_acs where n2 = :ln2;

COUNT(1)
-----
1099049

SQL> repeat this query 2 times

```

```
SQL> select
      child_number
      ,bucket_id
      ,count
  from
    v$sql_cs_histogram
 where  sql_id = 'f2pmwazy1rnfd' ;
```

| CHILD_NUMBER | BUCKET_ID | COUNT |
|--------------|-----------|--|
| 0 | 0 | 9 → ceil(9/3) = 3 = COUNT of BUCKET_ID 2 |
| 0 | 1 | 0 |
| 0 | 2 | 3 → incremented 3 times = 3 executions |

```

SQL> select count(1) from t_acs where n2 = :ln2;
          COUNT(1)
-----
          1099049
SQL> select
      child_number
      ,bucket_id
      ,count
  from
    v$sql_cs_histogram
  where  sql_id = 'f2pmwazy1rnfd' ;

CHILD_NUMBER  BUCKET_ID      COUNT
-----  -----
        1          0          0
        1          1          0 → bucket 1 not involved
1          2          1
        0          0          9 → bucket 0 incremented 9 times: ceil(9/3)= 3
        0          1          0
        0          2          3 → bucket 2 incremented 3 times: 3 = ceil (9/3)

```

```
SQL> select
      child_number
      ,bucket_id
      ,count
  from
      v$sql_cs_histogram
 where   sql_id = 'f2pmwazy1rnfd';
```

| CHILD_NUMBER | BUCKET_ID | COUNT |
|--------------|-----------|-------|
| 0 | 0 | 3 |
| 0 | 1 | 1 |
| 0 | 2 | 1 |

```
SQL> exec :ln2 := 100
```

```
SQL> select count(1) from t_acs where n2 = :ln2;
```

| CHILD_NUMBER | BUCKET_ID | COUNT |
|--------------|-----------|-------|
|--------------|-----------|-------|

| | | |
|---|---|--|
| 1 | 0 | 1 → when the 6th execution is for :ln2=100 |
| 1 | 1 | 0 |
| 1 | 2 | 0 |
| 0 | 0 | 3 |
| 0 | 1 | 1 |
| 0 | 2 | 1 |

```
SQL> exec :ln2 := 10000
```

```
SQL> select count(1) from t_acs where n2 = :ln2;
```

| CHILD_NUMBER | BUCKET_ID | COUNT |
|--------------|-----------|-------|
|--------------|-----------|-------|

| | | |
|---|---|--|
| 1 | 0 | 0 |
| 1 | 1 | 1 → when the 6th execution is for :ln2 = 10000 |
| 1 | 2 | 0 |
| 0 | 0 | 3 |
| 0 | 1 | 1 |
| 0 | 2 | 1 |

```
SQL> exec :ln2 := 1000000
```

```
SQL> select count(1) from t_acs where n2 = :ln2;
```

| CHILD_NUMBER | BUCKET_ID | COUNT |
|--------------|-----------|-------|
|--------------|-----------|-------|

| | | |
|---|---|--|
| 1 | 0 | 0 |
| 1 | 1 | 0 |
| 1 | 2 | 1 → when the 6th execution is for:ln2 =1000000 |
| 0 | 0 | 3 |
| 0 | 1 | 1 |
| 0 | 2 | 1 |

| CHILD_NUMBER | BUCKET_ID | COUNT |
|--------------|-----------|-------|
| 0 | 0 | 11 |
| 0 | 1 | 4 |
| 0 | 2 | 3 |

```
SQL> exec :ln2 := 1000000
SQL> select count(1) from t_acs where n2 = :ln2;
SQL> select
      child_number
    , bucket_id
    , count
  from
    v$sql_cs_histogram
  where  sql_id = 'f2pmwazy1rnfd' ;

CHILD_NUMBER  BUCKET_ID      COUNT
-----  -----
      1          0            0
      1          1            0
1          2            1 → execution done at this bucket
      0          0           11
      0          1            4
      0          2            3
```

```
SQL> select fv_will_cs_be_bind_aware(3,1,1) IS_BIND_AWARE from dual;
```

```
IS_BIND_AWARE
```

```
-----  
Y
```

```
SQL> select fv_will_cs_be_bind_aware(11,4,3) IS_BIND_AWARE from dual;
```

```
IS_BIND_AWARE
```

```
-----  
Y
```

```
SQL> alter system flush shared_pool;
SQL> exec :ln2 := 100
SQL> select count(1) from t_acs where n2 = :ln2;

SQL> exec :ln2 := 10000
SQL> select count(1) from t_acs where n2 = :ln2;

SQL> select
      child_number
    , predicate
    , low
    , high
  from
    v$sql_cs_selectivity
 where
   sql_id = 'f2pmwazylrnfd';

no rows selected
```

```

SQL> print :ln2

LN2
-----
10000
SQL> select count(1) from t_acs where n2 = :ln2;

SQL> select
      child_number
    , predicate
    , low
    , high
  from
    v$sql_cs_selectivity
 where
   sql_id = 'f2pmwazy1rnfd';

```

| CHILD_NUMBER | PREDICATE | LOW | HIGH |
|--------------|-----------|----------|----------|
| 1 | =LN2 | 0.074579 | 0.091152 |

```

SQL> exec :ln2 := 100
SQL> select count(1) from t_acs where n2 = :ln2;

      COUNT(1)
-----
      100

SQL> select
      child_number
      ,predicate
      ,low
      ,high
  from
    v$sql_cs_selectivity
 where
   sql_id = 'f2pmwazy1rnfd';

CHILD_NUMBER PREDICATE          LOW        HIGH
-----  -----
  2 =LN2                      0.000361  0.091152
  1 =LN2                      0.074579  0.091152

```

```
SQL> exec :ln2 := 1000000
```

```
SQL> select count(1) from t_acs where n2 = :ln2;
```

```
 COUNT(1)
```

```
-----  
 1099049
```

```
SQL> select
```

```
    child_number  
 ,predicate  
 ,low  
 ,high  
 from  
   v$sql_cs_selectivity  
 where  
   sql_id = 'f2pmwazy1rnfd';
```

| CHILD_NUMBER | PREDICATE | LOW | HIGH |
|--------------|-----------|-----------------|-----------------|
| 3 | =LN2 | 0.823886 | 1.006972 |
| 2 | =LN2 | 0.000361 | 0.091152 |
| 1 | =LN2 | 0.074579 | 0.091152 |

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

BANNER

```
-----  
Oracle Database 11g Enterprise Edition Release 11.2.0.3.0 - 64bit Production  
PL/SQL Release 11.2.0.3.0 - Production  
CORE    11.2.0.3.0      Production  
TNS for Linux: Version 11.2.0.3.0 - Production  
NLSRTL Version 11.2.0.3.0 - Production
```

```
SQL> select  
      sql_id  
 ,count(1)  
  from  
    v$sql  
 where executions < 2  
 group by sql_id  
 having count(1) > 10  
 order by 2 desc;
```

| SQL_ID | COUNT(1) |
|----------------|----------|
| 7zwq7z1nj7vga | 44217 |
| 39ax31acw29z6 | 75 |
| 0v3dvmc22qnam | 29 |
| 412j04p609svj | 25 |
| 5s34t44u10q4g | 17 |
| c8gnrhxma4tas | 16 |
| g8m7zdgak6pmm | 16 |
| gjm43un5cy843 | 16 |
| 6wdut577suw74s | 15 |
| 23nad9x295gkf | 14 |
| 848dyu9288c3h | 14 |
| 6wm3n4d7bnddg | 14 |
| 2am60vd2kw8ux | 14 |
| 0ctk7jpx5chm | 14 |
| 3x13xht9dh5c3 | 14 |
| gdn3ysuyssf82 | 13 |
| gg17hgzmmttbu | 13 |
| 53ps5ua5ms44q | 13 |
| 1z2rnd9bubah9 | 13 |

```
SQL> @nonshared 7zwq7z1nj7vga
```

```
Show why existing SQL child cursors were not reused (V$SQL_SHARED_CURSOR)...
```

```
-----  
SQL_ID          : 7zwq7z1nj7vga  
ADDRESS         : 000000406DBB30F8  
CHILD_ADDRESS   : 00000042CE36F7E8  
CHILD_NUMBER    : 0  
BIND_EQUIV_FAILURE : Y  
REASON          : <ChildNode><ChildNumber>0</ChildNumber><ID>40</ID>  
                  <reason>Bindmismatch(33)</reason><size>2x4</size>  
                  <init_ranges_in_first_pass>0</init_ranges_in_first_pass>  
                  <selectivity>1097868685</selectivity>  
                  </ChildNode>  
  
-----  
SQL_ID          : 7zwq7z1nj7vga  
ADDRESS         : 000000406DBB30F8  
CHILD_ADDRESS   : 00000045B5C5E478  
CHILD_NUMBER    : 1  
BIND_EQUIV_FAILURE : Y  
REASON          : <ChildNode><ChildNumber>1</ChildNumber><ID>40</ID>  
                  <reason>Bindmismatch(33)</reason><size>2x4</size>  
                  <init_ranges_in_first_pass>0</init_ranges_in_first_pass>  
                  <selectivity>915662630</selectivity>  
                  </ChildNode>  
  
-----  
SQL_ID          : 7zwq7z1nj7vga  
ADDRESS         : 000000406DBB30F8  
CHILD_ADDRESS   : 00000038841E2868  
CHILD_NUMBER    : 2  
BIND_EQUIV_FAILURE : Y  
REASON          : <ChildNode><ChildNumber>2</ChildNumber><ID>40</ID>  
                  <reason>Bindmismatch(33)</reason><size>2x4</size>  
                  <init_ranges_in_first_pass>0</init_ranges_in_first_pass>  
                  <selectivity>163647208</selectivity>  
                  </ChildNode>
```

```
-----  
SQL_ID          : 7zwq7z1nj7vga  
ADDRESS         : 000000406DBB30F8  
CHILD_ADDRESS   : 00000038841E2708  
CHILD_NUMBER    : 3  
BIND_EQUIV_FAILURE  
REASON          :<ChildNode><ChildNumber>3</ChildNumber><ID>40</ID>  
                  <reason>Bindmismatch(33)</reason><size>2x4</size>  
                  <init_ranges_in_first_pass>0</init_ranges_in_first_pass>  
                  <selectivity>4075662961</selectivity>  
                  </ChildNode>  
.../  
-----  
SQL_ID          : 7zwq7z1nj7vga  
ADDRESS         : 000000406DBB30F8  
CHILD_ADDRESS   : 00000042DD3D7208  
CHILD_NUMBER    : 97  
BIND_EQUIV_FAILURE  
REASON          :<ChildNode><ChildNumber>97</ChildNumber><ID>40</ID>  
                  <reason>Bindmismatch(33)</reason><size>2x4</size>  
                  <init_ranges_in_first_pass>0</init_ranges_in_first_pass>  
                  <selectivity>3246589452</selectivity>  
                  </ChildNode>  
-----  
SQL_ID          : 7zwq7z1nj7vga  
ADDRESS         : 000000406DBB30F8  
CHILD_ADDRESS   : 00000042DD3D70A8  
CHILD_NUMBER    : 98  
BIND_EQUIV_FAILURE  
REASON          :<ChildNode><ChildNumber>98</ChildNumber><ID>40</ID>  
                  <reason>Bindmismatch(33)</reason><size>2x4</size>  
                  <init_ranges_in_first_pass>0</init_ranges_in_first_pass>  
                  <selectivity>3246589452</selectivity>  
                  </ChildNode>  
-----  
SQL_ID          : 7zwq7z1nj7vga  
ADDRESS         : 000000406DBB30F8  
CHILD_ADDRESS   : 00000045B5C5E5D8  
CHILD_NUMBER    : 99  
BIND_EQUIV_FAILURE  
REASON          :<ChildNode><ChildNumber>99</ChildNumber><ID>40</ID>  
                  <reason>Bindmismatch(33)</reason><size>2x4</size>  
                  <init_ranges_in_first_pass>0</init_ranges_in_first_pass>  
                  <selectivity>3246589452</selectivity>  
                  </ChildNode>
```

```
SQL> select
      count(1)
    from
      v$sql_shared_cursor
    where SQL_ID = '7zwq7z1nj7vga' ;
```

COUNT(1)

45125

```
SQL> select
      count(1)
    from
      v$sql_shared_cursor
    where sql_id = '7zwq7z1nj7vga'
      and BIND_EQUIV_FAILURE = 'Y' ;
```

COUNT(1)

45121

```
SQL> select
      count(1)
    from
      v$sql_cs_selectivity
   where
     sql_id = '7zwq7z1nj7vga';

COUNT(1)
-----
16,847,320
```

```
SQL> select * from table(dbms_xplan.display_awr('7zwq7z1nj7vga'));
```

Plan hash value: **587060143**

| Id | Operation | Name | Rows |
|----|-----------------------------|----------------|------|
| 0 | SELECT STATEMENT | | |
| 1 | TABLE ACCESS BY INDEX ROWID | MHO_TABLES_ACS | 159K |
| 2 | INDEX RANGE SCAN | TABL_ACS_INDX1 | 159K |

Note

- dynamic sampling used for this statement (level=4)

Plan hash value: **1114469665**

| Id | Operation | Name | Rows |
|----|-----------------------------|------------------|------|
| 0 | SELECT STATEMENT | | |
| 1 | TABLE ACCESS BY INDEX ROWID | MHO_TABLES_ACS | 1 |
| 2 | INDEX RANGE SCAN | TABL_ACS_INDX_PK | 1 |

Note

- dynamic sampling used for this statement (level=4)

Plan hash value: **2117864734**

| Id | Operation | Name | Rows |
|----|-----------------------------|----------------|------|
| 0 | SELECT STATEMENT | | |
| 1 | TABLE ACCESS BY INDEX ROWID | MHO_TABLES_ACS | 1 |
| 2 | INDEX RANGE SCAN | TABL_ACS_INDX2 | 1 |

Note

- - dynamic sampling used for this statement (level=4)

Plan hash value: **3054136074**

| Id | Operation | Name | Rows |
|----|-----------------------------|----------------|------|
| 0 | SELECT STATEMENT | | |
| 1 | TABLE ACCESS BY INDEX ROWID | MHO_TABLES_ACS | 159K |
| 2 | INDEX RANGE SCAN | TABL_ACS_INDX3 | 159K |

Note

- - dynamic sampling used for this statement (level=4)

```

SQL> select * from v$version where rownum=1;

BANNER                                     CON_ID
-----
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production      0

SQL> create table t1
  (col1 number
   ,col2 varchar2(50)
   ,flag varchar2(2));

SQL> insert into t1
  select
    rownum
   ,lpad('X',50,'X')
   ,case when rownum = 1
         then 'Y1'
         when rownum = 2
         then 'Y2'
         when mod(rownum,2) = 0
         then 'N1'
         else 'N2'
        end
  from dual
 connect by rownum <= 100000;

SQL> create index i1 on t1(flag);

-- gather statistics without histogram
SQL> exec dbms_stats.gather_table_stats(user,'t1', method_opt => 'for all columns size 1');

```

```
SQL> alter session set cursor_sharing=force;
```

```
SQL> begin
  exec dbms_stats.gather_table_stats(user
                                      , 't1'
                                      , method_opt => 'for columns flag size skewonly');
end;
```

```
SQL> select count(*), max(col2) from t1 where flag = 'N1';

  COUNT(*)  MAX(COL2)
-----
 49999 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

```
SQL_ID  6fbvysnhkvugw, child number 0
-----
select count(*), max(col2) from t1 where flag = :"SYS_B_0"
```

```
Plan hash value: 3724264953
```

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|----|---------------------|------|-------|-------|-------------|----------|
| 0 | SELECT STATEMENT | | | | 273 (100) | |
| 1 | SORT AGGREGATE | | 1 | 54 | | |
| * | 2 TABLE ACCESS FULL | T1 | 48640 | 2565K | 273 (1) | 00:00:01 |

```
Predicate Information (identified by operation id):
```

```
2 - filter("FLAG"=:SYS_B_0)
```

```
SQL> select count(*), max(col2) from t1 where flag = 'Y1';
```

```
COUNT(*) MAX(COL2)
```

```
-----  
1 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

```
SQL_ID 6fbvysnhkvugw, child number 0
```

```
select count(*), max(col2) from t1 where flag = :"SYS_B_0"
```

```
Plan hash value: 3724264953
```

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|----|---------------------|------|-------|-------|-------------|----------|
| 0 | SELECT STATEMENT | | | | 273 (100) | |
| 1 | SORT AGGREGATE | | 1 | 54 | | |
| * | 2 TABLE ACCESS FULL | T1 | 48640 | 2565K | 273 (1) | 00:00:01 |

```
Predicate Information (identified by operation id):
```

```
2 - filter("FLAG"=:SYS_B_0)
```

```
SQL> show parameter baseline
```

| NAME | TYPE | VALUE |
|--------------------------------------|---------|-------|
| optimizer_capture_sql_plan_baselines | boolean | FALSE |
| optimizer_use_sql_plan_baselines | boolean | TRUE |

```
SQL> alter session set optimizer_capture_sql_plan_baselines=TRUE;

SQL> Select count(*), max(col2) From t1 where flag = 'Y1';

COUNT(*) MAX(COL2)
-----
1 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

SQL> /

COUNT(*) MAX(COL2)
-----
1 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

SQL> alter session set optimizer_capture_sql_plan_baselines=FALSE;
```

```
SQL> select
      to_char(signature) signature
    ,plan_name
    ,origin
    ,enabled
    ,accepted
  from
    dba_sql_plan_baselines
 where
   sql_text like '%Select count(*)%';
```

| SIGNATURE | PLAN_NAME | ORIGIN | ENA ACC |
|----------------------|--------------------------------|--------------|---------|
| 15340826253708983785 | SQL_PLAN_d9tch6banyzg97823646b | AUTO-CAPTURE | YES YES |

```
SQL> select
      sql_id
    ,child_number
    ,sql_text
  from
    v$sql a
 where
   a.is_shareable = 'Y'
 and exists (select
              null
            from
              dba_sql_plan_baselines b
            where
              b.signature = a.exact_matching_signature
            );

```

| SQL_ID | CHILD_N | SQL_TEXT |
|---------------|---------|--|
| 98ub2xj0nt01g | 0 | Select count(*), max(col2) From t1 where flag = :"SYS_B_0" |
| 6fbvysnhkvugw | 0 | select count(*), max(col2) from t1 where flag = :"SYS_B_0" |

```
v$sql.exact_matching_signature = dba_sql_plan_baselines.signature
```

```
SQL> select * from table(dbms_xplan.display_sql_plan_baseline(plan_name =>
'SQL_PLAN_d9tch6banyzg97823646b'));
```

```
-----  
SQL handle: SQL_d4e59032d54f7de9
```

```
SQL text: Select count(*), max(col2) From t1 where flag = :"SYS_B_0"
```

```
-----  
Plan name: SQL_PLAN_d9tch6banyzg97823646b          Plan id: 2015585387  
Enabled: YES      Fixed: NO      Accepted: YES      Origin: AUTO-CAPTURE  
Plan rows: From dictionary
```

```
-----  
Plan hash value: 497086120
```

| Id Operation | | Name | Rows | Bytes | Cost | (%CPU) |
|----------------|-------------------------------------|------|------|-------|------|--------|
| 0 | SELECT STATEMENT | | 1 | 54 | 2 | (0) |
| 1 | SORT AGGREGATE | | 1 | 54 | | |
| 2 | TABLE ACCESS BY INDEX ROWID BATCHED | T1 | 1 | 54 | 2 | (0) |
| * 3 | INDEX RANGE SCAN | I1 | 1 | | 1 | (0) |

```
Predicate Information (identified by operation id):
```

```
-----  
3 - access("FLAG"=:SYS_B_0)
```

```
SQL> SELECT count(*), max(col2) FROM t1 where flag = 'N2';
```

```
COUNT(*) MAX(COL2)
```

```
-----  
49999 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

| Id Operation | Name | Rows | Bytes | Cost (%CPU) |
|---|------|-------|-------|-------------|
| 0 SELECT STATEMENT | | | | 1002 (100) |
| 1 SORT AGGREGATE | | 1 | 54 | |
| 2 TABLE ACCESS BY INDEX ROWID BATCHED | T1 | 50887 | 2683K | 1002 (1) |
| * 3 INDEX RANGE SCAN | I1 | 50887 | | 100 (0) |

```
Predicate Information (identified by operation id):
```

```
-----  
3 - access("FLAG"=:SYS_B_0)
```

Note

```
- SQL plan baseline SQL_PLAN_d9tch6banyzg97823646b used for this statement
```

```
SQL> SeLeCt count(*), max(col2) FROM t1 where flag = 'Y2';
```

```
COUNT(*) MAX(COL2)
```

```
-----  
1 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

| Id Operation | Name | Rows | Bytes | Cost (%CPU) |
|---|------|------|-------|-------------|
| 0 SELECT STATEMENT | | | | 2 (100) |
| 1 SORT AGGREGATE | | 1 | 54 | |
| 2 TABLE ACCESS BY INDEX ROWID BATCHED | T1 | 1 | 54 | 2 (0) |
| * 3 INDEX RANGE SCAN | I1 | 1 | | 1 (0) |

```
Predicate Information (identified by operation id):
```

```
-----  
3 - access("FLAG"=:SYS_B_0)
```

Note

```
- SQL plan baseline SQL_PLAN_d9tch6banyzg97823646b used for this statement
```

```
SQL> declare
      rs pls_integer;
begin
  -- disabling a SQLplan from a SQL Management Base
  rs := dbms_spm.alter_sql_plan_baseline
    (plan_name      => 'SQL_PLAN_d9tch6banyzg97823646b'
     ,attribute_name => 'ENABLED'
     ,attribute_value => 'NO'
    );
  -- dropping a SQL plan from a SQL Management Base
  rs := dbms_spm.drop_sql_plan_baseline
    (plan_name      => 'SQL_PLAN_d9tch6banyzg97823646b'
     );
end;
/
PL/SQL procedure successfully completed.
```

```
SQL> SELECT count(*), max(col2) FROM t1 where flag = 'N2';
```

```
COUNT(*) MAX(COL2)
```

```
-----  
49999 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

```
SQL_ID 9cmb2wv5gkq2x, child number 0
```

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) |
|----|--------------------------|------|-------|-------|-------------|
| 0 | SELECT STATEMENT | | | | 273 (100) |
| 1 | SORT AGGREGATE | | 1 | 54 | |
| * | TABLE ACCESS FULL | T1 | 50887 | 2683K | 273 (1) |

```
Predicate Information (identified by operation id):
```

```
2 - filter("FLAG"=:SYS_B_0)
```

```
SQL> declare
      rs pls_integer;
begin
  rs := dbms_spm.load_plans_from_cursor_cache('9cmb2wv5gkq2x');
end;
/
PL/SQL procedure successfully completed.
```

```
SQL> SELECT count(*), MAX(col2) FROM t1 where flag = 'Y2';
```

```
COUNT(*) MAX(COL2)
```

```
-----  
1 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|----|-------------------|------|------|-------|-------------|----------|
| 0 | SELECT STATEMENT | | | | 273 (100) | |
| 1 | SORT AGGREGATE | | 1 | 54 | | |
| * | TABLE ACCESS FULL | T1 | 1 | 54 | 273 (1) | 00:00:01 |

```
Predicate Information (identified by operation id):
```

```
-----  
2 - filter("FLAG"=:SYS_B_0)
```

```
Note
```

```
-----  
- SQL plan baseline SQL_PLAN_d9tch6banyzg9616acf47 used for this statement
```

```
SQL> SELECT count(*), max(col2) FROM t1 where flag = 'N2';
```

```
SQL> SELECT /*+ index (t1) */ count(*), max(col2) FROM t1 where flag = 'N2';
```

```
SQL> SELECT /*+ index (t1) */ count(*), max(col2) FROM t1 where flag = 'N2';
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

```
SQL_ID  fxk36bs5436us, child number 0 Plan hash value: 497086120
```

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) |
|----|-------------------------------------|------|-------|-------|-------------|
| 0 | SELECT STATEMENT | | | | 1002 (100) |
| 1 | SORT AGGREGATE | | 1 | 54 | |
| 2 | TABLE ACCESS BY INDEX ROWID BATCHED | T1 | 50887 | 2683K | 1002 (1) |
| * | INDEX RANGE SCAN | I1 | 50887 | | 100 (0) |

```
Predicate Information (identified by operation id):
```

```
3 - access("FLAG"=:SYS_B_0)
```

```
SQL> declare
      rs pls_integer;
begin
  rs := dbms_spm.load_plans_from_cursor_cache('fxk36bs5436us');
end;
/
```

```

SQL> select
      sql_handle
    , plan_name
    , substr(sql_text,1,30) sql_text
    , accepted
  from
    dba_sql_plan_baselines
 where
   enabled  = 'YES'
 ;

```

| SQL_HANDLE | PLAN_NAME | SQL_TEXT | ACC |
|----------------------|--------------------------------|--------------------------------|-----|
| SQL_d422e56d9adb0b66 | SQL_PLAN_d88r5dqddq2v67823646b | SELECT /*+ index (t1) */ count | YES |
| SQL_d4e59032d54f7de9 | SQL_PLAN_d9tch6banyzg9616acf47 | Select count(*), max(col2) Fro | YES |
| SQL_d4e59032d54f7de9 | SQL_PLAN_d9tch6banyzg97823646b | Select count(*), max(col2) Fro | NO |

```
SQL> SELECT count(*), max(col2) FROM t1 where flag = 'N2';
```

```
-----  
SQL_ID 9cmb2wv5gkq2x, child number 0  
-----
```

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|----|-------------------|------|-------|-------|-------------|----------|
| 0 | SELECT STATEMENT | | | | 273 (100) | |
| 1 | SORT AGGREGATE | | 1 | 54 | | |
| * | TABLE ACCESS FULL | T1 | 50887 | 2683K | 273 (1) | 00:00:01 |

```
Predicate Information (identified by operation id):  
-----
```

```
2 - filter("FLAG"=:SYS_B_0)
```

```
Note  
-----
```

```
- SQL plan baseline SQL_PLAN_d9tch6banyzg9616acf47 used for this statement
```

```
SQL> declare
  ln_ps number;
begin
  ln_ps := dbms_spm.alter_sql_plan_baseline
    (sql_handle      => 'SQL_d4e59032d54f7de9'
     ,plan_name       => 'SQL_PLAN_d9tch6banyzg9616acf47'
     ,attribute_name  => 'enabled'
     ,attribute_value => 'NO');
end;
/

```

PL/SQL procedure successfully completed.

```
SQL> declare
  ln_ps pls_integer;
begin
  ln_ps := dbms_spm.load_plans_from_cursor_cache
(sql_id  => 'fxk36bs5436us'    -- sql_id of the index range scan plan
,plan_hash_value => 497086120 -- plan hash value of the index range scan plan
,sql_handle=> 'SQL_d4e59032d54f7de9' -- sql handle of the full table scan plan
);
end;
/
PL/SQL procedure successfully completed.
```

```
SQL> SELECT count(*), max(col2) FROM t1 where flag = 'N2';
```

```
COUNT(*) MAX(COL2)
```

```
-----  
49999 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

```
SQL_ID 9cmb2wv5gkq2x, child number 0
```

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) |
|----|-------------------------------------|------|-------|-------|-------------|
| 0 | SELECT STATEMENT | | | | 1002 (100) |
| 1 | SORT AGGREGATE | | 1 | 54 | |
| 2 | TABLE ACCESS BY INDEX ROWID BATCHED | T1 | 50887 | 2683K | 1002 (1) |
| * | INDEX RANGE SCAN | I1 | 50887 | | 100 (0) |

```
Predicate Information (identified by operation id):
```

```
3 - access("FLAG"=:SYS_B_0)
```

```
Note
```

```
- SQL plan baseline SQL_PLAN_d9tch6banyzg97823646b used for this statement
```

```
SQL> declare
      ln_ps pls_integer;
begin
  for x in (select plan_name from dba_sql_plan_baselines)
  loop
    ln_ps := dbms_spm.drop_sql_plan_baseline(plan_name => x.plan_name);
  end loop;
exception
  when others then
    null; -- don't use this in production
end;
/
PL/SQL procedure successfully completed.
```

```
SQL> select count(1) from dba_sql_plan_baselines;
```

| COUNT(1) |
|----------|
| ----- |
| 0 |

```
SQL> start coe_load_sql_baseline.sql 9cmb2wv5gkq2x fxk36bs5436us 497086120
coe_load_sql_baseline completed.
```

```
SQL> SELECT count(*), max(col2) FROM t1 where flag = 'N2';
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

```
SQL_ID  9cmb2wv5gkq2x, child number 0
```

```
-----  
SELECT count(*), max(col2) FROM t1 where flag = :"SYS_B_0"
```

```
Plan hash value: 497086120
```

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) |
|----|-------------------------------------|------|-------|-------|-------------|
| 0 | SELECT STATEMENT | | | | 1002 (100) |
| 1 | SORT AGGREGATE | | 1 | 54 | |
| 2 | TABLE ACCESS BY INDEX ROWID BATCHED | T1 | 50887 | 2683K | 1002 (1) |
| * | INDEX RANGE SCAN | I1 | 50887 | | 100 (0) |

```
Predicate Information (identified by operation id):
```

```
-----  
 3 - access ("FLAG"=:SYS_B_0)
```

```
Note
```

```
-----  
- SQL plan baseline SQL_PLAN_d9tch6banyzg97823646b used for this statement
```

```
SQL> select
      a.plan_name
    ,b.plan_id
    ,a.enabled
    ,a.accepted
  from
    dba_sql_plan_baselines a
  ,sys.sqlobj$ b
 where
   a.signature = b.signature
 and a.plan_name = b.name;
```

| PLAN_NAME | PLAN_ID | ENA | ACC |
|--------------------------------|-------------------|-----|-----------------------------|
| SQL_PLAN_d9tch6banyzg9616acf47 | 1634389831 | YES | YES → full table scan plan |
| SQL_PLAN_d9tch6banyzg97823646b | 2015585387 | YES | YES → index range scan plan |

```
SQL> alter session set events '10053 trace name context forever, level 1';

SQL> SELECT count(*), max(col2) FROM t1 where flag = 'Y1';

COUNT(*)  MAX(COL2)
-----
1 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

1 row selected.

SQL> alter session set events '10053 trace name context off';
```

```
*****  
---- Current SQL Statement for this session (sql_id=4sdth4kka4ykw) ----  
SELECT count(*), max(col2) FROM t1 where flaG = :"SYS_B_0"  
*****
```

SPM: cost-based plan found in the plan baseline, planId = 2015585387
SPM: cost-based plan **successfully matched**, planId = 2015585387

=====

Plan Table

=====

| Id | Operation | Name | Rows | Bytes | Cost | Time |
|----|-------------------------------------|------|------|-------|------|----------|
| 0 | SELECT STATEMENT | | | | 2 | |
| 1 | SORT AGGREGATE | | 1 | 54 | | |
| 2 | TABLE ACCESS BY INDEX ROWID BATCHED | T1 | 1 | 54 | 2 | 00:00:01 |
| 3 | INDEX RANGE SCAN | I1 | 1 | | 1 | 00:00:01 |

Predicate Information:

3 - access("FLAG"=:SYS\_B\_0)

Content of other\_xml column

=====

db\_version : 12.1.0.1
parse\_schema : C##MHOURI
plan\_hash : 497086120
plan\_hash\_2 : 2015585387

```

SQL> SELECT
      p.sql_id
    ,p.plan_hash_value
    ,p.child_number
    ,t.phv2
  FROM
    v$sql_plan p
   ,xmltable('for $i in /other_xml/info
              where $i/@type eq "plan_hash_2"
              return $i'
             passing xmltype(p.other_xml)
             columns phv2 number path '/') t
 WHERE p.sql_id = '4sdth4kka4ykw'
 AND   p.other_xml is not null;

```

| SQL_ID | PLAN_HASH_VALUE | CHILD_NUMBER | PHV2 |
|----------------------|------------------|--------------|-------------------|
| 4sdth4kka4ykw | 497086120 | 0 | 2015585387 |

```
*****
----- Current SQL Statement for this session (sql_id=gvrgd85zjjdps) -----
select count(*), max(col2) FROM t1 where flag = :"SYS_B_0"
*****
SPM: cost-based plan found in the plan baseline, planId = 1634389831
SPM: cost-based plan successfully matched, planId = 1634389831
```

```
=====
```

Plan Table

```
=====
```

| Id | Operation | Name | Rows | Bytes | Cost | Time |
|----|-------------------|------|------|-------|------|----------|
| 0 | SELECT STATEMENT | | | | 273 | |
| 1 | SORT AGGREGATE | | 1 | 54 | | |
| 2 | TABLE ACCESS FULL | T1 | 50K | 2683K | 273 | 00:00:04 |

Predicate Information:

```
2 - filter("FLAG"=:SYS_B_0)
```

Content of other\_xml column

```
=====
```

```
db_version      : 12.1.0.1
parse_schema    : C##MHOURI
plan_hash       : 3724264953
plan_hash_2     : 1634389831
```

```
SQL> alter session set optimizer_features_enable='11.2.0.3';
```

```
SQL> select plan_name from dba_sql_plan_baselines where accepted = 'YES';
PLAN_NAME
-----
SQL_PLAN_d9tch6banyzg9616acf47
SQL_PLAN_d9tch6banyzg97823646b
```

```
SQL> select plan_name from dba_sql_plan_baselines where accepted = 'NO';
```

```
no rows selected
```

```
SQL> SELECT count(*), max(col2) FROM t1 where flag = 'Y1';
```

```
COUNT(*) MAX(COL2)
```

```
-----  
1 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
SQL> select plan_name from dba_sql_plan_baselines where accepted = 'NO';
```

```
PLAN_NAME
```

```
-----  
SQL_PLAN_d9tch6banyzg98576eb1f
```

```
SQL>select * from table(dbms_xplan.display_sql_plan_baseline(plan_name =>
'SQL_PLAN_d9tch6banyzg98576eb1f'));
```

```
-----  
SQL handle: SQL_d4e59032d54f7de9  
SQL text: SELECT count(*), max(col2) FROM t1 where flag = :"SYS_B_0"  
-----
```

```
-----  
Plan name: SQL_PLAN_d9tch6banyzg98576eb1f Plan id: 2239163167  
Enabled: YES Fixed: NO Accepted: NO Origin: AUTO-CAPTURE  
Plan rows: From dictionary  
-----
```

```
Plan hash value: 3625400295
```

| Id Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|--|------|------|-------|-------------|----------|
| 0 SELECT STATEMENT | | 1 | 54 | 2 (0) | 00:00:01 |
| 1 SORT AGGREGATE | | 1 | 54 | | |
| 2 TABLE ACCESS BY INDEX ROWID | T1 | 1 | 54 | 2 (0) | 00:00:01 |
| * 3 INDEX RANGE SCAN | I1 | 1 | | 1 (0) | 00:00:01 |

```
Predicate Information (identified by operation id):
```

```
-----  
3 - access("FLAG"=:SYS_B_0)
```

```

SQL> SELECT
      p.sql_id
    ,p.plan_hash_value
    ,p.child_number
    ,t.phv2
  FROM
    v$sql_plan p
   XMLTABLE('for $i in /other_xml/info
             where $i/@type eq "plan_hash_2"
             return $i'
             passing XMLTYPE(p.other_xml)
             columns phv2 number path '/') t
 WHERE p.sql_id = '4sdth4kka4ykw'
   AND p.other_xml is not null;

SQL_ID          PLAN_HASH_VALUE CHILD_NUMBER        PHV2
-----          -----          -----          -----
4sdth4kka4ykw      497086120          0  2015585387 → 12c ofe execution plan

```

```

SQL> alter session set optimizer_use_sql_plan_baselines = FALSE;
SQL>SELECT count(*), max(col2) FROM t1 where flag = 'Y1';
   COUNT(*) MAX(COL2)
-----
   1 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

SQL> SELECT
      p.sql_id
     ,p.plan_hash_value
     ,p.child_number
     ,t.phv2
  FROM
    v$sql_plan p
   XMLTYPE('for $i in /other_xml/info
            where $i/@type eq "plan_hash_2"
                  return $i'
            passing XMLTYPE(p.other_xml)
            columns phv2 number path '/') t
 WHERE p.sql_id = '4sdth4kka4ykw'
 AND   p.other_xml is not null;

SQL_ID      PLAN_HASH_VALUE CHILD_NUMBER      PHV2
-----      -----
4sdth4kka4ykw      3625400295          0  2239163167 → new 11g ofe execution plan
4sdth4kka4ykw      497086120           0  2015585387 → 12c ofe execution plan

```

```
*****  
---- Current SQL Statement for this session (sql_id=4sdth4kka4ykw) ----  
SELECT count(*), max(col2) FROM t1 where flaG = :"SYS_B_0"  
*****
```

```
SPM: setup to add new plan to existing plan baseline
, sig = 15340826253708983785, planId = 2239163167
SPM: sql stmt=SELECT count(*), max(col2) FROM t1 where flag = :"SYS_B_0"
SPM: planId's of plan baseline are: 1634389831 2015585387
```

SPM: using qksan to reproduce, cost and select accepted plan, sig = 15340826253708983785
SPM: plan reproducibility round 1 (plan outline + session OFE)

SPM: using qksan to reproduce accepted plan, planId = 1634389831
SPM: plan reproducibility - **session OFE = 11020003, hinted OFE = 12010001**

Final query after transformations:\*\*\*\*\* UNPARSED QUERY IS \*\*\*\*\*
SELECT /\*+ FULL ("T1") \*/ COUNT(\*) "COUNT(\*)",MAX("T1"."COL2")
"MAX(COL2)" FROM "C##MHOURI"."T1" "T1" WHERE "T1"."FLAG"=:B1

```
Outline Data:  
/*+  
BEGIN_OUTLINE_DATA  
IGNORE_OPTIM_EMBEDDED_HINTS  
OPTIMIZER_FEATURES_ENABLE('11.2.0.3') --> this is the altered session OFE  
DB_VERSION('12.1.0.1')  
ALL_ROWS  
OUTLINE_LEAF(@"SEL$1")  
FULL(@"SEL$1" "T1@"@"SEL$1") --> this is the SPM full scan hint  
END_OUTLINE_DATA  
*/
```

SPM: planId in plan baseline = 1634389831, planId of reproduced plan = 1634389831
SPM: **best cost so far = 272.81**, current accepted plan cost = 272.81

SPM: plan reproducibility round 1 (plan outline + session OFE)
SPM: using qksan to reproduce accepted plan, planId = 2015585387

Stmt: \*\*\*\*\* UNPARSED QUERY IS \*\*\*\*\*
SELECT /\*+ BATCH\_TABLE\_ACCESS\_BY\_ROWID ("T1")
INDEX\_RS\_ASC ("T1" "I1") \*/ COUNT(\*) "COUNT(\*)",MAX("T1"."COL2")
"MAX(COL2)"
FROM "C##MHOURI"."T1" "T1" WHERE "T1"."FLAG"=:B1

Outline Data:

```
/*+
BEGIN_OUTLINE_DATA
  IGNORE_OPTIM_EMBEDDED_HINTS
  OPTIMIZER_FEATURES_ENABLE('11.2.0.3') --> this is the altered session OFE
  DB_VERSION('12.1.0.1')
  ALL_ROWS
  OUTLINE_LEAF(@"SEL$1")
  INDEX_RS_ASC(@"SEL$1" "T1@""SEL$1" ("T1"."FLAG")) --> this the SPM hint
  BATCH_TABLE_ACCESS_BY_ROWID(@"SEL$1" "T1@""SEL$1") --> this the SPM hint
END_OUTLINE_DATA
*/
```

SPM: planId in plan baseline = 2015585387, planId of reproduced plan = **2015585387**
SPM: best cost so far = 2.00, current accepted plan cost = 2.00

```
SQL> select
      extractValue(value(i),'.') outline_hints
    from
      sys.sqlobj$data sqd,
      table(xmlsequence(extract(xmltype(sqd.comp_data) , '/outline_data/hint')))i
   where
     signature = '15285764617405881585';

OUTLINE_HINTS
-----
/*+
BEGIN_OUTLINE_DATA
IGNORE_OPTIM_EMBEDDED_HINTS
OPTIMIZER_FEATURES_ENABLE('12.1.0.1') --> this is OFE at the SPM capture time
DB_VERSION('12.1.0.1')
ALL_ROWS
OUTLINE_LEAF(@"SEL$1")
INDEX_RS_ASC(@"SEL$1" "T1"@"SEL$1" ("T1"."FLAG"))
BATCH_TABLE_ACCESS_BY_ROWID(@"SEL$1" "T1"@"SEL$1")
END_OUTLINE_DATA
*/
```

```
SPM: statement found in SMB
SPM: re-parsing to generate selected accepted plan, planId = 2015585387
SPM: re-parsed to use selected accepted plan, planId = 2015585387
```

Outline Data:

```
/*+
BEGIN_OUTLINE_DATA
  IGNORE_OPTIM_EMBEDDED_HINTS
  OPTIMIZER_FEATURES_ENABLE('12.1.0.1')
  DB_VERSION('12.1.0.1')
  ALL_ROWS
  OUTLINE_LEAF(@"SEL$1")
  INDEX_RS_ASC(@"SEL$1" "T1"@"SEL$1" ("T1"."FLAG"))
  BATCH_TABLE_ACCESS_BY_ROWID(@"SEL$1" "T1"@"SEL$1")
END_OUTLINE_DATA
*/
```

```
SPM: PHV2 on user parse = 2015585387, PHV2 on recursive parse = 2239163167
SPM: recursive parse succeeded, sig = 15340826253708983785, new planId = 2239163167
SPM: add new plan: sig = 15340826253708983785, planId = 2239163167
SPM: new plan added to existing plan baseline
, sig = 15340826253708983785, planId = 2239163167
```

```
SQL> alter session set optimizer_use_sql_plan_baselines = TRUE;
-- back to normal OFE
SQL> alter session set optimizer_features_enable='12.1.0.1.1';

SQL> declare
      rs pls_integer;
begin
  -- disabling the full table SPM plan
  rs := dbms_spm.alter_sql_plan_baseline
    (plan_name      => 'SQL_PLAN_d9tch6banyzg9616acf47'
     ,attribute_name => 'ENABLED'
     ,attribute_value => 'NO'
    );
end;
/
SQL> alter index i1 rename to rn_i1;
```

```
SQL> SELECT count(*), max(col2) FROM t1 where flag = 'Y1';
```

```
-----  
SQL_ID 4sdth4kka4ykw, child number 0  
-----
```

```
Plan hash value: 2491235600  
-----
```

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) |
|----|-------------------------------------|-------|------|-------|-------------|
| 0 | SELECT STATEMENT | | | | 2 (100) |
| 1 | SORT AGGREGATE | | 1 | 54 | |
| 2 | TABLE ACCESS BY INDEX ROWID BATCHED | T1 | 1 | 54 | 2 (0) |
| * | INDEX RANGE SCAN | RN_I1 | 1 | | 1 (0) |

```
-----
```

```
Predicate Information (identified by operation id):  
-----
```

```
3 - access("FLAG"=:SYS_B_0)
```

```
SPM: setup to add new plan to existing plan baseline  
, sig = 15340826253708983785, planId = 842046253  
SPM: sql stmt=SELECT count(*), max(col2) FROM t1 where flaG = :"SYS_B_0"
```

```
SQL> select
      a.plan_name
      ,b.plan_id
      ,a.enabled
      ,a.accepted
  from
    dba_sql_plan_baselines a
   ,sys.sqlobj$ b
 where
      a.signature = b.signature
  and a.plan_name = b.name
  and b.plan_id = 842046253;
```

| PLAN_NAME | PLAN_ID | ENA | ACC |
|--------------------------------|-----------|-----|-----|
| SQL_PLAN_d9tch6banyzg932309b2d | 842046253 | YES | NO |

SPM: planId's of plan baseline are: **2015585387**
SPM: using qksan to reproduce, cost and select accepted plan, sig = 15340826253708983785
SPM: plan reproducibility round 1(plan outline + session OFE)
SPM: using qksan to reproduce accepted plan, planId = 2015585387
SPM: plan reproducibility - session OFE = **99999999**, hinted OFE = 12010001

```
SPM: planId in plan baseline = 2015585387, planId of reproduced plan = 842046253
----- START SPM Plan Dump -----
SPM: failed to reproduce the plan using the following info:
parse_schema name      : C##MHOURI
plan_baseline signature : 15340826253708983785
plan_baseline plan_id   : 2015585387
plan_baseline hintset   :
  hint num 1 len 27 text: IGNORE_OPTIM_EMBEDDED_HINTS
  hint num 2 len 37 text: OPTIMIZER_FEATURES_ENABLE('12.1.0.1')
  hint num 3 len 22 text: DB_VERSION('12.1.0.1')
  hint num 4 len 8 text: ALL_ROWS
  hint num 5 len 22 text: OUTLINE_LEAF(@"SEL$1")
  hint num 6 len 49 text: INDEX_RS_ASC(@"SEL$1" "T1"@"SEL$1" ("T1"."FLAG"))
  hint num 7 len 50 text: BATCH_TABLE_ACCESS_BY_ROWID(@"SEL$1" "T1"@"SEL$1")
SPM: generated non-matching plan:
```

SPM: plan reproducibility round 1 (**plan outline only**)

SPM: using qksan to reproduce accepted plan, planId = 2015585387

```
SPM: planId in plan baseline = 2015585387, planId of reproduced plan = 842046253
----- START SPM Plan Dump -----
SPM: failed to reproduce the plan using the following info:
parse_schema name      : C##MHOURI
plan_baseline signature : 15340826253708983785
plan_baseline plan_id   : 2015585387
plan_baseline hintset   :
  hint num 1 len 27 text: IGNORE_OPTIM_EMBEDDED_HINTS
  hint num 2 len 37 text: OPTIMIZER_FEATURES_ENABLE('12.1.0.1')
  hint num 3 len 22 text: DB_VERSION('12.1.0.1')
  hint num 4 len 8 text: ALL_ROWS
  hint num 5 len 22 text: OUTLINE_LEAF(@"SEL$1")
  hint num 6 len 49 text: INDEX_RS_ASC(@"SEL$1" "T1"@"SEL$1" ("T1"."FLAG"))
  hint num 7 len 50 text: BATCH_TABLE_ACCESS_BY_ROWID(@"SEL$1" "T1"@"SEL$1")
```

SPM: plan reproducibility round 2 (**hinted OFE only**)

SPM: using qksan to reproduce accepted plan, planId = 2015585387

```
SPM: planId in plan baseline = 2015585387, planId of reproduced plan = 3153386212
----- START SPM Plan Dump -----
SPM: failed to reproduce the plan using the following info:
  parse_schema name      : C##MHOURI
  plan_baseline signature : 15340826253708983785
  plan_baseline plan_id   : 2015585387
  plan_baseline hintset   :
    hint num 1 len 37 text: OPTIMIZER_FEATURES_ENABLE('12.1.0.1')
```

```
SPM: change REPRODUCED status to NO, planName = SQL_PLAN_d9tch6banyzg97823646b
SPM: REPRODUCED status changes: cntRepro = 1, bitvecRepro = 000
SPM: couldn't reproduce any enabled+accepted plan so using
the cost-based plan, planId = 842046253
```

```
INDEX_RS_ASC(@"SEL$1" "T1"@ "SEL$1" ("T1"."FLAG"))
```

```

SQL> create table t_range
(
  id          number          not null,
  X           varchar2(30 char) not null,
  D           date,
  C1          number
)
partition by range (id)
(
  partition P_10000 values less than (10000) ,
  partition P_20000 values less than (20000) ,
  partition P_30000 values less than (30000) ,
  partition P_40000 values less than (40000) ,
  partition P_50000 values less than (50000) ,
  partition P_60000 values less than (60000)
);

SQL> create index t_r_i1 on t_range(id, c1);

SQL> select * from t_range where id =150 and c1 = 42;

SQL> select * from table(dbms_xplan.display_cursor);

SQL_ID  by2nsk6r62312, child number 0
-----
-----| Id  | Operation                                | Name    | Rows  |
-----| 0   | SELECT STATEMENT                           |
-----| 1   | TABLE ACCESS BY GLOBAL INDEX ROWID BATCHED| T_RANGE | 1     |
-----|* 2   | INDEX RANGE SCAN                          | T_R_I1  | 1     |
-----|* 2   | INDEX RANGE SCAN                          | T_R_I1  | 1     |

-----| Predicate Information (identified by operation id): |
-----| 2 - access("ID"=:SYS_B_0 AND "C1"=:SYS_B_1) |
-----| Note
-----| - dynamic statistics used: dynamic sampling (level=2)

```

```

SQL> SELECT
      p.sql_id
    ,p.plan_hash_value
    ,p.child_number
    ,t.phv2
  FROM
    v$sql_plan p
   ,xmltable('for $i in /other_xml/info
              where $i/@type eq "plan_hash_2"
              return $i'
             passing xmltype(p.other_xml)
             columns phv2 number path '/') t
 WHERE p.sql_id = 'by2nsk6r62312'
 AND   p.other_xml is not null;

```

| SQL_ID | PLAN_HASH_VALUE | CHILD_NUMBER | PHV2 |
|---------------|-----------------|--------------|-------------------|
| by2nsk6r62312 | 2479844656 | 0 | 2995765617 |

```

SQL> alter index t_r_i1 rename to t_r_i2;
SQL> select * from t_range where id =150 and c1 = 42;
SQL> select * from table(dbms_xplan.display_cursor);

```

SQL\_ID by2nsk6r62312, child number 0

| Id | Operation | Name | Rows |
|-----|--|---------------|------|
| 0 | SELECT STATEMENT | | |
| 1 | TABLE ACCESS BY GLOBAL INDEX ROWID BATCHED | T_RANGE | 1 |
| * 2 | INDEX RANGE SCAN | T_R_I2 | 1 |

Predicate Information (identified by operation id):

```
2 - access ("ID"=:SYS_B_0 AND "C1"=:SYS_B_1)
```

Note

- dynamic statistics used: dynamic sampling (level=2)

```

SQL> SELECT
      p.sql_id
    ,p.plan_hash_value
    ,p.child_number
    ,t.phv2
  FROM
    v$sql_plan p
   ,xmltable('for $i in /other_xml/info
              where $i/@type eq "plan_hash_2"
              return $i'
             passing xmltype(p.other_xml)
             columns phv2 number path '/') t
 WHERE p.sql_id = 'by2nsk6r62312'
 AND   p.other_xml is not null;

```

| SQL_ID | PLAN_HASH_VALUE | CHILD_NUMBER | PHV2 |
|---------------|-----------------|--------------|-------------------|
| by2nsk6r62312 | 1375529486 | 0 | 2863680406 |

```

SQL> drop index t_r_i2;

SQL> create index t_r_i1 on t_range(id,c1) local;

SQL> select * from t_range where id =150 and c1 = 42;

SQL> select * from table(dbms_xplan.display_cursor);

SQL_ID  by2nsk6r62312, child number 0
-----
| Id  | Operation          | Name   | Rows |
|-----|
| 0  | SELECT STATEMENT   |         |       |
| 1  | PARTITION RANGE SINGLE |       |       |
| 2  | TABLE ACCESS BY LOCAL INDEX ROWID BATCHED | T_RANGE | 1    |
|* 3  | INDEX RANGE SCAN   | T_R_I1 | 1    |

Predicate Information (identified by operation id):
-----
      3 - access("ID"=:SYS_B_0 AND "C1"=:SYS_B_1)

Note
-----
- dynamic statistics used: dynamic sampling (level=2)

SQL> SELECT
      p.sql_id
     ,p.plan_hash_value
     ,p.child_number
     ,t.phv2
  FROM
    v$sql_plan p
   ,xmltable('for $i in /other_xml/info
              where $i/@type eq "plan_hash_2"
              return $i'
             passing xmltype(p.other_xml)
             columns phv2 number path '/') t
 WHERE p.sql_id = 'by2nsk6r62312'
   AND p.other_xml is not null;

SQL_ID      PLAN_HASH_VALUE CHILD_NUMBER      PHV2
-----      -----          -----      -----
by2nsk6r62312      4134837294          0  794144393

```

```

SQL> drop index t_r_i1;

SQL> create index t_r_i1 on t_range(id,c1, d desc);

SQL> select * from t_range where id =150 and c1 = 42;

SQL> select * from table(dbms_xplan.display_cursor);

SQL_ID  by2nsk6r62312, child number 0
-----
-----| Id  | Operation          | Name    | Rows  |
-----| 0  | SELECT STATEMENT   |          |        |
| 1  | TABLE ACCESS BY GLOBAL INDEX ROWID BATCHED | T_RANGE | 1     |
|* 2  | INDEX RANGE SCAN   | T_R_I1  | 1     |
-----
```

Predicate Information (identified by operation id):

```
2 - access("ID"=:SYS_B_0 AND "C1"=:SYS_B_1)
```

Note

- dynamic statistics used: dynamic sampling (level=2)

```

SQL> SELECT
      p.sql_id
      ,p.plan_hash_value
      ,p.child_number
      ,t.phv2
  FROM
      v$sql_plan p
      ,xmltable('for $i in /other_xml/info
                  where $i/@type eq "plan_hash_2"
                  return $i'
                passing xmltype(p.other_xml)
                columns phv2 number path '/') t
 WHERE p.sql_id = 'by2nsk6r62312'
 AND   p.other_xml is not null;
```

| SQL_ID | PLAN_HASH_VALUE | CHILD_NUMBER | PHV2 |
|---------------|-----------------|--------------|-------------------|
| by2nsk6r62312 | 2479844656 | 0 | 2995765617 |

```

SQL> drop index t_r_i1;

SQL> create index t_r_i1 on t_range(id,c1) reverse;

SQL> select * from t_range where id =150 and c1 = 42;

SQL> select * from table(dbms_xplan.display_cursor);

SQL_ID  by2nsk6r62312, child number 0
-----
select * from t_range where id =:"SYS_B_0" and c1 = :"SYS_B_1"
-----
| Id  | Operation          | Name    | Rows |
|-----|-----|-----|-----|
| 0   | SELECT STATEMENT   |         |       |
| 1   |  TABLE ACCESS BY GLOBAL INDEX ROWID BATCHED | T_RANGE | 1    |
|* 2  |    INDEX RANGE SCAN | T_R_I1  | 1    |
|-----|-----|-----|-----|
Predicate Information (identified by operation id):
-----
 2 - access("ID"=:SYS_B_0 AND "C1"=:SYS_B_1)

Note
-----
- dynamic statistics used: dynamic sampling (level=2)

SQL> SELECT
      p.sql_id
     ,p.plan_hash_value
     ,p.child_number
     ,t.phv2
  FROM
    v$sql_plan p
   ,xmltable('for $i in /other_xml/info
              where $i/@type eq "plan_hash_2"
              return $i'
             passing xmltype(p.other_xml)
             columns phv2 number path '/') t
 WHERE p.sql_id = 'by2nsk6r62312'
   AND p.other_xml is not null;

SQL_ID      PLAN_HASH_VALUE CHILD_NUMBER      PHV2
-----      -----      -----      -----
by2nsk6r62312      2479844656      0 2995765617

```

```
SQL> create table t
      (c1 varchar2(64), c2 char(15), d1 date);

SQL> insert into t
      select
          mod(abs(dbms_random.random),3)+1||chr(ascii('Y'))
        ,dbms_random.string('L',dbms_random.value(1,5))||rownum
        ,to_date(to_char(to_date('01/01/1980','dd/mm/yyyy'),'J') +
          trunc(dbms_random.value(1,11280)),'J')
      from dual
      connect by level <= 1e3;

SQL> alter table t add constraint t_pk primary key (c1,c2);
```

```

SQL> show parameter nls_sort
NAME                      TYPE        VALUE
-----                    -----      -----
nls_sort                  string      BINARY

SQL> alter session set optimizer_capture_sql_plan_baselines=TRUE;
SQL> select
      c1
    from t
   group by c1
  order by c1 asc nulls last;

SQL> /
SQL> alter session set optimizer_capture_sql_plan_baselines=FALSE;

SQL> select
      c1
    from t
   group by c1
  order by c1 asc nulls last;

SQL> select * from table(dbms_xplan.display_cursor);

-----
SQL_ID  16nxkzb12va0b, child number 1
-----
Plan hash value: 1476560607
-----
| Id  | Operation          | Name | Rows | Bytes |
|---|---|---|---|---|
| 0  | SELECT STATEMENT  |       |       |       |
| 1  |   SORT GROUP BY   |       | 1000 | 34000 |
| 2  |     TABLE ACCESS FULL| T   | 1000 | 34000 |
-----
Note
-----
- dynamic sampling used for this statement (level=2)
- SQL plan baseline SQL_PLAN_90sg67694zwyj4b85249e used for this statement

```

```

SQL> -- if running 11g
SQL> select
      extractValue(value(i),'.') outline_hints
  from
    sys.sqlobj$data sqd,
    table(xmlsequence(extract(xmltype(sqd.comp_data), '/outline_data/hint')))i
 where
  signature in (select signature from
                 dba_sql_plan_baselines
                where plan_name = 'SQL_PLAN_90sg67694zwyj4b85249e');

OUTLINE_HINTS
-----
FULL(@"SEL$1" "T":@"SEL$1")
OUTLINE_LEAF(@"SEL$1")
ALL_ROWS
DB_VERSION('11.2.0.3')
OPTIMIZER_FEATURES_ENABLE('11.2.0.3')
IGNORE_OPTIM_EMBEDDED_HINTS

SQL> -- if running 12c
SQL> select
      substr(extractvalue(value(d), '/hint'), 1, 100) as outline_hints
  from
    xmltable('/outline_data/hint'
            passing (
              select
                xmltype(other_xml) as xmlval
              from
                sys.sqlobj$plan
              where
                signature in (select
                               signature
                               from dba_sql_plan_baselines
                               where plan_name = 'SQL_PLAN_90sg67694zwyj4b85249e'
                           )
              and other_xml is not null
            )
        ) d;

OUTLINE_HINTS
-----
FULL(@"SEL$1" "T":@"SEL$1")
OUTLINE_LEAF(@"SEL$1")
ALL_ROWS
DB_VERSION('12.1.0.1')
OPTIMIZER_FEATURES_ENABLE('12.1.0.1.1')
IGNORE_OPTIM_EMBEDDED_HINTS

```

```
SQL> alter session set nls_sort = FRENCH;
SQL> select
      c1
    from t
   group by c1
  order by c1 asc nulls last;

SQL> select * from table(dbms_xplan.display_cursor);

-----
SQL_ID  16nxkhb12va0b, child number 2

An uncaught error happened in prepare_sql_statement: ORA-01403: no data found

NOTE: cannot fetch plan for SQL_ID: 16nxkhb12va0b, CHILD_NUMBER: 2
      Please verify value of SQL_ID and CHILD_NUMBER;
      It could also be that the plan is no longer in cursor cache (check v$sql_plan)
```

```

SQL> select
      c1
    from t
   group by c1
  order by c1 asc nulls last;

SQL> select * from table(dbms_xplan.display_cursor(format=>'advanced'));
-----
SQL_ID  16nxkxb12va0b, child number 1
-----

| Id | Operation          | Name | Rows | Bytes | Cost (%CPU) | Time |
|----|----|----|----|----|----|----|
| 0 | SELECT STATEMENT   |       |       |       |       |       |
| 1 |   SORT ORDER BY    |       | 1000 | 34000 | 5 (40) | 00:00:01 |
| 2 |   HASH GROUP BY    |       | 1000 | 34000 | 5 (40) | 00:00:01 |
| 3 |   TABLE ACCESS FULL| T    | 1000 | 34000 | 3 (0)  | 00:00:01 |

Column Projection Information (identified by operation id):
-----
1 - (#keys=1) NLSSORT("C1",'nls_sort=''FRENCH''') [522],
  "C1" [VARCHAR2,64]
2 - "C1" [VARCHAR2,64]
3 - (rowset=200) "C1" [VARCHAR2,64]

```

Note

- dynamic sampling used for this statement (level=2)

```

SQL> create table t1
  as
  with generator as (
    select --+ materialize
      rounum id
    from dual
    connect by
      level <= 10000)
  select
    rounum           id,
    trunc(dbms_random.value(1,1000))   n1,
    lpad(rounum,10,'0') small_vc,
    rpad('x',100)      padding
  from
    generator v1,
    generator v2
  where
    rounum <= 1000000;

SQL> create index t1_n1 on t1(id, n1);

SQL> create table t2
  as
  with generator as (
    select --+ materialize
      rounum id
    from dual
    connect by
      level <= 10000)
  select
    rounum           id,
    trunc(dbms_random.value(10001,20001))  x1,
    lpad(rounum,10,'0') small_vc,
    rpad('x',100)      padding
  from
    generator v1,
    generator v2
  where
    rounum <= 1000000;

SQL> create index t2_i1 on t2(x1);

SQL> create index ind_t1_extra on t1(id);

SQL> exec dbms_stats.gather_table_stats(user, 't2', method_opt => 'for all columns size 1');
SQL> exec dbms_stats.gather_table_stats(user, 't1', method_opt => 'for all columns size 1');

```

```
SQL> show parameter optimizer_mode

NAME                      TYPE        VALUE
-----
optimizer_mode            string      ALL_ROWS

SQL> alter session set optimizer_capture_sql_plan_baselines=TRUE;

SQL> select * from t1 where id in (select id from t2 where x1 = 17335) order by id;

SQL> /
SQL> alter session set optimizer_capture_sql_plan_baselines=FALSE;
```

```
SQL> select * from t1 where id in (select id from t2 where x1 = 17335) order by id;  
88 rows selected.
```

Elapsed: 00:00:00.18

```
SQL> select * from table(dbms_xplan.display_cursor);
```

| Id Operation | Name | Rows | Bytes |
|----------------|-----------------------------|--------------|-------|
| 0 | SELECT STATEMENT | | |
| 1 | NESTED LOOPS | | |
| 2 | NESTED LOOPS | 100 | 13100 |
| 3 | SORT UNIQUE | 100 | 1000 |
| 4 | TABLE ACCESS BY INDEX ROWID | T2 | 100 |
| * | INDEX RANGE SCAN | T2_I1 | 100 |
| * | INDEX RANGE SCAN | IND_T1_EXTRA | 1 |
| 7 | TABLE ACCESS BY INDEX ROWID | T1 | 1 |
| | | | 121 |

Predicate Information (identified by operation id):

```
5 - access("X1"=17335)  
6 - access("ID"="ID")
```

Note

- SQL plan baseline SQL\_PLAN\_bjazgx0cv6xtwff8eddc2 used for this statement

```
SQL> -- if running 11g
SQL> select
      extractValue(value(i),'.') outline_hints
    from
      sys.sqlobj$data sqd,
      table(xmlsequence(extract(xmltype(sqd.comp_data), '/outline_data/hint')))i
   where
     exists (select null
              from
                dba_sql_plan_baselines dsb
             where dsb.signature = sqd.signature
               and dsb.plan_name = 'SQL_PLAN_bjazgx0cv6xtwff8eddc2'
            );

```

OUTLINE\_HINTS

```
-----  
NLJ_BATCHING(@"SEL$5DA710D3" "T1@""SEL$1")  
USE_NL(@"SEL$5DA710D3" "T1@""SEL$1")  
LEADING(@"SEL$5DA710D3" "T2"@"SEL$2" "T1@""SEL$1")  
INDEX(@"SEL$5DA710D3" "T1@""SEL$1" ("T1"."ID"))  
INDEX_RS_ASC(@"SEL$5DA710D3" "T2"@"SEL$2" ("T2"."X1"))  
OUTLINE(@"SEL$2")  
OUTLINE(@"SEL$1")  
UNNEST(@"SEL$2")  
OUTLINE_LEAF(@"SEL$5DA710D3")  
ALL_ROWS  
DB_VERSION('11.2.0.3')  
OPTIMIZER_FEATURES_ENABLE('11.2.0.3')  
IGNORE_OPTIM_EMBEDDED_HINTS
```

```
SQL> -- if running 12c
```

```
SQL> select
      substr(extractvalue(value(d), '/hint'), 1, 100) as outline_hints
    from
      xmltable('/outline_data/hint'
      passing (select
                  xmltype(other_xml) as xmlval
                from
                  sys.sqlobj$plan
```

```
        where
        signature in (select
                        signature
                   from
                       dba_sql_plan_baselines
                  where plan_name = 'SQL_PLAN_aw0ntx07w1vggff8eddc2'
                    )
      and other_xml is not null
    )
  ) d;
```

OUTLINE\_HINTS

```
SEMI_TO_INNER(@"SEL$5DA710D3" "T2":@"SEL$2")
NLJ_BATCHING(@"SEL$5DA710D3" "T1":@"SEL$1")
USE_NL(@"SEL$5DA710D3" "T1":@"SEL$1")
LEADING(@"SEL$5DA710D3" "T2":@"SEL$2" "T1":@"SEL$1")
INDEX(@"SEL$5DA710D3" "T1":@"SEL$1" ("T1"."ID"))
INDEX_RS_ASC(@"SEL$5DA710D3" "T2":@"SEL$2" ("T2"."X1"))
OUTLINE(@"SEL$2")
OUTLINE(@"SEL$1")
UNNEST(@"SEL$2")
OUTLINE_LEAF(@"SEL$5DA710D3")
ALL_ROWS
DB_VERSION('12.1.0.1')
OPTIMIZER_FEATURES_ENABLE('12.1.0.1.1')
IGNORE_OPTIM_EMBEDDED_HINTS
```

```
SQL> alter session set optimizer_mode = first_rows;
```

```
SQL> alter session set optimizer_use_sql_plan_baselines = FALSE;  
SQL> select * from t1 where id in (select id from t2 where x1 = 17335) order by id;  
88 rows selected.  
Elapsed: 00:01:54.61
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

| Id | Operation | Name | Rows | Bytes |
|-----|-----------------------------|---------------------|-------|-------|
| 0 | SELECT STATEMENT | | | |
| 1 | NESTED LOOPS SEMI | | 100 | 13100 |
| 2 | TABLE ACCESS BY INDEX ROWID | T1 | 1000K | 115M |
| 3 | INDEX FULL SCAN | IND_T1_EXTRA | 1000K | |
| * 4 | TABLE ACCESS BY INDEX ROWID | T2 | 1 | 10 |
| * 5 | INDEX RANGE SCAN | T2_I1 | 100 | |

```
Predicate Information (identified by operation id):
```

```
4 - filter("ID"="ID")  
5 - access("X1"=17335)
```

```
SQL> alter session set optimizer_use_sql_plan_baselines=TRUE;  
SQL> select * from t1 where id in (select id from t2 where x1 = 17335) order by id;  
SQL> select * from table(dbms_xplan.display_cursor);
```

| Id | Operation | Name | Rows | Bytes |
|----|-----------------------------|---------------------|------|-------|
| 0 | SELECT STATEMENT | | | |
| 1 | NESTED LOOPS | | | |
| 2 | NESTED LOOPS | | 100 | 13100 |
| 3 | SORT UNIQUE | | 100 | 1000 |
| 4 | TABLE ACCESS BY INDEX ROWID | T2 | 100 | 1000 |
| * | INDEX RANGE SCAN | T2_I1 | 100 | |
| * | INDEX RANGE SCAN | IND_T1_EXTRA | 1 | |
| 7 | TABLE ACCESS BY INDEX ROWID | T1 | 1 | 121 |

Predicate Information (identified by operation id):

```
-----  
5 - access("X1"=17335)  
6 - access("ID"="ID")
```

Note

```
-----  
- SQL plan baseline SQL_PLAN_bjazgx0cv6xtwff8eddc2 used for this statement
```

```

SQL> alter session set optimizer_mode=all_rows;

SQL> select count(*), max(col2) from t1 where flag = 'Y1';

SQL> select
      child_number
    , is_bind_aware
    , is_bind_sensitive
    , to_char(exact_matching_signature) sig
      , plan_hash_value
  from v$sql
 where sql_id = '6fbvysnhkvugw'
   and is_shareable = 'Y';

```

| CHILD_NUMBER | IS_BIND_AWARE | IS_BIND_SENSITIVE | SIG | PLAN_HASH_VALUE |
|--------------|---------------|-------------------|----------------------|-----------------|
| 1 | Y | Y | 15340826253708983785 | 3724264953 |
| 2 | Y | Y | 15340826253708983785 | 3625400295 |

```

SQL> declare
      rs pls_integer;
begin
  rs := dbms_spm.load_plans_from_cursor_cache('6fbvysnhkvugw');
end;
/
SQL> select
      to_char(signature) sig
     ,plan_name
     ,enabled
     ,accepted
  from
    dba_sql_plan_baselines;

SIG          PLAN_NAME           ENABLED ACCEPTED
-----  -----
15340826253708983785 SQL_PLAN_d9tch6banyzg9616acf47 YES      YES -> full table scan
15340826253708983785 SQL_PLAN_d9tch6banyzg97823646b YES      YES -> index range scan

```

```
SQL> select * from v$version where rownum =1;
```

BANNER

```
-----  
Oracle Database 11g Enterprise Edition Release 11.2.0.3.0 - 64bit Production
```

```
SQL> select count(*), max(col2) from t1 where flag = 'Y1';
```

| Id | Operation | Name | Rows | Bytes |
|----|-----------------------------|------|------|-------|
| 0 | SELECT STATEMENT | | | |
| 1 | SORT AGGREGATE | | 1 | 54 |
| 2 | TABLE ACCESS BY INDEX ROWID | T1 | 18 | 972 |
| * | INDEX RANGE SCAN | I1 | 18 | |

Predicate Information (identified by operation id):

```
-----  
3 - access("FLAG"=:SYS_B_0)
```

Note

```
- SQL plan baseline SQL_PLAN_d9tch6banyzg98576eb1f used for this statement
```

```
SQL> select count(*), max(col2) from t1 where flag = 'N1';
```

| Id | Operation | Name | Rows | Bytes |
|----|-------------------|------|-------|-------|
| 0 | SELECT STATEMENT | | | |
| 1 | SORT AGGREGATE | | 1 | 54 |
| * | TABLE ACCESS FULL | T1 | 50505 | 2663K |

Predicate Information (identified by operation id):

```
-----  
2 - filter("FLAG"=:SYS_B_0)
```

Note

```
- SQL plan baseline SQL_PLAN_d9tch6banyzg9616acf47 used for this statement
```

```

SQL> select
      sql_id
    , child_number
    , is_bind_aware
    , is_bind_sensitive
    , to_char(exact_matching_signature) sig
    , executions
    , plan_hash_value
  from v$sql
 where sql_id = '6fbvysnhkvugw'
   and is_shareable = 'Y'
;

```

| SQL_ID | CHILD_NUMBER | I | I | SIG | EXECUTIONS | PLAN_HASH_VALUE |
|---------------|--------------|---|---|----------------------|------------|-----------------|
| 6fbvysnhkvugw | 1 | Y | Y | 15340826253708983785 | 1 | 3724264953 |
| 6fbvysnhkvugw | 2 | Y | Y | 15340826253708983785 | 1 | 3625400295 |
| 6fbvysnhkvugw | 5 | Y | Y | 15340826253708983785 | 5 | 3625400295 |
| 6fbvysnhkvugw | 6 | Y | Y | 15340826253708983785 | 4 | 3724264953 |

```
SQL> create index i2 on t1(flag,col2);

SQL> select count(*), max(col2) from t1 where flag = 'N2';

SQL> select * from table(dbms_xplan.display_cursor);

-----
SQL_ID  6fbvysnhkvugw, child number 7

An uncaught error happened in prepare_sql_statement: ORA-01403: no data found
NOTE: cannot fetch plan for SQL_ID: 6fbvysnhkvugw, CHILD_NUMBER: 7
      Please verify value of SQL_ID and CHILD_NUMBER;
      It could also be that the plan is no longer in cursor cache (check v$sql_plan)
```

SQL\_ID 6fbvysnhkvugw, child number 7

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|----|---------------------|------|-------|-------|-------------|----------|
| 0 | SELECT STATEMENT | | | | 273 (100) | |
| 1 | SORT AGGREGATE | | 1 | 54 | | |
| * | 2 TABLE ACCESS FULL | T1 | 49749 | 2623K | 273 (1) | 00:00:04 |

Predicate Information (identified by operation id):

2 - filter("FLAG"=:SYS\_B\_0)

Note

- SQL plan baseline SQL\_PLAN\_d9tch6banyzg9616acf47 used for this statement

```
SQL> select count(*), max(col2) from t1 where flag = 'Y2';
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

```
SQL_ID  6fbvysnhkvugw, child number 7
```

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|----|---------------------|------|-------|-------|-------------|----------|
| 0 | SELECT STATEMENT | | | | 273 (100) | |
| 1 | SORT AGGREGATE | | 1 | 54 | | |
| * | 2 TABLE ACCESS FULL | T1 | 49749 | 2623K | 273 (1) | 00:00:04 |

```
Predicate Information (identified by operation id):
```

```
2 - filter("FLAG"=:SYS_B_0)
```

```
Note
```

```
- SQL plan baseline SQL_PLAN_d9tch6banyzg9616acf47 used for this statement
```

```
SQL> select count(*), max(col2) from t1 where flag = 'Y2';
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

```
SQL_ID 6fbvysnhkvugw, child number 7
```

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|----|--------------------------|------|-------|-------|-------------|----------|
| 0 | SELECT STATEMENT | | | | 273 (100) | |
| 1 | SORT AGGREGATE | | 1 | 54 | | |
| * | TABLE ACCESS FULL | T1 | 49749 | 2623K | 273 (1) | 00:00:04 |

```
Predicate Information (identified by operation id):
```

```
2 - filter("FLAG"=:SYS_B_0)
```

```
Note
```

```
- SQL plan baseline SQL_PLAN_d9tch6banyzg9616acf47 used for this statement
```

```

SQL> select
      ,sql_id
      ,child_number
      ,is_bind_aware
      ,is_bind_sensitive
      ,to_char(exact_matching_signature) sig
      ,executions
      ,plan_hash_value
  from v$sql
 where sql_id = '6fbvysnhkvugw'
 and is_shareable = 'Y'
 ;

```

| SQL_ID | CHILD_NUMBER | I | I | SIG | EXECUTIONS | PLAN_HASH_VALUE |
|---------------|--------------|---|---|------------------------|------------|-----------------|
| 6fbvysnhkvugw | | 1 | Y | Y 15340826253708983785 | 1 | 3724264953 |
| 6fbvysnhkvugw | | 2 | Y | Y 15340826253708983785 | 1 | 3625400295 |
| 6fbvysnhkvugw | | 5 | N | N 15340826253708983785 | 14 | 3724264953 |

```
SQL> alter session set optimizer_use_sql_plan_baselines = FALSE;
```

```
SQL> select count(*), max(col2) from t1 where flag = 'Y2';
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|----|-----------------------------|------|------|-------|-------------|----------|
| 0 | SELECT STATEMENT | | | | 2 (100) | |
| 1 | SORT AGGREGATE | | 1 | 54 | | |
| 2 | TABLE ACCESS BY INDEX ROWID | T1 | 19 | 1026 | 2 (0) | 00:00:01 |
| * | INDEX RANGE SCAN | I1 | 19 | | 1 (0) | 00:00:01 |

Predicate Information (identified by operation id):

```
3 - access("FLAG"=:SYS_B_0)
```

```
SQL> alter session set optimizer_use_sql_plan_baselines = TRUE;
```

```
SQL> select count(*), max(col2) from t1 where flag = 'Y2';
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

| Id Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|----------------------|------|------|-------|-------------|------|
| 0 SELECT STATEMENT | | | | 273 (100) | |
| 1 SORT AGGREGATE | | 1 | 54 | | |
| | | | | | |

Predicate Information (identified by operation id):

```
2 - filter("FLAG"=:SYS_B_0)
```

Note

```
- SQL plan baseline SQL_PLAN_d9tch6banyzg9616acf47 used for this statement
```

```
SQL> select
      sql_id
    ,child_number
    ,is_bind_aware
    ,is_bind_sensitive
    ,to_char(exact_matching_signature) sig
    ,executions
    ,plan_hash_value
  from v$sql
 where sql_id = '6fbvysnhkvugw'
 and is_shareable = 'Y'
 ;
```

| SQL_ID | CHILD_NUMBER | I | I | SIG | EXECUTIONS | PLAN_HASH_VALUE |
|---------------|--------------|---|---|----------------------|------------|-----------------|
| 6fbvysnhkvugw | 1 | N | Y | 15340826253708983785 | 1 | 3625400295 |
| 6fbvysnhkvugw | 5 | N | N | 15340826253708983785 | 15 | 3724264953 |

```
SQL> select * from v$version where rownum =1;
```

BANNER

CON\_ID

```
-----  
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production 0
```

```
SQL> select count(*), max(col2) from t1 where flag = 'Y1';
```

| Id Operation | Name | Rows | Bytes |
|----------------|-------------------------------------|------|-------|
| 0 | SELECT STATEMENT | | |
| 1 | SORT AGGREGATE | | 54 |
| 2 | TABLE ACCESS BY INDEX ROWID BATCHED | T1 | 54 |
| * 3 | INDEX RANGE SCAN | I1 | 54 |

Predicate Information (identified by operation id):

```
-----  
3 - access("FLAG"=:SYS_B_0)
```

Note

```
- SQL plan baseline SQL_PLAN_d9tch6banyzg97823646b used for this statement
```

```
SQL> select count(*), max(col2) from t1 where flag = 'N1';
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

| Id Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|----------------|-------------------|------|-------|-------------|------------------|
| 0 | SELECT STATEMENT | | | 273 (100) | |
| 1 | SORT AGGREGATE | 1 | 54 | | |
| * 2 | TABLE ACCESS FULL | T1 | 49874 | 2630K | 273 (1) 00:00:01 |

Predicate Information (identified by operation id):

```
-----  
2 - filter("FLAG"=:SYS_B_0)
```

Note

```
- SQL plan baseline SQL_PLAN_d9tch6banyzg9616acf47 used for this statement
```

```
SQL> create index i2 on t1(flag,col2);

SQL> select count(*), max(col2) from t1 where flag = 'N2';

SQL> select * from table(dbms_xplan.display_cursor);
-----

| Id | Operation                | Name | Rows  | Bytes | Cost (%CPU) | Time     |
|----|--------------------------|------|-------|-------|-------------|----------|
| 0  | SELECT STATEMENT         |      |       |       | 273 (100)   |          |
| 1  | SORT AGGREGATE           |      | 1     | 54    |             |          |
| *  | <b>TABLE ACCESS FULL</b> | T1   | 49874 | 2630K | 273 (1)     | 00:00:01 |


```

Predicate Information (identified by operation id):

```
-----  
2 - filter("FLAG"=:SYS_B_0)
```

Note

```
-----  
- SQL plan baseline SQL_PLAN_d9tch6banyzg9616acf47 used for this statement
```

```
SQL> select count(*), max(col2) from t1 where flag = 'Y2';
```

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|----|--------------------------|------|-------|-------|-------------|----------|
| 0 | SELECT STATEMENT | | | | 273 (100) | |
| 1 | SORT AGGREGATE | | 1 | 54 | | |
| * | TABLE ACCESS FULL | T1 | 49874 | 2630K | 273 (1) | 00:00:01 |

Predicate Information (identified by operation id):

```
2 - filter("FLAG"=:SYS_B_0)
```

Note

```
- SQL plan baseline SQL_PLAN_d9tch6banyzg9616acf47 used for this statement
```

```
SQL> select count(*), max(col2) from t1 where flag = 'Y2';
```

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) |
|----|-------------------------------------|------|------|-------|-------------|
| 0 | SELECT STATEMENT | | | | 2 (100) |
| 1 | SORT AGGREGATE | | 1 | 54 | |
| 2 | TABLE ACCESS BY INDEX ROWID BATCHED | T1 | 1 | 54 | 2 (0) |
| * | INDEX RANGE SCAN | I1 | 1 | | 1 (0) |

Predicate Information (identified by operation id):

```
-----  
3 - access("FLAG"=:SYS_B_0)
```

Note

```
-----  
- SQL plan baseline SQL_PLAN_d9tch6banyzg97823646b used for this statement
```

```
SQL> select count(*), max(col2) from t1 where flag = 'N2';
```

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|----|-------------------|------|-------|-------|-------------|----------|
| 0 | SELECT STATEMENT | | | | 273 (100) | |
| 1 | SORT AGGREGATE | | 1 | 54 | | |
| * | TABLE ACCESS FULL | T1 | 49874 | 2630K | 273 (1) | 00:00:01 |

Predicate Information (identified by operation id):

```
2 - filter("FLAG"=:SYS_B_0)
```

Note

```
- SQL plan baseline SQL_PLAN_d9tch6banyzg9616acf47 used for this statement
```

```

SQL> select
      sql_id
    ,child_number
  ,is_bind_aware
  ,is_bind_sensitive
  ,to_char(exact_matching_signature) sig
  ,executions
  ,plan_hash_value
from v$sql
where sql_id = '6fbvysnhkvugw'
and is_shareable = 'Y';

```

| SQL_ID | CHILD_NUMBER | I | I | SIG | EXECUTIONS | PLAN_HASH_VALUE |
|---------------|--------------|---|---|----------------------|------------|-----------------|
| 6fbvysnhkvugw | 1 | Y | Y | 15340826253708983785 | 1 | 3724264953 |
| 6fbvysnhkvugw | 2 | Y | Y | 15340826253708983785 | 2 | 497086120 |
| 6fbvysnhkvugw | 4 | Y | Y | 15340826253708983785 | 1 | 497086120 |
| 6fbvysnhkvugw | 6 | Y | Y | 15340826253708983785 | 1 | 3724264953 |

```
SQL> create table t1
  as select
    rownum n1
   , trunc ((rownum-1)/3) n2
   , trunc(dbms_random.value(rownum, rownum*10)) n3
   , dbms_random.string('U', 10) c1
  from dual
 connect by level <= 3e6;
```

```
SQL> desc t1
```

| | Name | Null? | Type |
|---|------|-------|----------------------|
| 1 | N1 | | NUMBER |
| 2 | N2 | | NUMBER |
| 3 | N3 | | NUMBER |
| 4 | C1 | | VARCHAR2 (4000 CHAR) |

```
SQL> alter table t1 add C_DDL number DEFAULT 42 NOT NULL;
```

```
10.2.0.4.0> alter table t1 add C_DDL number default 42 not null;
```

Table altered.

Elapsed: 00:00:**48.53**

```
11.2.0.3.0> alter table t1 add C_DDL number default 42 not null;
```

Table altered.

Elapsed: 00:00:00.**04**

```
SQL> select count(1) from t1;
```

```
 COUNT(1)
```

```
-----  
 3000000
```

```
SQL> select count(1) from t1 where c_ddl = 42;
```

```
 COUNT(1)
```

```
-----  
 3000000
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|----|---------------------|------|-------|-------|-------------|----------|
| 0 | SELECT STATEMENT | | | | 3736 (100) | |
| 1 | SORT AGGREGATE | | 1 | 13 | | |
| * | 2 TABLE ACCESS FULL | T1 | 2712K | 33M | 3736 (1) | 00:00:45 |

Predicate Information (identified by operation id):

```
2 - filter(NVL("C_DDL",42)=42)
```

Note

```
- dynamic sampling used for this statement (level=2)
```

```
10.2.0.4.0 > select count(1) from t1 where c_ddl = 42;
```

```
COUNT(1)
```

```
-----
```

```
3000000
```

```
10.2.0.4.0> select * from table(dbms_xplan.display_cursor);
```

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|----|---------------------|------|-------|-------|-------------|----------|
| 0 | SELECT STATEMENT | | | | 4001 (100) | |
| 1 | SORT AGGREGATE | | 1 | 3 | | |
| * | 2 TABLE ACCESS FULL | T1 | 3000K | 8789K | 4001 (8) | 00:00:09 |

```
Predicate Information (identified by operation id):
```

```
2 - filter("C_DDL"=42)
```

```
SQL> SELECT
      segment_name
      ,bytes/1024/1024 mb
  FROM
    dba_segments
 WHERE
    segment_type = 'TABLE'
 AND segment_name = 'T1';
```

SEGMENT\_NAME MB

T1 112

```
SQL> alter table t1 add C_DDL number default 42 not null;
```

```
SQL> SELECT
      segment_name
      ,bytes/1024/1024 mb
  FROM
    dba_segments
 WHERE
    segment_type = 'TABLE'
 AND segment_name = 'T1';
```

SEGMENT\_NAME MB

T1 112

```
SQL> exec dbms_stats.gather_table_stats(user , 't1', method_opt => 'for all columns size 1' , no_invalidate => false);
```

```
SQL> select /*+ gather_plan_statistics */ count(1) from t1 where C_DDL = 42;
```

```
COUNT(1)
```

```
-----
```

```
3000000
```

```
SQL> select * from table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

| Id | Operation | Name | Starts | E-Rows | A-Rows | A-Time |
|----|-------------------|------|--------|--------|--------|-------------|
| 0 | SELECT STATEMENT | | 1 | | 1 | 00:00:00.60 |
| 1 | SORT AGGREGATE | | 1 | 1 | 1 | 00:00:00.60 |
| * | TABLE ACCESS FULL | T1 | 1 | 3000K | 3000K | 00:00:00.45 |

```
Predicate Information (identified by operation id):
```

```
-----
```

```
2 - filter(NVL("C_DDL",42)=42)
```

```
SQL> select
          ut.num_rows*us.density as card
        from
          user_tables ut
        ,user_tab_col_statistics us
      where
          ut.table_name = us.table_name
      and ut.table_name = 'T1'
      and us.column_name = 'C_DDL';
```

CARD

3000000

```
SQL> update t1
      set c_ddl = 25
      where
        mod(n1,100) = 0;

30000 rows updated.

SQL> commit;

SQL> begin
dbms_stats.gather_table_stats(user , 't1'
                               , method_opt => 'for all columns size 1'
                               , no_invalidate => false);
end;
/
SQL> select
      c_ddl
      ,count(1)
    from t1
   group by
      c_ddl;

C_DDL  COUNT(1)
-----
42      2970000
25      30000
```

```
SQL> select /*+ gather_plan_statistics */ count(1) from t1 where C_DDL = 42;
```

```
COUNT(1)
```

```
-----  
2970000
```

```
SQL> select * from table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

| Id | Operation | Name | Starts | E-Rows | A-Rows | A-Time |
|----|---------------------|------|--------|--------|--------|-------------|
| 0 | SELECT STATEMENT | | 1 | | 1 | 00:00:00.40 |
| 1 | SORT AGGREGATE | | 1 | 1 | 1 | 00:00:00.40 |
| * | 2 TABLE ACCESS FULL | T1 | 1 | 1500K | 2970K | 00:00:00.50 |

```
Predicate Information (identified by operation id):
```

```
-----  
2 - filter(NVL("C_DDL",42)=42)
```

```

SQL> alter table t1 add c_ddl_virt number
   generated always as (case when c_ddl = 42 then c_ddl else null end)
   virtual;

SQL> exec dbms_stats.gather_table_stats (user, 't1', method_opt => 'for columns c_ddl_virt size 1',
no_invalidate => false);
SQL> select /*+ gather_plan_statistics */ count(1) from t1 where c_ddl_virt = 42;

COUNT(1)
-----
2970000

SQL> select * from table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
-----  

| Id | Operation           | Name | Starts | E-Rows | A-Rows | A-Time      | |
|---|---|---|---|---|---|---|---|
|  0 | SELECT STATEMENT    |      |        |        |        | 00:00:00.85 |
|  1 |   SORT AGGREGATE    |      |        |        |        | 00:00:00.85 |
| * 2 |     TABLE ACCESS FULL| T1  |        | 1      | 2970K  | 2970K|00:00:00.70 |
-----  


```

Predicate Information (identified by operation id):

```

2 - filter(CASE NVL("C_DDL",42) WHEN 42 THEN NVL("C_DDL",42) ELSE NULL
           END =42)

```

```
SQL> select /*+ gather_plan_statistics */ count(1) from t1 where C_DDL = 42;
```

```
COUNT(1)
```

```
-----  
2970000
```

| Id | Operation | Name | Starts | E-Rows | A-Rows | A-Time |
|----|---------------------|------|--------|--------|--------|-------------|
| 0 | SELECT STATEMENT | | 1 | | 1 | 00:00:00.41 |
| 1 | SORT AGGREGATE | | 1 | 1 | 1 | 00:00:00.41 |
| * | 2 TABLE ACCESS FULL | T1 | 1 | 1500K | 2970K | 00:00:00.51 |

```
Predicate Information (identified by operation id):
```

```
2 - filter(NVL("C_DDL",42)=42)
```

```

SQL> select /*+ gather_plan_statistics */ count(1)
  from t1
 where C_DDL = 42
   and n2      = 22;

  COUNT(1)
-----
          3

SQL> select * from table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));

-----| Id  | Operation           | Name | Starts | E-Rows | A-Rows | A-Time   |
-----| 0   | SELECT STATEMENT    |       |        1 |        1 |        1 | 00:00:00.15 |
| 1   | SORT AGGREGATE      |       |        1 |        1 |        1 | 00:00:00.15 |
|*  2   | TABLE ACCESS FULL   | T1   |        1 |        1 |        3 | 00:00:00.15 |

-----| Predicate Information (identified by operation id): |
-----| 2 - filter(("N2"=22 AND NVL("C_DDL",42)=42)) |

```

```
SQL> SELECT
      dbms_stats.create_extended_stats
    (ownname=>user
     ,tabname=>'t1'
     ,extension=>'(c_ddl,n2)')
  FROM dual;

DBMS_STATS.CREATE_EXTENDED_STATS(OWNNAME)
-----
SYS_STU5FUD4#KCC03#_TZNTSTN5AV
```

```
SQL> begin
  dbms_stats.gather_table_stats
    (user
     , 't1'
     ,method_opt      => 'for columns SYS_STU5FUD4#KCC03#_TZNTSTN5AV size 1'
     ,cascade         => true
     ,no_invalidate  => false
    );
end;
/

```

```
SQL> select /*+ gather_plan_statistics */ count(1)
  from t1
 where C_DDL = 42
   and n2      = 22;
```

```
COUNT(1)
```

```
---
```

```
3
```

```
SQL> select * from table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

| Id | Operation | Name | Starts | E-Rows | A-Rows | A-Time |
|----|-------------------|------|--------|--------|--------|-------------|
| 0 | SELECT STATEMENT | | 1 | | 1 | 00:00:00.14 |
| 1 | SORT AGGREGATE | | 1 | 1 | 1 | 00:00:00.14 |
| * | TABLE ACCESS FULL | T1 | 1 | 3 | 3 | 00:00:00.14 |

```
Predicate Information (identified by operation id):
```

```
2 - filter(("N2"=22 AND NVL("C_DDL",42)=42))
```

```

SQL> select
      round (ut.num_rows*us.density) as card
    from
      user_tables ut
     ,user_tab_col_statistics us
   where
     ut.table_name  = us.table_name
   and ut.table_name  = 'T1'
   and us.column_name = 'SYS_STU5FUD4#KCC03#_TZNTSTN5AV';

CARD
-----
3

Access path analysis for T1
*****
SINGLE TABLE ACCESS PATH
  Single Table Cardinality Estimation for T1[T1]
SPD: Return code in qosdDSDirSetup: NODIR, estType = TABLE
  Column (#5): C_DDL(NUMBER)
    AvgLen: 3 NDV: 2 Nulls: 0 Density: 0.000000 Min: 0.000000 Max: 25.000000
  Column (#2): N2(NUMBER)
    AvgLen: 5 NDV: 1000000 Nulls: 0 Density: 0.000000 Min: 0.000000 Max: 17.000000
  Column (#6): SYS_STU5FUD4#KCC03#_TZNTSTN5AV(NUMBER)
    AvgLen: 12 NDV: 1030000 Nulls: 0 Density: 0.000000
  ColGroup (#1, VC) SYS_STU5FUD4#KCC03#_TZNTSTN5AV
    Col#: 2 5 CorStregh: 1.94
  ColGroup Usage:: PredCnt: 2 Matches Full: #1 Partial: Sel: 0.0000
Table: T1 Alias: T1
  Card: Original: 3000000.000000 Rounded: 3 Computed: 2.91 Non Adjusted: 2.91

```

```
card = t1(num_rows) / NDV(SYS_STU5FUD4#KCC03#_TZNTSTN5AV)
card = 3000000.000000 / 1030000 = 2,912621 rounded to 3
```

```
SQL> alter table t1 modify c_ddl default 0;
```

```
Elapsed: 00:00:00.03
```

```
SQL> select /*+ gather_plan_statistics */ count(1) from t1 where C_DDL = 42;  
COUNT(1)  
-----  
2970000  
  
SQL> select * from table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));  
  
-----  


| Id | Operation         | Name | Starts | E-Rows | A-Rows | A-Time      |
|----|-------------------|------|--------|--------|--------|-------------|
| 0  | SELECT STATEMENT  |      | 1      |        | 1      | 00:00:00.63 |
| 1  | SORT AGGREGATE    |      | 1      | 1      | 1      | 00:00:00.63 |
| *  | TABLE ACCESS FULL | T1   | 1      | 1500K  | 2970K  | 00:00:00.48 |

  
Predicate Information (identified by operation id):  
-----  
2 - filter(NVL("C_DDL",42)=42)
```

```
SQL> create table t2 as select
rownum n1
, trunc ((rownum -1)/3) n2
from dual
connect by level<=10;
SQL> alter table t2 add n3 number default 10;
```

SQL> select \* from t2;

| N1 | N2 | N3 |
|----|----|----|
| 1 | 0 | 10 |
| 2 | 0 | 10 |
| 3 | 0 | 10 |
| 4 | 1 | 10 |
| 5 | 1 | 10 |
| 6 | 1 | 10 |
| 7 | 2 | 10 |
| 8 | 2 | 10 |
| 9 | 2 | 10 |
| 10 | 3 | 10 |

```
SQL> alter table t2 modify n3 default 20;
```

```
SQL> select * from t2;
```

| N1 | N2 | N3 |
|----|----|----|
| 1 | 0 | 10 |
| 2 | 0 | 10 |
| 3 | 0 | 10 |
| 4 | 1 | 10 |
| 5 | 1 | 10 |
| 6 | 1 | 10 |
| 7 | 2 | 10 |
| 8 | 2 | 10 |
| 9 | 2 | 10 |
| 10 | 3 | 10 |

```
SQL> insert into t2(n1,n2) values (10,5);
```

```
SQL> select * from t2 where n1 = 10;
```

| N1 | N2 | N3 |
|----|----|----|
| 10 | 3 | 10 |
| 10 | 5 | 20 |

```
SQL> create index i1_c_ddl on t1(c_ddl);
```

Index created.

Elapsed: 00:00:02.14

```
SQL> select /*+ gather_plan_statistics */ count(1) from t1 where C_DDL = 42;
```

```
COUNT(1)
```

```
-----
```

```
2970000
```

```
SQL> select * from table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

| Id | Operation | Name | Starts | E-Rows | A-Rows | A-Time |
|----|----------------------|----------|--------|--------|--------|-------------|
| 0 | SELECT STATEMENT | | 1 | | 1 | 00:00:00.84 |
| 1 | SORT AGGREGATE | | 1 | 1 | 1 | 00:00:00.84 |
| * | INDEX FAST FULL SCAN | I1_C_DDL | 1 | 1500K | 2970K | 00:00:00.70 |

```
Predicate Information (identified by operation id):
```

```
-----
```

```
2 - filter("C_DDL"=42)
```

```
SQL>SELECT
          segment_name
        ,bytes/1024/1024 mb
  FROM
        dba_segments
 WHERE
          segment_type = 'INDEX'
 AND segment_name = 'I1_C_DDL';
```

| SEGMENT_NAME | MB |
|--------------|----|
| I1_C_DDL | 47 |

```

SQL> drop index i1_c_ddl;
Index dropped.

SQL> alter table t1 modify n1 not null;
Table altered.

SQL> create index i2_n1_c_ddl on t1(n1,c_ddl);
Index created.

SQL> select /*+ gather_plan_statistics */ count(1) from t1 where n1= 101 and C_DDL = 42;
COUNT(1)
-----
1
SQL> select * from table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));

-----  

| Id | Operation           | Name      | Starts | E-Rows | A-Rows |
|----|----|----|----|----|----|
| 0 | SELECT STATEMENT    |          | 1 |       | 1 |
| 1 |   SORT AGGREGATE    |          | 1 |       | 1 |
| * 2 |   INDEX RANGE SCAN | I2_N1_C_DDL | 1 |       | 1 |
-----  


```

Predicate Information (identified by operation id):

```

-----  

2 - access("N1"=101 AND "C_DDL"=42)

```

```

SQL> drop index i2_n1_c_ddl;

SQL> create index i2_n1_c_ddl on t1(n1);

SQL> select /*+ gather_plan_statistics */ count(1) from t1 where n1= 101 and C_DDL = 42;

SQL> select * from table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));

```

| Id | Operation | Name | Starts | E-Rows | A-Rows |
|-----|-----------------------------|-------------|--------|--------|--------|
| 0 | SELECT STATEMENT | | 1 | | 1 |
| 1 | SORT AGGREGATE | | 1 | 1 | 1 |
| * 2 | TABLE ACCESS BY INDEX ROWID | T1 | 1 | 1 | 1 |
| * 3 | INDEX RANGE SCAN | I2_N1_C_DDL | 1 | 1 | 1 |

Predicate Information (identified by operation id):

```

2 - filter(NVL("C_DDL",42)=42)
3 - access("N1"=101)

```

```
12c> alter table t1 add C_DDL number default 42 not null;
```

```
Elapsed: 00:00:00.02
```

```
12c> select count(1) from t1 where c_ddl=42;
```

```
COUNT(1)
```

```
-----  
3000000
```

```
12c> select * from table(dbms_xplan.display_cursor);
```

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|----|---------------------|------|-------|-------|-------------|----------|
| 0 | SELECT STATEMENT | | | | 3802 (100) | |
| 1 | SORT AGGREGATE | | 1 | 13 | | |
| * | 2 TABLE ACCESS FULL | T1 | 3538K | 43M | 3802 (1) | 00:00:01 |

```
Predicate Information (identified by operation id):
```

```
2 - filter(NVL("C_DDL",42)=42)
```

```
Note
```

```
- dynamic statistics used: dynamic sampling (level=2)
```

```
11.2.0.3.0> alter table t1 add C_DDL_2 number default 84;
```

Table altered.

Elapsed: 00:00:**58.25**

```
12c> alter table t1 add C_DDL_2 number default 84;
```

Elapsed: 00:00:00.**02**

```
12c before alter> SELECT
    segment_name
    ,bytes/1024/1024 mb
FROM
    dba_segments
WHERE
    segment_type = 'TABLE'
AND segment_name = 'T1';
```

| SEGMENT_NAME | MB |
|--------------|-----|
| T1 | 112 |

```
12c after alter> SELECT
    segment_name
    ,bytes/1024/1024 mb
FROM
    dba_segments
WHERE
    segment_type = 'TABLE'
AND segment_name = 'T1';
```

| SEGMENT_NAME | MB |
|--------------|-----|
| T1 | 112 |

```
12c> select c_ddl_2, count(1) from t1 group by c_ddl_2;
```

```
C_DDL_2 COUNT(1)
```

```
-----  
84      3000000
```

```
SQL> select count(1) from t1 where c_ddl_2=84;
```

```
-----  
COUNT(1)
```

```
-----  
3000000
```

```
SQL> select * from table(dbms_xplan.display_cursor);
```

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|----|---------------------|------|-------|-------|-------------|----------|
| 0 | SELECT STATEMENT | | | | 3803 (100) | |
| 1 | SORT AGGREGATE | | 1 | 13 | | |
| * | 2 TABLE ACCESS FULL | T1 | 3538K | 43M | 3803 (1) | 00:00:01 |

```
Predicate Information (identified by operation id):
```

```
-----  
2 - filter(DECODE(TO_CHAR(SYS_OP_VECBIT("SYS_NC00006$",0)),NULL,NVL("C_DDL_2",84),'0',NVL("C_DDL_2",84),'1',"C_DDL_2")=84)
```

```
Note
```

```
-----  
- dynamic statistics used: dynamic sampling (level=2)
```

```
12c> SELECT
          column_name
        ,virtual_column
        ,hidden_column
        ,user_generated
  FROM
        user_tab_cols
 WHERE
        table_name = 'T1'
 AND column_name = 'SYS_NC00006$';
```

| COLUMN_NAME | VIR | HID | USE |
|---------------|-----|-----|-----|
| ----- | - | - | - |
| SYS_NC00006\$ | NO | YES | NO |

```
12c> insert into t1(n1,n2,n3,c1,c_ddl,c_ddl_2) values (0,0,0,'xxxxx',110,130);
12c> insert into t1(n1,n2,n3,c1,c_ddl,c_ddl_2) values (1,1,1,'xxxxx',140,150);
12c> insert into t1(n1,n2,n3,c1,c_ddl, c_ddl_2) values (1,1,1,'xxxxx',200,null);

12c> select
      a.c_ddl_2
      ,a.SYS_NC00006$
  from t1 a
 where a.c_ddl_2 in (130,150);

C_DDL_2  SYS_NC00006$  
-----  
130 01  
150 01

SQL> select
      a.c_ddl_2
      ,a.SYS_NC00006$
  from t1 a
 where a.c_ddl_2 is null;

C_DDL_2  SYS_NC00006$  
-----  
01
```

Predicate Information (identified by operation id) :

```
2 - filter(DECODE(TO_CHAR(SYS_OP_VECBIT("SYS_NC00006$",0)),NULL,NVL("C_DDL_2",84),'0',NVL("C_DDL_2",84),'1',"C_DDL_2")=84)
```

```

12c> SELECT
      a.c_ddl_2
      ,TO_CHAR(sys_op_vecbit(a.sys_nc00006$,0)) cbo_ddl
   FROM t1 a
 WHERE a.c_ddl_2 IN (130,150) -- supplied values
UNION ALL
  SELECT
      a.c_ddl_2
      ,TO_CHAR(sys_op_vecbit(a.sys_nc00006$,0)) cbo_ddl
   FROM t1 a
 WHERE a.c_ddl_2 IS NULL          -- supplied value
UNION ALL
  SELECT
      a.c_ddl_2
      ,TO_CHAR(sys_op_vecbit(a.sys_nc00006$,0)) cbo_ddl
   FROM t1 a
 WHERE c_ddl_2 =84                -- default value
   AND rownum <=1
  order by c_ddl_2 nulls last;

```

| C_DDL_2 | CBO_DDL |
|---------|---------|
| 84 | {null} |
| 130 | 1 |
| 150 | 1 |
| {null} | 1 |

```
SQL> CREATE BIGFILE tablespace data_ts DATAFILE size 10G;
SQL> CREATE BIGFILE TABLESPACE data_ts DATAFILE '/oradata/data_ts01.dbf' SIZE 10G;
```

```
SQL> select name,bigfile from v$tablespace;
```

| NAME | BIG |
|----------|-----|
| SYSTEM | NO |
| UNDOTBS1 | NO |
| SYSAUX | NO |
| USERS | NO |
| TEMP | NO |
| DATA_TS | YES |

```
SQL> CREATE TABLE table_name (column_list...) ROW COMPRESS ADVANCED;
SQL> ALTER TABLE table_name ROW STORE COMPRESS BASIC;
SQL> ALTER TABLE table_name ROW STORE COMPRESS |ADVANCED;
SQL> ALTER TABLE table_name MOVE PARTITION partition1 ROW STORE COMPRESS BASIC | ADVANCE;
SQL> ALTER TABLE table_name NOCOMPRESS;
```

```
SQL> SELECT table_name,compression, compress_for FROM user_tables WHERE compression = 'ENABLED';
```

```
SQL> ALTER SESSION | SYSTEM SET HEAT_MAP = ON;
SQL> ALTER SESSION | SYSTEM SET HEAT_MAP = OFF; -- disable Heat Map
```

```
SQL> ALTER TABLE sales
      ILM ADD POLICY
      ROW STORE
      COMPRESS ADVANCED
      SEGMENT
      AFTER 60 DAYS OF NO MODIFICATION;
```

```
SQL> ALTER TABLE sales
      MODIFY PARTITION sales_dec14
      ILM ADD POLICY
      ROW STORE
      COMPRESS ADVANCED
      ROW
      AFTER 30 DAYS OF NO MODIFICATION;
```

```
SQL> ALTER TABLE sales ILM DISABLE_ALL;  
SQL> ALTER TABLE sales ILM DELETE_ALL;
```

```
SQL> SELECT policy_name, policy_type, enabled FROM user_ilm_policies;
```

```
SQL> CREATE INDEX index_name ON(column) COMPRESS ADVANCED HIGH LOCAL (PARTITION
      ip1 COMPRESS ADVANCED LOW, PARTITION ip2 COMPRESS HIGH, PARTITION ip3 NOCOMPRESS);
SQL> CREATE INDEX index_name ON(column) LOCAL (PARTITION ip1 COMPRESS ADVANCED LOW,
      PARTITION ip2 COMPRESS HIGH, PARTITION ip3);
SQL> CREATE INDEX index_name ON(columns_list..) COMPRESS ADVANCED LOW;
```

```
SQL> CREATE TABLE emp (eno number(9),ename varchar2(100),dob date, dept number(2), salary
      number(6)) INDEXING OFF
PARTITION BY RANGE (salary)
(PARTITION p10000 values less than (10001) INDEXING OFF,
partition p20000 values less than (20001) INDEXING ON,
partition p30000 values less than (30001));

SQL> CREATE INDEX index_p1 ON emp(eno) GLOBAL INDEXING FULL|PARTIAL;
SQL> CREATE INDEX index_p2 ON emp(salary) LOCAL INDEXING PARTIAL;
SQL> ALTER TABLE emp MODIFY PARTITION p10000 INDEXING ON;
```

| | | |
|---------------------------------|---------|--------|
| parallel_degree_policy | string | MANUAL |
| parallel_execution_message_size | integer | 16384 |
| parallel_max_servers | integer | 240 |
| parallel_min_percent | integer | 0 |
| parallel_min_servers | integer | 0 |
| parallel_server | boolean | TRUE |
| parallel_servers_target | integer | 96 |
| parallel_threads_per_cpu | integer | 2 |

```
SQL> exec dbms_stats.set_database_prefs('INCREMENTAL_STALENESS', 'USE_STALE_PERCENTAGE');
SQL> exec dbms_stats.set_database_prefs('STALE_PERCENT', '25');
```

```
SQL> exec dbms_stats.get_prefs('STALE_PERCENT', 'SCHEMA', 'TABLENAME') STALE_VALUE FROM dual;
```

```
SQL> SELECT dbms_stats.get_prefs('INCREMENTAL', 'SCHEMA', 'TABLENAME')
      tab_incr_prefs FROM dual;
```

```
TAB_INCR_PREFS
```

```
-----  
FALSE
```

```
SQL> exec dbms_stats.gather_table_stasts('SCHEMA',
   'TABLENAME',ESTMATE_PERCENT=>dbms_stats.auto_sample_size);

SELECT dbms_stats.get_prefs('INCREMENTAL', 'SCHEMA', 'TABLENAME')
  tab_inc_perf FROM dual;
```

```
SQL> exec dbms_stats.set_table_prefs('SCHEMA', 'TABLENAME', 'INCREMENTAL', 'TRUE');

TAB_INC_PERF
-----
TRUE
```

```
SQL> SELECT owner,    object_name
  FROM dba_objects
 WHERE object_id IN
   (SELECT DISTINCT(obj#)
    FROM optstat_user_prefs$
   WHERE PNAME= 'INCREMENTAL'
   AND VARCHAR= 'TRUE' ) ;
```

```
SQL> exec dbms_stats.set_SCHEMA_prefs('SCHEMA', 'INCREMENTAL', 'FALSE');
```

```
SQL> exec dbms_stats.set_table_prefs('SCHEMA', 'TABLENAME', 'INCREMENTAL', 'FALSE');
```

```
SQL> exec dbms_stats.set_schema_prefs('SCHEMA', 'INCREMENTAL', 'FALSE');
```

```
SQL> exec dbms_stats.set_global_prefs('CONCURRENT', 'TRUE');
SQL> SELECT dbms_stats.get_prefs('CONCURRENT') FROM dual;
      DBMS_STATS.GET_PREFS('CONCURRENT')
-----
TRUE
```

```
SQL> SELECT owner,job_name,state,start_date,max_run_duration
  FROM dba_scheduler_jobs
 WHERE job_class like 'CONCURRENT%' AND state in ('RUNNING', 'SCHEDULED');

SQL> SELECT job_name, state, comments
  FROM dba_scheduler_jobs
 WHERE job_class LIKE 'CONC%' and STATE = 'RUNNING';

SQL> SELECT job_name, elapsed_time
  FROM dba_scheduler_running_jobs WHERE job_name LIKE 'ST$%';
```

```
SQL> exec dbms_stats.gather_table_stasts(null , 'tablename' ,
estimate_percent=>dbms_stats.auto_sample_size);
```

```
RMAN> run
{
    allocate channel ch1 type 'SBT_TAPE';
    allocate channel ch2 type 'SBT_TAPE';
    BACKUP INCREMENTAL LEVEL=0
        FORMAT 'bk_u%u_s%s_p%p_t%t'
        DATABASE PLUS ARCHIVELOG;
    release channel ch1;
    release channel ch2;
}
```

```
--- RMAN binary compressed full database online backups
RMAN> run {
    BACKUP
    AS COMPRESSED BACKUPSET
    DATABASE PLUS ARCHIVELOG;
}
```

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

```
SQL> ALTER DATABASE BLOCK CHANGE TRACKING USING FILE '<location/filename>' ;
```

```
# RMAN daily cumulative incremental database online backups
RMAN> run
{
    allocate channel ch1 type 'SBT_TAPE';
    BACKUP
        INCREMENTAL LEVEL=1 CUMULATIVE
        FORMAT 'bk_u%u_s%s_p%p_t%t'
        DATABASE PLUS ARCHIVELOG;
    release channel ch1;
}
```

```
RMAN> run
{
  RECOVER COPY OF DATABASE WITH TAG 'weekly_iud';
  BACKUP
    INCREMENTAL LEVEL 1
    FOR RECOVER OF COPY WITH TAG 'weekly_iud'
    DATABASE
    PLUS ARCHIVELOG;
}
```

```

#####
# Results from 1st run:
# - Incremental Level 0 image copies of all datafiles now exist
#   because even though a Level 1 incremental backup was requested,
#   no Level 0 incremental backup has yet been created with the tag
#   of img_cpy_upd.
#####

RMAN> # Implement Incrementally-Updateable Image Copy (IUIC) strategy
2> # for RPO of 24 hours (i.e., RECOVERY WINDOW OF ONE DAY)
3> RUN {
4>   RECOVER
5>     COPY OF DATABASE
6>     WITH TAG 'img_cpy_upd';
7>   BACKUP
8>     INCREMENTAL LEVEL 1
9>     FOR RECOVER OF COPY WITH TAG 'img_cpy_upd'
10>    SECTION SIZE 100M
11>    DATABASE
12>    PLUS ARCHIVELOG DELETE INPUT;
13> }

Starting recover at 2014-04-07 13:17:07
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=496 device type=DISK
allocated channel: ORA_DISK_2
channel ORA_DISK_2: SID=13 device type=DISK
allocated channel: ORA_DISK_3
channel ORA_DISK_3: SID=38 device type=DISK
allocated channel: ORA_DISK_4
channel ORA_DISK_4: SID=486 device type=DISK
no copy of datafile 1 found to recover
no copy of datafile 2 found to recover
no copy of datafile 3 found to recover
no copy of datafile 4 found to recover
no copy of datafile 5 found to recover
no copy of datafile 6 found to recover
no copy of datafile 7 found to recover
no copy of datafile 8 found to recover
no copy of datafile 9 found to recover
no copy of datafile 10 found to recover
no copy of datafile 11 found to recover
no copy of datafile 12 found to recover
no copy of datafile 13 found to recover
no copy of datafile 14 found to recover
no copy of datafile 15 found to recover
Finished recover at 2014-04-07 13:17:08

Starting backup at 2014-04-07 13:17:09
current log archived
using channel ORA_DISK_1
using channel ORA_DISK_2
using channel ORA_DISK_3
using channel ORA_DISK_4
channel ORA_DISK_1: starting archived log backup set
channel ORA_DISK_1: specifying archived log(s) in backup set
input archived log thread=1 sequence=508 RECID=9 STAMP=844262229
channel ORA_DISK_1: starting piece 1 at 2014-04-07 13:17:09
channel ORA_DISK_1: finished piece 1 at 2014-04-07 13:17:10
piece handle=+FRA/NCDB121/BACKUPSET/2014_04_07/annnf0_img_cpy_upd_0.307.844262231
  tag=IMG_CPY_UPD comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:02
channel ORA_DISK_1: deleting archived log(s)
archived log file name=+FRA/NCDB121/ARCHIVELOG/2014_04_07/thread_1_seq_508.272.844262229
  RECID=9 STAMP=844262229
Finished backup at 2014-04-07 13:17:11

```

```
Starting backup at 2014-04-07 13:17:11
using channel ORA_DISK_1
using channel ORA_DISK_2
using channel ORA_DISK_3
using channel ORA_DISK_4
no parent backup or copy of datafile 3 found
no parent backup or copy of datafile 1 found
no parent backup or copy of datafile 4 found
no parent backup or copy of datafile 12 found
no parent backup or copy of datafile 2 found
no parent backup or copy of datafile 10 found
no parent backup or copy of datafile 14 found
no parent backup or copy of datafile 15 found
no parent backup or copy of datafile 8 found
no parent backup or copy of datafile 9 found
no parent backup or copy of datafile 11 found
no parent backup or copy of datafile 13 found
no parent backup or copy of datafile 5 found
no parent backup or copy of datafile 7 found
no parent backup or copy of datafile 6 found
channel ORA_DISK_1: starting datafile copy
input datafile file number=00003 name=+DATA/NCDB121/DATAFILE/sysaux.283.836524477
channel ORA_DISK_2: starting datafile copy
input datafile file number=00001 name=+DATA/NCDB121/DATAFILE/system.282.836524551
channel ORA_DISK_3: starting datafile copy
input datafile file number=00004 name=+DATA/NCDB121/DATAFILE/undotbs1.284.836524619
channel ORA_DISK_4: starting datafile copy
input datafile file number=00012 name=+FRA/NCDB121/DATAFILE/ado_cold_data.283.841267371
output file name=+FRA/NCDB121/DATAFILE/ado_cold_data.274.844262235
    tag=IMG_CPY_UPD RECID=55 STAMP=844262264
channel ORA_DISK_4: datafile copy complete, elapsed time: 00:00:35
channel ORA_DISK_4: starting datafile copy
input datafile file number=00002 name=+DATA/NCDB121/DATAFILE/example.280.836524747
output file name=+FRA/NCDB121/DATAFILE/undotbs1.303.844262235
    tag=IMG_CPY_UPD RECID=56 STAMP=844262271
channel ORA_DISK_3: datafile copy complete, elapsed time: 00:00:41
```

```

...
{ omitted for sake of brevity }

piece handle=+FRA/NCDB121/BACKUPSET/2014_04_07/nnsnn1_img_cpy_upd_0.295.844262321
  tag=IMG_CPY_UPD comment=NONE
channel ORA_DISK_4: backup set complete, elapsed time: 00:00:02
Finished backup at 2014-04-07 13:18:42

Starting backup at 2014-04-07 13:18:42
current log archived
using channel ORA_DISK_1
using channel ORA_DISK_2
using channel ORA_DISK_3
using channel ORA_DISK_4
channel ORA_DISK_1: starting archived log backup set
channel ORA_DISK_1: specifying archived log(s) in backup set
input archived log thread=1 sequence=509 RECID=10 STAMP=844262322
channel ORA_DISK_1: starting piece 1 at 2014-04-07 13:18:43
channel ORA_DISK_1: finished piece 1 at 2014-04-07 13:18:44
piece handle=+FRA/NCDB121/BACKUPSET/2014_04_07/annnf0_img_cpy_upd_0.304.844262323
  tag=IMG_CPY_UPD comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:01
channel ORA_DISK_1: deleting archived log(s)
archived log file name=+FRA/NCDB121/ARCHIVELOG/2014_04_07/thread_1_seq_509.271.844262323
  RECID=10 STAMP=844262322
Finished backup at 2014-04-07 13:18:45

#####
# Results from 2nd run:
# - Two new tablespaces have been created to show how they will be wrapped
#   into the IUIC strategy automatically (as Incremental Level 0 image
#   copy backups).
# - Incremental Level 0 image copies of all datafiles now exist, but since
#   no Incremental Level 1 image copy backups yet exist with a tag of
#   img_cpy_upd, none will be applied to the existing Level 0 backups.
# - Incremental Level 1 backups of all datafiles will be taken because
#   Incremental Level 0 backups (their eventual "parents") now exist.
#####

```

```

# Create new tablespaces:
CREATE TABLESPACE rollon_staging_data
  DATAFILE '+DATA'
  SIZE 200M
  NOLOGGING;

CREATE TABLESPACE rollon_staging_idx
  DATAFILE '+DATA'
  SIZE 100M
  NOLOGGING;

RMAN> # Implement Incrementally-Updateable Image Copy (IUIC) strategy
2> # for RPO of 24 hours (i.e., RECOVERY WINDOW OF ONE DAY)
3> RUN {
4>   RECOVER
5>     COPY OF DATABASE
6>     WITH TAG 'img_cpy_upd';
7>   BACKUP
8>     INCREMENTAL LEVEL 1
9>     FOR RECOVER OF COPY WITH TAG 'img_cpy_upd'
10>    SECTION SIZE 100M
11>    DATABASE
12>    PLUS ARCHIVELOG DELETE INPUT;
13> }

Starting backup at 2014-04-07 13:29:50
using channel ORA_DISK_1
using channel ORA_DISK_2
using channel ORA_DISK_3
using channel ORA_DISK_4
no parent backup or copy of datafile 16 found
no parent backup or copy of datafile 17 found
channel ORA_DISK_1: starting incremental level 1 datafile backup set
channel ORA_DISK_1: specifying datafile(s) in backup set
input datafile file number=00003 name=+DATA/NCDB121/DATFILE/sysaux.283.836524477
backing up blocks 1 through 12800
channel ORA_DISK_1: starting piece 1 at 2014-04-07 13:29:52
channel ORA_DISK_2: starting incremental level 1 datafile backup set
channel ORA_DISK_2: specifying datafile(s) in backup set
input datafile file number=00001 name=+DATA/NCDB121/DATFILE/system.282.836524551
backing up blocks 1 through 12800
channel ORA_DISK_2: starting piece 1 at 2014-04-07 13:29:53
channel ORA_DISK_3: starting incremental level 1 datafile backup set
channel ORA_DISK_3: specifying datafile(s) in backup set
input datafile file number=00004 name=+DATA/NCDB121/DATFILE/undotbs1.284.836524619
backing up blocks 1 through 12800
channel ORA_DISK_3: starting piece 1 at 2014-04-07 13:29:53
channel ORA_DISK_4: starting incremental level 1 datafile backup set
channel ORA_DISK_4: specifying datafile(s) in backup set
input datafile file number=00012 name=+FRA/NCDB121/DATFILE/ado_cold_data.283.841267371
backing up blocks 1 through 12800
channel ORA_DISK_4: starting piece 1 at 2014-04-07 13:29:55
channel ORA_DISK_1: finished piece 1 at 2014-04-07 13:29:55
piece
...
{ omitted for sake of brevity }
...
handle=+FRA/NCDB121/BACKUPSET/2014_04_07/nnndn1_img_cpy_upd_0.279.844263037
  tag=IMG_CPY_UPD comment=NONE
channel ORA_DISK_2: backup set complete, elapsed time: 00:00:01
channel ORA_DISK_4: finished piece 1 at 2014-04-07 13:30:38
piece handle=+FRA/NCDB121/BACKUPSET/2014_04_07/ncnnn1_img_cpy_upd_0.356.844263037
  tag=IMG_CPY_UPD comment=NONE
channel ORA_DISK_4: backup set complete, elapsed time: 00:00:01
Finished backup at 2014-04-07 13:30:38

```

```

Starting backup at 2014-04-07 13:30:39
current log archived
using channel ORA_DISK_1
using channel ORA_DISK_2
using channel ORA_DISK_3
using channel ORA_DISK_4
channel ORA_DISK_1: starting archived log backup set
channel ORA_DISK_1: specifying archived log(s) in backup set
input archived log thread=1 sequence=515 RECID=16 STAMP=844263040
channel ORA_DISK_1: starting piece 1 at 2014-04-07 13:30:40
channel ORA_DISK_1: finished piece 1 at 2014-04-07 13:30:41
piece handle=+FRA/NCDB121/BACKUPSET/2014_04_07/annnf0_img_cpy_upd_0.359.844263041
tag=IMG_CPY_UPD comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:01
channel ORA_DISK_1: deleting archived log(s)
archived log file name=+FRA/NCDB121/ARCHIVELOG/2014_04_07/thread_1_seq_515.358.844263039
RECID=16 STAMP=844263040
Finished backup at 2014-04-07 13:30:43

```

```

#####
# Results from 3rd run:
# - The Incremental Level 1 backups from the 1st night's backups will now
#   be applied to their Incremental Level 0 image copy "parents."
# - Incremental Level 1 backups of any changed datafiles will be taken.
# - The Incremental Level 0 image copy backup for the two new datafiles
#   created during the 2nd night's run will not yet be updated from
#   Incremental Level 1 backups until the next night's run.
#####

```

```

RMAN> # Implement Incrementally-Updateable Image Copy (IUIC) strategy
2> # for RPO of 24 hours (i.e., RECOVERY WINDOW OF ONE DAY)
3> RUN {
4>   RECOVER
5>   COPY OF DATABASE
6>   WITH TAG 'img_cpy_upd';
7>   BACKUP
8>   INCREMENTAL LEVEL 1
9>   FOR RECOVER OF COPY WITH TAG 'img_cpy_upd'
10>  SECTION SIZE 100M
11>  DATABASE
12>  PLUS ARCHIVELOG DELETE INPUT;
13> }
13> }

Starting recover at 2014-04-07 14:27:31
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=496 device type=DISK
allocated channel: ORA_DISK_2
channel ORA_DISK_2: SID=13 device type=DISK
allocated channel: ORA_DISK_3
channel ORA_DISK_3: SID=38 device type=DISK
allocated channel: ORA_DISK_4
channel ORA_DISK_4: SID=486 device type=DISK
no copy of datafile 16 found to recover
no copy of datafile 17 found to recover
channel ORA_DISK_1: starting incremental datafile backup set restore
channel ORA_DISK_1: specifying datafile copies to recover
recovering datafile copy file number=00003 name=+FRA/NCDB121/DATAFILE/sysaux.353.844262767
channel ORA_DISK_1: restoring section 1 of 15
channel ORA_DISK_1: reading from backup piece
+FRA/NCDB121/BACKUPSET/2014_04_07/nnndn1_img_cpy_upd_0.338.844262993

```

```

channel ORA_DISK_2: starting incremental datafile backup set restore
channel ORA_DISK_2: specifying datafile copies to recover
recovering datafile copy file number=00001 name=+FRA/NCDB121/DATAFILE/system.357.844262767
channel ORA_DISK_2: restoring section 1 of 8
channel ORA_DISK_2: reading from backup piece
  +FRA/NCDB121/BACKUPSET/2014_04_07/nnndn1_img_cpy_upd_0.267.844262993
channel ORA_DISK_3: starting incremental datafile backup set restore
channel ORA_DISK_3: specifying datafile copies to recover
recovering datafile copy file
  number=00004 name=+FRA/NCDB121/DATAFILE/undotbs1.354.844262767
channel ORA_DISK_3: restoring section 1 of 6
channel ORA_DISK_3: reading from backup piece
  +FRA/NCDB121/BACKUPSET/2014_04_07/nnndn1_img_cpy_upd_0.268.844262993
...
{ omitted for sake of brevity }
...
{ omitted for sake of brevity }
...
channel ORA_DISK_4: piece handle=
  +FRA/NCDB121/BACKUPSET/2014_04_07/nnndn1_img_cpy_upd_0.303.844263035 tag=IMG_CPY_UPD
channel ORA_DISK_4: restored backup piece 1
channel ORA_DISK_4: restore complete, elapsed time: 00:00:01
channel ORA_DISK_2: piece handle=
  +FRA/NCDB121/BACKUPSET/2014_04_07/nnndn1_img_cpy_upd_0.279.844263037 tag=IMG_CPY_UPD
channel ORA_DISK_2: restored backup piece 1
channel ORA_DISK_2: restore complete, elapsed time: 00:00:00
Finished recover at 2014-04-07 14:27:49

Starting backup at 2014-04-07 14:27:50
current log archived
using channel ORA_DISK_1
using channel ORA_DISK_2
using channel ORA_DISK_3
using channel ORA_DISK_4
channel ORA_DISK_1: starting archived log backup set
channel ORA_DISK_1: specifying archived log(s) in backup set
input archived log thread=1 sequence=516 RECID=17 STAMP=844266471
channel ORA_DISK_1: starting piece 1 at 2014-04-07 14:27:51
channel ORA_DISK_1: finished piece 1 at 2014-04-07 14:27:52
piece handle=+FRA/NCDB121/BACKUPSET/2014_04_07/annnf0_img_cpy_upd_0.360.844266471
  tag=IMG_CPY_UPD comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:01
channel ORA_DISK_1: deleting archived log(s)
archived log file name=
  +FRA/NCDB121/ARCHIVELOG/2014_04_07/thread_1_seq_516.358.844266471
  RECID=17 STAMP=844266471
Finished backup at 2014-04-07 14:27:52

Starting backup at 2014-04-07 14:27:52
using channel ORA_DISK_1
using channel ORA_DISK_2
using channel ORA_DISK_3
using channel ORA_DISK_4
no parent backup or copy of datafile 3 found
no parent backup or copy of datafile 1 found
no parent backup or copy of datafile 4 found
no parent backup or copy of datafile 12 found
no parent backup or copy of datafile 2 found

```

```

no parent backup or copy of datafile 10 found
no parent backup or copy of datafile 14 found
no parent backup or copy of datafile 15 found
no parent backup or copy of datafile 8 found
no parent backup or copy of datafile 9 found
no parent backup or copy of datafile 11 found
no parent backup or copy of datafile 13 found
channel ORA_DISK_1: starting datafile copy
input datafile file number=00003 name=+DATA/NCDB121/DATAFILE/sysaux.283.836524477
channel ORA_DISK_2: starting datafile copy
input datafile file number=00001 name=+DATA/NCDB121/DATAFILE/system.282.836524551
channel ORA_DISK_3: starting datafile copy
input datafile file number=00004 name=+DATA/NCDB121/DATAFILE/undotbs1.284.836524619
channel ORA_DISK_4: starting datafile copy
input datafile file number=00012 name=+FRA/NCDB121/DATAFILE/ado_cold_data.283.841267371
output file name=+FRA/NCDB121/DATAFILE/ado_cold_data.363.844266475
tag=IMG_CPY_UPD RECID=114 STAMP=844266520
channel ORA_DISK_4: datafile copy complete, elapsed time: 00:00:56
...
{ omitted for sake of brevity }
...
{ omitted for sake of brevity }
...
handle=+FRA/NCDB121/BACKUPSET/2014_04_07/ncnnn1_img_cpy_upd_0.293.844266629 tag=IMG_CPY_UPD
comment=NONE
channel ORA_DISK_2: backup set complete, elapsed time: 00:00:01
Finished backup at 2014-04-07 14:30:31

Starting backup at 2014-04-07 14:30:31
current log archived
using channel ORA_DISK_1
using channel ORA_DISK_2
using channel ORA_DISK_3
using channel ORA_DISK_4
channel ORA_DISK_1: starting archived log backup set
channel ORA_DISK_1: specifying archived log(s) in backup set
input archived log thread=1 sequence=517 RECID=18 STAMP=844266631
channel ORA_DISK_1: starting piece 1 at 2014-04-07 14:30:31
channel ORA_DISK_1: finished piece 1 at 2014-04-07 14:30:32
piece handle=+FRA/NCDB121/BACKUPSET/2014_04_07/annnf0_img_cpy_upd_0.301.844266633 tag=IMG_CPY_UPD
comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:01
channel ORA_DISK_1: deleting archived log(s)
archived log file name=+FRA/NCDB121/ARCHIVELOG/2014_04_07/thread_1_seq_517.270.844266631
RECID=18 STAMP=844266631
Finished backup at 2014-04-07 14:30:33

```

```
RMAN> run
{
  allocate channel ch1 type 'SBT_TAPE';
  VALIDATE BACKUPSET <backupset_number>;
  release channel ch1;
}
```

```
RMAN> run {
      BACKUP VALIDATE
      CHECK LOGICAL
      DATABASE AND ARCHIVELOG ALL;
}
```

```
RMAN> BACKUP SECTION SIZE 2G TABLESPACE <tablespace_name>;
```

```
RMAN> CONFIGURE CHANNEL DEVICE TYPE DISK DEBUG=5 TRACE 1;
```

```

SQL> select
  ksppinm,
  ksppdesc
from
  x$ksppi
where
  substr(ksppinm,1,5) = '_back'
order by
  1,2;

```

| KSPPINM | KSPPDESC |
|--------------------------|--|
| _backup_align_write_io | align backup write I/Os |
| _backup_disk_bufcnt | number of buffers used for DISK channels |
| _backup_disk_bufsz | size of buffers used for DISK channels |
| _backup_disk_io_slaves | BACKUP Disk I/O slaves |
| _backup_dynamic_buffers | dynamically compute backup/restore buffer sizes |
| _backup_encrypt_opt_mode | specifies encryption block optimization mode |
| _backup_file_bufcnt | number of buffers used for file access |
| _backup_file_bufsz | size of buffers used for file access |
| _backup_io_pool_size | memory to reserve from the large pool |
| _backup_kgc_blk siz | specifies buffer size to be used by HIGH compression |
| _backup_kgc_bufsz | specifies buffer size to be used by BASIC compression |
| _backup_kgc_memlevel | specifies memory level for MEDIUM compression |
| _backup_kgc_niters | specifies number of iterations done by BASIC compression |
| _backup_kgc_perflevel | specifies compression (performance) level for MEDIUM compression |
| _backup_kgc_scheme | specifies compression scheme |
| _backup_kgc_type | specifies compression type used by kgc BASIC compression |
| _backup_kgc_windowbits | specifies window size for MEDIUM compression |
| _backup_ksfq_bufcnt | number of buffers used for backup/restore |
| _backup_ksfq_bufcnt_max | maximum number of buffers used for backup/restore |
| _backup_ksfq_bufsz | size of buffers used for backup/restore |
| _backup_lzo_size | specifies buffer size for LOW compression |
| _backup_max_gap_size | largest gap in an incremental/optimized backup buffer, in bytes |
| _backup_seq_bufcnt | number of buffers used for non-DISK channels |
| _backup_seq_bufsz | size of buffers used for non-DISK channels |

```
RMAN> configure device type [sbt|disk] parallelism 2;
```

```
$ srvctl add service -d RDB -s rdb_main -r rdb1,rdb2  
$ srvctl start service -d RDB -s rdb_main
```

```
RDB_MAIN =
(DESCRIPTION =
(ADDRESS_LIST =
(ADDRESS = (PROTOCOL = TCP)(HOST = rdbscan)(PORT = 1522))
)
(CONNECT_DATA =
(SERVICE_NAME = RDB_MAIN)
)
)
```

```
RMAN> run
{
    allocate channel ch1 connect 'sys/password@rdb_main';
    backup database;
}
```

```
RDB1 =
(DESCRIPTION =
(ADDRESS_LIST =
(ADDRESS = (PROTOCOL = TCP)(HOST = prdn1)(PORT = 1522))
)
(CONNECT_DATA =
(SERVICE_NAME = RDB)
)
)

RDB2 =
(DESCRIPTION =
(ADDRESS_LIST =
(ADDRESS = (PROTOCOL = TCP)(HOST = prdn22)(PORT = 1522))
)
(CONNECT_DATA =
(SERVICE_NAME = RDB)
)
)
```

```
RMAN> run
{
    allocate channel ch1 connect 'sys/password@rdb1';
    allocate channel ch2 connect 'sys/password@rdb2';
    backup database;
}
```

```
RMAN> configure snapshot controlfile name to '+DG_FRA\snapcf_RDB';
```

```
RMAN> RESTORE DATABASE VALIDATE;  
RMAN> RESTORE ARCHIVELOG ALL VALIDATE;
```

```
RMAN> CONFIGURE DEVICE TYPE DISK PARALLELISM 8;
```

```
RMAN> recover database parallel 8;
```

```
SQL> SELECT name,item,sofar,total FROM v$recovery_progress;
```

RMAN-00571: ======
RMAN-00569: ====== ERROR MESSAGE STACK FOLLOWS ======
RMAN-00571: ======
RMAN-03002: failure of list command at 04/13/2015 13:58:43
RMAN-05533: LIST FAILURE is not supported on RAC database

```
BEGIN  
    DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS(  
        retention => 44640, interval => 15, topnsql => 50,  
        dbid => 6611949349);  
END;  
/
```

WORKLOAD REPOSITORY report for

| DB Name | DB Id | Instance | Inst Num | Startup Time | Release | RAC |
|---------|------------|----------|----------|-----------------|------------|-----|
| AULTDB | 4030696936 | aultdb1 | | 04-Aug-08 10:16 | 11.1.0.6.0 | YES |

| Host Name | Platform | CPUs | Cores | Sockets | Memory (GB) |
|------------|-------------------|------|-------|---------|-------------|
| aultlinux3 | Linux IA (32-bit) | 2 | 1 | 1 | 2.97 |

| | Snap Id | Snap Time | Sessions | Curs/Sess |
|-------------|---------|--------------------|----------|-----------|
| Begin Snap: | 91 | 04-Aug-08 12:00:15 | 41 | 1.2 |
| End Snap: | 92 | 04-Aug-08 13:00:28 | 47 | 1.1 |
| Elapsed: | | 60.22 (mins) | | |
| DB Time: | | 139.52 (mins) | | |

| Cache Sizes | Begin | End | |
|-------------------|--------|--------|---------------------|
| Buffer Cache: | 1,312M | 1,312M | Std Block Size: 8K |
| Shared Pool Size: | 224M | 224M | Log Buffer: 10,604K |

| Load Profile | Per Second | Per Transaction | Per Exec | Per Call |
|--------------------------|------------------|--------------------|----------|----------|
| DB Time(s): | 2.3 | 7.1 | 0.63 | 1.05 |
| DB CPU(s): | 0.3 | 0.9 | 0.07 | 0.13 |
| Redo size: | 800.5 | 2,461.8 | | |
| Logical reads: | 6,307.6 | 19,396.7 | | |
| Block changes: | 3.6 | 10.9 | | |
| Physical reads: | 2,704.9 | 8,317.8 | | |
| Physical writes: | 86.9 | 267.3 | | |
| User calls: | 2.2 | 6.8 | | |
| Parses: | 2.0 | 6.1 | | |
| Hard parses: | 0.0 | 0.1 | | |
| W/A MB processed: | 932,965.4 | 2,868,990.9 | | |
| Logons: | 0.1 | 0.2 | | |
| Executes: | 3.7 | 11.3 | | |
| Rollbacks: | 0.1 | 0.3 | | |
| Transactions: | 0.3 | | | |

Instance Efficiency Percentages (Target 100%)

| | | | |
|------------------------------|--------|-------------------|--------|
| Buffer Nowait %: | 100.00 | Redo NoWait %: | 99.97 |
| Buffer Hit %: | 96.09 | In-memory Sort %: | 100.00 |
| Library Hit %: | 98.17 | Soft Parse %: | 97.88 |
| Execute to Parse %: | 45.80 | Latch Hit %: | 99.95 |
| Parse CPU to Parse Elapsd %: | 0.00 | % Non-Parse CPU: | 99.77 |

| Shared Pool Statistics | Begin | End |
|-----------------------------|-------|-------|
| Memory Usage %: | 81.53 | 85.39 |
| % SQL with executions>1: | 79.29 | 79.48 |
| % Memory for SQL w/execs>1: | 76.73 | 78.19 |

Top 5 Timed Foreground Events

| Event | Waits | Time (s) | Avg | % DB | Time | Wait Class |
|--|---------|------------|-----------|-------------|----------|------------|
| | | | Wait (ms) | | | |
| db file sequential read | 465,020 | 3,969 | 9 | 47.4 | User I/O | |
| DB CPU | | 995 | | 11.9 | | |
| db file parallel read | 2,251 | 322 | 143 | 3.8 | User I/O | |
| db file scattered read | 15,268 | 153 | 10 | 1.8 | User I/O | |
| gc current block 2-way | 108,739 | 116 | 1 | 1.4 | Cluster | |
| Host CPU (CPUs: 2 Cores: 1 Sockets: 1) | | | | | | |
| <hr/> | | | | | | |
| Load Average | | | | | | |
| Begin | End | %User | %System | %WIO | %Idle | |
| ----- | ----- | ----- | ----- | ----- | ----- | |
| 0.37 | 3.05 | 10.6 | 6.7 | 45.3 | 82.6 | |

Instance CPU

% of total CPU for Instance: 14.8
% of busy CPU for Instance: 85.0
%DB time waiting for CPU - Resource Mgr: 0.0

Memory Statistics

| | Begin | End |
|------------------------------|---------|---------|
| Host Mem (MB): | 3,041.4 | 3,041.4 |
| SGA use (MB): | 1,584.0 | 1,584.0 |
| PGA use (MB): | 169.0 | 301.7 |
| % Host Mem used for SGA+PGA: | 57.64 | 57.64 |

| Event | Waits | Time (s) | Avg | % DB | Time | Wait | Class |
|-------------------------|---------|----------|-----------|------|------|------|-------|
| | | | Wait (ms) | | | | |
| db file sequential read | 465,020 | 3,969 | 9 | 47.4 | User | I/O | |
| db file parallel read | 2,251 | 322 | 143 | 3.8 | User | I/O | |
| db file scattered read | 15,268 | 153 | 10 | 1.8 | User | I/O | |

Host CPU (CPUs: 2 Cores: 1 Sockets: 1)

| Load Average | | | | | |
|--------------|------|-------|---------|------|-------|
| Begin | End | %User | %System | %WIO | %Idle |
| 0.37 | 3.05 | 10.6 | 6.7 | 45.3 | 82.6 |

Instance CPU

| | |
|--|------|
| % of total CPU for Instance: | 14.8 |
| % of busy CPU for Instance: | 85.0 |
| %DB time waiting for CPU - Resource Mgr: | 0.0 |

Memory Statistics

| | Begin | End |
|------------------------------|---------|---------|
| Host Mem (MB) : | 3,041.4 | 3,041.4 |
| SGA use (MB) : | 1,584.0 | 1,584.0 |
| PGA use (MB) : | 169.0 | 301.7 |
| % Host Mem used for SGA+PGA: | 57.64 | 57.64 |

RAC Statistics DB/Inst: AULTDB/aultdb1 Snaps: 91-92

| | Begin | End |
|----------------------|-------|-----|
| Number of Instances: | 2 | 2 |

Global Cache Load Profile

| | Per Second | Per Transaction |
|--------------------------------|------------|-----------------|
| Global Cache blocks received: | 26.51 | 81.54 |
| Global Cache blocks served: | 26.02 | 80.01 |
| GCS/GES messages received: | 156.31 | 480.68 |
| GCS/GES messages sent: | 157.74 | 485.06 |
| DBWR Fusion writes: | 0.01 | 0.04 |
| Estd Interconnect traffic (KB) | 481.59 | |

Global Cache Efficiency Percentages (Target local+remote 100%)

| | |
|---------------------------------|-------|
| Buffer access - local cache %: | 95.44 |
| Buffer access - remote cache %: | 0.65 |
| Buffer access - disk %: | 3.91 |

Global Cache and Enqueue Services - Workload Characteristics

| | |
|---|------|
| Avg global enqueue get time (ms): | 0.2 |
| Avg global cache cr block receive time (ms): | 1.8 |
| Avg global cache current block receive time (ms): | 1.8 |
| Avg global cache cr block build time (ms): | 0.0 |
| Avg global cache cr block send time (ms): | 0.1 |
| Global cache log flushes for cr blocks served %: | 0.8 |
| Avg global cache cr block flush time (ms): | 17.5 |
| Avg global cache current block pin time (ms): | 0.0 |
| Avg global cache current block send time (ms): | 0.1 |
| Global cache log flushes for current blocks served %: | 0.0 |
| Avg global cache current block flush time (ms): | 20.0 |

Global Cache and Enqueue Services - Messaging Statistics

| | |
|---|-------|
| Avg message sent queue time (ms): | 1.1 |
| Avg message sent queue time on ksxp (ms): | 1.3 |
| Avg message received queue time (ms): | 0.1 |
| Avg GCS message process time (ms): | 0.0 |
| Avg GES message process time (ms): | 0.0 |
| % of direct sent messages: | 35.13 |
| % of indirect sent messages: | 64.34 |
| % of flow controlled messages: | 0.54 |

| | |
|--|-----|
| Avg global cache cr block receive time (ms) : | 1.8 |
| Avg global cache current block receive time (ms) : | 1.8 |

| | |
|--|-----|
| Avg global cache cr block receive time (ms) : | 2.1 |
| Avg global cache current block receive time (ms) : | 1.7 |

Cluster Interconnect

| Interface | IP Address | Pub Source | Begin | End |
|-----------|------------|-----------------------------|-------|------------|
| | | | C | IP Pub Src |
| eth0 | 11.1.1.1 | N Oracle Cluster Repository | | |

Time Model Statistics DB/Inst: AULTDB/aultdb1 Snaps: 91-92
-> Total time in database user-calls (DB Time): 8371.3s
-> Statistics including the word "background" measure background process
time, and so do not contribute to the DB time statistic
-> Ordered by % or DB time desc, Statistic name

| Statistic Name | Time (s) | % of DB Time |
|--|----------|--------------|
| sql execute elapsed time | 8,145.5 | 97.3 |
| DB CPU | 995.1 | 11.9 |
| parse time elapsed | 7.4 | .1 |
| hard parse elapsed time | 5.2 | .1 |
| PL/SQL execution elapsed time | 4.8 | .1 |
| Java execution elapsed time | 0.7 | .0 |
| hard parse (sharing criteria) elapsed time | 0.2 | .0 |
| sequence load elapsed time | 0.1 | .0 |
| repeated bind elapsed time | 0.1 | .0 |
| PL/SQL compilation elapsed time | 0.0 | .0 |
| failed parse elapsed time | 0.0 | .0 |
| hard parse (bind mismatch) elapsed time | 0.0 | .0 |
| DB time | 8,371.3 | |
| background elapsed time | 214.7 | |
| background cpu time | 75.8 | |

Operating System Statistics DB/Inst: AULTDB/aultdb1 Snaps: 91-92
-> \*TIME statistic values are diffed.
 All others display actual values. End Value is displayed if different
-> ordered by statistic type (CPU Use, Virtual Memory, Hardware Config), Name

| Statistic | Value | End Value |
|--------------------------|---------------|-----------|
| BUSY_TIME | 126,029 | |
| IDLE_TIME | 597,505 | |
| IOWAIT_TIME | 327,861 | |
| NICE_TIME | 766 | |
| SYS_TIME | 48,452 | |
| USER_TIME | 76,784 | |
| LOAD | 0 | 3 |
| PHYSICAL_MEMORY_BYTES | 3,189,190,656 | |
| NUM_CPUS | 2 | |
| NUM_CPU_CORES | 1 | |
| NUM_CPU_SOCKETS | 1 | |
| GLOBAL_RECEIVE_SIZE_MAX | 4,194,304 | |
| GLOBAL_SEND_SIZE_MAX | 262,144 | |
| TCP_RECEIVE_SIZE_DEFAULT | 87,380 | |
| TCP_RECEIVE_SIZE_MAX | 1,048,576 | |
| TCP_RECEIVE_SIZE_MIN | 4,096 | |
| TCP_SEND_SIZE_DEFAULT | 65,536 | |
| TCP_SEND_SIZE_MAX | 1,048,576 | |
| TCP_SEND_SIZE_MIN | 4,096 | |

Operating System Statistics - Detail DB/Inst: AULTDB/aultdb1 Snaps: 91-92

| Snap Time | Load | %busy | %user | %sys | %idle | %iowait |
|-----------------|------|-------|-------|------|-------|---------|
| 04-Aug 12:00:15 | 0.4 | N/A | N/A | N/A | N/A | N/A |
| 04-Aug 13:00:28 | 3.0 | 17.4 | 10.6 | 6.7 | 45.3 | 82.6 |

Foreground Wait Class DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> s - second, ms - millisecond - 1000th of a second
 -> ordered by wait time desc, waits desc
 -> %Timeouts: value of 0 indicates value was < .5%. Value of null is truly 0
 -> Captured Time accounts for 68.9% of Total DB time 8,371.33 (s)
 -> Total FG Wait Time: 4,770.85 (s) DB CPU time: 995.13 (s)

| Wait Class | % Time | | Total Wait Time (s) | Avg Wait | |
|---------------|-----------|-------|---------------------|----------|-----------|
| | Waits | -outs | | (ms) | % DB Time |
| User I/O | 518,267 | 0 | 4,449 | 9 | 53.1 |
| DB CPU | | | 995 | | 11.9 |
| Cluster | 188,753 | 9 | 173 | 1 | 2.1 |
| Other | 3,806,446 | 100 | 146 | 0 | 1.7 |
| Concurrency | 1,854 | 2 | 2 | 1 | 0.0 |
| Commit | 15 | 0 | 1 | 39 | 0.0 |
| Application | 740 | 0 | 0 | 0 | 0.0 |
| System I/O | 40 | 0 | 0 | 3 | 0.0 |
| Network | 6,970 | 0 | 0 | 0 | 0.0 |
| Configuration | 0 | | 0 | | 0.0 |

Foreground Wait Events DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> s - second, ms - millisecond - 1000th of a second
 -> Only events with Total Wait Time (s) >= .001 are shown
 -> ordered by wait time desc, waits desc (idle events last)
 -> %Timeouts: value of 0 indicates value was < .5%. Value of null is truly 0

| Event | % Time | | Total Wait Time (s) | Wait (ms) | Avg | |
|----------------------------|-----------|-------|---------------------|-----------|---------|-----------|
| | Waits | -outs | | | /txn | % DB Time |
| db file sequential read | 465,020 | 0 | 3,969 | 9 | 395.8 | 47.4 |
| db file parallel read | 2,251 | 0 | 322 | 143 | 1.9 | 3.8 |
| db file scattered read | 15,268 | 0 | 153 | 10 | 13.0 | 1.8 |
| gc current block 2-way | 108,739 | 11 | 116 | 1 | 92.5 | 1.4 |
| PX Deq: reap credit | 3,247,703 | 100 | 107 | 0 | 2,764.0 | 1.3 |
| gc cr grant 2-way | 57,265 | 7 | 28 | 0 | 48.7 | .3 |
| gc cr multi block request | 22,451 | 6 | 23 | 1 | 19.1 | .3 |
| enq: BF - allocation conte | 14 | 93 | 14 | 983 | 0.0 | .2 |
| PX qref latch | 555,843 | 100 | 9 | 0 | 473.1 | .1 |
| IPC send completion sync | 1,070 | 52 | 8 | 8 | 0.9 | .1 |
| gc remaster | 22 | 0 | 5 | 221 | 0.0 | .1 |

Background Wait Events DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> ordered by wait time desc, waits desc (idle events last)
 -> Only events with Total Wait Time (s) >= .001 are shown
 -> %Timeouts: value of 0 indicates value was < .5%. Value of null is truly 0

| Event | Waits | % Time | Total Wait Time (s) | Avg | Waits /txn | % bg Time |
|----------------------------|--------|--------|---------------------|-----------|------------|-----------|
| | | | | Wait (ms) | | |
| control file sequential re | 8,336 | 0 | 72 | 9 | 7.1 | 33.5 |
| control file parallel writ | 1,287 | 0 | 31 | 24 | 1.1 | 14.5 |
| db file parallel write | 792 | 0 | 11 | 14 | 0.7 | 5.3 |
| log file parallel write | 701 | 0 | 11 | 15 | 0.6 | 4.9 |
| events in waitclass Other | 44,191 | 98 | 5 | 0 | 37.6 | 2.5 |
| library cache pin | 449 | 0 | 2 | 4 | 0.4 | .8 |
| db file sequential read | 221 | 0 | 2 | 7 | 0.2 | .8 |
| gc cr multi block request | 1,915 | 0 | 2 | 1 | 1.6 | .7 |
| os thread startup | 19 | 0 | 1 | 56 | 0.0 | .5 |
| gc cr block 2-way | 246 | 0 | 0 | 1 | 0.2 | .2 |
| db file scattered read | 18 | 0 | 0 | 12 | 0.0 | .1 |
| db file parallel read | 3 | 0 | 0 | 59 | 0.0 | .1 |
| gc current grant 2-way | 98 | 0 | 0 | 1 | 0.1 | .1 |

Wait Event Histogram DB/Inst: AULTDB/aultdb1 Snaps: 91-92
-> Units for Total Waits column: K is 1000, M is 1000000, G is 1000000000
-> % of Waits: value of .0 indicates value was <.05%. Value of null is truly 0
-> % of Waits: column heading of <=1s is truly <1024ms, >1s is truly >=1024ms
-> Ordered by Event (idle events last)

| Event | Total | % of Waits | | | | | | | | | |
|----------------------------|-------|------------|------|------|------|------|-------|-------|------|------|-----|
| | | Waits | <1ms | <2ms | <4ms | <8ms | <16ms | <32ms | <=1s | >1s | |
| control file parallel writ | 1287 | | | | | | 59.0 | 24.1 | 16.9 | | |
| control file sequential re | 9147 | | | | 23.4 | 21.3 | 23.3 | 22.3 | 6.8 | 2.9 | .0 |
| db file parallel read | 2256 | | | | | .3 | 1.0 | 7.4 | 32.6 | 56.8 | 1.9 |
| db file parallel write | 792 | | .5 | .8 | 4.2 | 28.7 | 50.0 | 8.8 | 7.1 | | |
| db file scattered read | 15K | | | | .4 | 2.7 | 31.5 | 59.2 | 5.8 | .5 | |
| db file sequential read | 465K | | .0 | .6 | 2.2 | 49.5 | 45.0 | 2.3 | .4 | | |
| gc cr grant 2-way | 50K | 87.2 | 11.1 | 1.3 | .3 | .2 | | | .0 | | |
| gc cr multi block request | 24K | 59.0 | 36.8 | 3.0 | .5 | .6 | .0 | | | | |
| gc current block 2-way | 84K | 6.5 | 87.7 | 5.2 | .3 | .2 | .0 | | | | |
| library cache lock | 488 | 82.8 | 10.9 | 4.9 | 1.0 | .2 | .2 | | | | |
| library cache pin | 4371 | 77.6 | 11.1 | 7.4 | 3.1 | .6 | .0 | | | | |
| gcs remote message | 274K | 28.5 | 15.4 | 9.9 | 11.6 | 7.5 | 5.8 | 21.4 | | | |
| ges remote message | 53K | 11.4 | 3.3 | 2.7 | 1.9 | 1.8 | 2.1 | 76.8 | | | |

Service Statistics
-> ordered by DB Time

DB/Inst: AULTDB/aultdb1 Snaps: 91-92

| Service Name | DB Time (s) | DB CPU (s) | Physical Reads (K) | Logical Reads (K) |
|-----------------|--------------|------------|--------------------|-------------------|
| aultdb | 8,344 | 981 | 9,769 | 22,715 |
| SYS\$USERS | 23 | 12 | 1 | 56 |
| SYS\$BACKGROUND | 1 | 0 | 1 | 17 |
| aultdbXDB | 0 | 0 | 0 | 0 |

Service Wait Class Stats

DB/Inst: AULTDB/aultdb1 Snaps: 91-92

-> Wait Class info for services in the Service Statistics section.
-> Total Waits and Time Waited displayed for the following wait
classes: User I/O, Concurrency, Administrative, Network
-> Time Waited (Wt Time) in seconds

Service Name

| User I/O | User I/O | Concurcy | Concurcy | Admin | Admin | Network | Network |
|-----------------|----------|-----------|----------|-----------|---------|-----------|---------|
| Total Wts | Wt Time | Total Wts | Wt Time | Total Wts | Wt Time | Total Wts | Wt Time |
| aultdb | | | | | | | |
| 517710 | 4446 | 234 | 1 | 0 | 0 | 5828 | 0 |
| SYS\$USERS | | | | | | | |
| 555 | 3 | 1615 | 1 | 0 | 0 | 1140 | 0 |
| SYS\$BACKGROUND | | | | | | | |
| 350 | 3 | 3486 | 4 | 0 | 0 | 0 | 0 |

| Instance Activity Stats | DB/Inst: AULTDB/aultdb1 | Snaps: 91-92 | |
|-----------------------------------|-------------------------|--------------|--------------|
| Statistic | Total | per Second | per Trans |
| CPU used by this session | 77,997 | 21.6 | 66.4 |
| CPU used when call started | 288,270 | 79.8 | 245.3 |
| Number of read IOs issued | 27,685 | 7.7 | 23.6 |
| SQL*Net roundtrips to/from client | 6,970 | 1.9 | 5.9 |
| bytes received via SQL*Net from | 2,385,638 | 660.2 | 2,030.3 |
| bytes sent via SQL*Net to client | 2,595,626 | 718.4 | 2,209.0 |
| consistent gets | 22,777,682 | 6,303.9 | 19,385.3 |
| consistent gets - examination | 6,073,207 | 1,680.8 | 5,168.7 |
| consistent gets direct | 3,277,142 | 907.0 | 2,789.1 |
| consistent gets from cache | 14,648,585 | 4,054.1 | 12,466.9 |
| consistent gets from cache (fast) | 193,221 | 53.5 | 164.4 |
| db block changes | 12,812 | 3.6 | 10.9 |
| db block gets | 13,389 | 3.7 | 11.4 |
| db block gets from cache | 13,364 | 3.7 | 11.4 |
| db block gets from cache (fastpa | 3,512 | 1.0 | 3.0 |
| dirty buffers inspected | 825 | 0.2 | 0.7 |
| enqueue timeouts | 40 | 0.0 | 0.0 |
| enqueue waits | 499 | 0.1 | 0.4 |
| execute count | 13,287 | 3.7 | 11.3 |
| free buffer inspected | 556,747 | 154.1 | 473.8 |
| free buffer requested | 731,667 | 202.5 | 622.7 |
| gc CPU used by this session | 11,859 | 3.3 | 10.1 |
| gc blocks lost | 0 | 0.0 | 0.0 |
| gc cr block build time | 1 | 0.0 | 0.0 |
| gc cr block flush time | 7 | 0.0 | 0.0 |
| gc cr block receive time | 66 | 0.0 | 0.1 |
| gc cr block send time | 3 | 0.0 | 0.0 |
| gc cr blocks received | 361 | 0.1 | 0.3 |
| gc cr blocks served | 522 | 0.1 | 0.4 |
| gc current block flush time | 2 | 0.0 | 0.0 |
| gc current block pin time | 205 | 0.1 | 0.2 |
| gc current block receive time | 16,726 | 4.6 | 14.2 |
| gc current block send time | 577 | 0.2 | 0.5 |
| gc current blocks received | 95,445 | 26.4 | 81.2 |
| gc current blocks served | 93,484 | 25.9 | 79.6 |
| index fast full scans (direct re | 90 | 0.0 | 0.1 |
| index fast full scans (full) | 4 | 0.0 | 0.0 |
| index fast full scans (rowid ran | 90 | 0.0 | 0.1 |
| index fetch by key | 3,086,965 | 854.3 | 2,627.2 |
| index scans kdiixs1 | 29,551 | 8.2 | 25.2 |
| leaf node 90-10 splits | 19 | 0.0 | 0.0 |
| leaf node splits | 26 | 0.0 | 0.0 |
| opened cursors cumulative | 13,077 | 3.6 | 11.1 |
| parse count (failures) | 2 | 0.0 | 0.0 |
| parse count (hard) | 153 | 0.0 | 0.1 |
| parse count (total) | 7,202 | 2.0 | 6.1 |
| parse time cpu | 227 | 0.1 | 0.2 |
| parse time elapsed | 399 | 0.1 | 0.3 |
| physical read IO requests | 550,974 | 152.5 | 468.9 |
| physical read bytes | 32,562,569,216 | 9,011,916.7 | 27,712,824.9 |
| physical read total IO requests | 605,019 | 167.4 | 514.9 |
| physical read total bytes | 32,711,421,952 | 9,053,112.7 | 27,839,508.0 |
| physical read total multi block | 30,330 | 8.4 | 25.8 |
| physical reads | 9,773,380 | 2,704.9 | 8,317.8 |
| physical reads cache | 572,745 | 158.5 | 487.4 |
| physical reads cache prefetch | 153,965 | 42.6 | 131.0 |
| physical reads direct | 3,402,178 | 941.6 | 2,895.5 |
| physical reads direct temporary | 124,434 | 34.4 | 105.9 |
| physical reads prefetch warmup | 58,580 | 16.2 | 49.9 |
| physical write IO requests | 4,983 | 1.4 | 4.2 |
| physical write bytes | 1,037,123,584 | 287,031.1 | 882,658.4 |
| physical write total IO requests | 15,031 | 4.2 | 12.8 |
| physical write total bytes | 1,085,801,472 | 300,503.1 | 924,086.4 |
| physical write total multi block | 4,062 | 1.1 | 3.5 |

| | | | |
|----------------------------------|-------------|----------|-----------|
| physical writes | 314,090 | 86.9 | 267.3 |
| physical writes direct | 124,459 | 34.4 | 105.9 |
| physical writes direct (lob) | 0 | 0.0 | 0.0 |
| physical writes direct temporary | 124,434 | 34.4 | 105.9 |
| physical writes from cache | 2,143 | 0.6 | 1.8 |
| physical writes non checkpoint | 124,952 | 34.6 | 106.3 |
| recursive calls | 78,415 | 21.7 | 66.7 |
| recursive cpu usage | 77,189 | 21.4 | 65.7 |
| redo entries | 7,832 | 2.2 | 6.7 |
| redo log space requests | 2 | 0.0 | 0.0 |
| redo log space wait time | 28 | 0.0 | 0.0 |
| redo size | 2,892,568 | 800.5 | 2,461.8 |
| redo synch time | 66 | 0.0 | 0.1 |
| redo synch writes | 72 | 0.0 | 0.1 |
| redo wastage | 196,192 | 54.3 | 167.0 |
| redo write time | 1,110 | 0.3 | 0.9 |
| redo writes | 701 | 0.2 | 0.6 |
| rollback changes - undo records | 0 | 0.0 | 0.0 |
| session cursor cache hits | 12,415 | 3.4 | 10.6 |
| session logical reads | 22,791,070 | 6,307.6 | 19,396.7 |
| sorts (memory) | 3,875 | 1.1 | 3.3 |
| sorts (rows) | 1,460,468 | 404.2 | 1,243.0 |
| summed dirty queue length | 3,284 | 0.9 | 2.8 |
| table fetch by rowid | 1,322,667 | 366.1 | 1,125.7 |
| table fetch continued row | 13 | 0.0 | 0.0 |
| table scan blocks gotten | 2,780,775 | 769.6 | 2,366.6 |
| table scan rows gotten | 158,164,979 | 43,773.3 | 134,608.5 |
| table scans (direct read) | 776 | 0.2 | 0.7 |
| table scans (long tables) | 776 | 0.2 | 0.7 |
| table scans (rowid ranges) | 776 | 0.2 | 0.7 |
| table scans (short tables) | 2,255 | 0.6 | 1.9 |
| transaction rollbacks | 0 | 0.0 | 0.0 |
| undo change vector size | 1,870,904 | 517.8 | 1,592.3 |
| user I/O wait time | 445,246 | 123.2 | 378.9 |
| user calls | 7,943 | 2.2 | 6.8 |
| user commits | 794 | 0.2 | 0.7 |
| user rollbacks | 381 | 0.1 | 0.3 |
| workarea executions - onepass | 6 | 0.0 | 0.0 |
| workarea executions - optimal | 2,323 | 0.6 | 2.0 |

Instance Activity Stats - Absolute Values

DB/Inst: AULTDB/aultdb1 Snaps: 91-9

-> Statistics with absolute values (should not be diffed)

| Statistic | Begin Value | End Value |
|----------------------------|----------------|---------------|
| session pga memory max | 544,192,924 | 4,940,081,136 |
| session cursor cache count | 2,266 | 8,279 |
| session uga memory | 73,033,165,084 | 3.393545E+11 |
| opened cursors current | 48 | 54 |
| workarea memory allocated | 0 | 16,041 |
| logons current | 41 | 47 |
| session uga memory max | 4,427,536,236 | 5,963,059,828 |
| session pga memory | 390,773,148 | 826,689,340 |

Instance Activity Stats - Thread Activity

DB/Inst: AULTDB/aultdb1 Snaps: 91-92

-> Statistics identified by '(derived)' come from sources other than SYSSTAT

| Statistic | Total | per Hour |
|------------------------|-------|----------|
| log switches (derived) | 1 | 1.00 |

| Statistic | Total | per Second | per Trans |
|----------------------------|-------------|------------|-----------|
| table fetch by rowid | 1,322,667 | 366.1 | 1,125.7 |
| table fetch continued row | 13 | 0.0 | 0.0 |
| table scan blocks gotten | 2,780,775 | 769.6 | 2,366.6 |
| table scan rows gotten | 158,164,979 | 43,773.3 | 134,608.5 |
| table scans (direct read) | 776 | 0.2 | 0.7 |
| table scans (long tables) | 776 | 0.2 | 0.7 |
| table scans (rowid ranges) | 776 | 0.2 | 0.7 |
| table scans (short tables) | 2,255 | 0.6 | 1.9 |

```
Tablespace IO Stats           DB/Inst: AULTDB/aultdb1  Snaps: 91-92
-> ordered by IOs (Reads + Writes) desc
Tablespace
```

| | Av
Reads | Av
Reads/s | Av
Rd(ms) | Av
Blks/Rd | | Av
Writes | Av
Writes/s | Buffer
Waits | Av
Buf
Wt(ms) |
|----------|-------------|---------------|--------------|---------------|--|--------------|----------------|-----------------|---------------------|
| <hr/> | | | | | | | | | |
| DATA | 512,639 | 142 | 11.8 | 6.4 | | 0 | 0 | 6 | 151.7 |
| INDEXES | 32,625 | 9 | 11.3 | 16.7 | | 0 | 0 | 37 | 83.5 |
| TEMP | 4,024 | 1 | 17.6 | 30.9 | | 4,014 | 1 | 0 | 0.0 |
| SYSAux | 571 | 0 | 29.3 | 1.4 | | 698 | 0 | 0 | 0.0 |
| SYSTEM | 471 | 0 | 5.3 | 1.8 | | 56 | 0 | 0 | 0.0 |
| UNDOTBS1 | 9 | 0 | 10.0 | 1.0 | | 215 | 0 | 1 | 10.0 |
| USERS | 1 | 0 | 10.0 | 1.0 | | 0 | 0 | 0 | 0.0 |

File IO Stats DB/Inst: AULTDB/aultdb1 Snaps: 91-92
-> ordered by Tablespace, File
Tablespace Filename

| | AV | AV | AV | | AV | Buffer | Av | Bu |
|----------|---------|---------|--------|---|--------|----------|-------|--------|
| | Reads | Reads/s | Rd(ms) | Blks/Rd | Writes | Writes/s | Waits | Wt(ms) |
| DATA | | | | +DATA/autldb/datafile/data.257.660765277 | | | | |
| | 501,566 | 139 | 10.8 | 5.2 | 0 | 0 | 6 | 151.7 |
| DATA | | | | +DATA/autldb/datafile/data.259.660843403 | | | | |
| | 11,073 | 3 | 56.9 | 64.5 | 0 | 0 | 0 | 0.0 |
| INDEXES | | | | +DATA/autldb/datafile/indexes.258.660765437 | | | | |
| | 32,625 | 9 | 11.3 | 16.7 | 0 | 0 | 37 | 83.5 |
| SYSAUX | | | | +DATA2/autldb/datafile/sysaux.257.660755929 | | | | |
| | 571 | 0 | 29.3 | 1.4 | 698 | 0 | 0 | 0.0 |
| SYSTEM | | | | +DATA2/autldb/datafile/system.256.660755927 | | | | |
| | 471 | 0 | 5.3 | 1.8 | 56 | 0 | 0 | 0.0 |
| TEMP | | | | +DATA2/autldb/tempfile/temp.266.660756117 | | | | |
| | 1,340 | 0 | 17.5 | 31.0 | 1,338 | 0 | 0 | N/A |
| TEMP | | | | +DATA2/autldb/tempfile/temp.272.660915285 | | | | |
| | 2 | 0 | 10.0 | 1.0 | 0 | 0 | 0 | N/A |
| TEMP | | | | +DATA2/autldb/tempfile/temp.273.660930207 | | | | |
| | 2 | 0 | 10.0 | 1.0 | 0 | 0 | 0 | N/A |
| TEMP | | | | +DATA2/autldb/tempfile/temp.274.660989059 | | | | |
| | 1,340 | 0 | 17.7 | 31.0 | 1,338 | 0 | 0 | N/A |
| TEMP | | | | +DATA2/autldb/tempfile/temp.275.661003761 | | | | |
| | 1,340 | 0 | 17.6 | 31.0 | 1,338 | 0 | 0 | N/A |
| UNDOTBS1 | | | | +DATA2/autldb/datafile/undotbs1.258.660755929 | | | | |
| | 9 | 0 | 10.0 | 1.0 | 215 | 0 | 1 | 10.0 |
| USERS | | | | +DATA2/autldb/datafile/users.259.660755929 | | | | |
| | 1 | 0 | 10.0 | 1.0 | 0 | 0 | 0 | 0.0 |

Buffer Pool Statistics DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> Standard block size Pools D: default, K: keep, R: recycle
 -> Default Pools for other block sizes: 2k, 4k, 8k, 16k, 32k

| P | Number of Pool | Buffer | Physical | Physical | Free Writ | Buffer | | |
|---|----------------|--------|-----------|----------|-----------|--------|---|----|
| | Buffers | Gets | Reads | Writes | Buff Comp | Busy | | |
| D | 159,244 | 91 | 6,287,434 | 572,581 | 2,143 | 0 | 0 | 44 |

Instance Recovery Stats DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> B: Begin snapshot, E: End snapshot

| Targt | Estd | Log | File | Log | Ckpt | Log | Ckpt |
|-------|------|----------|--------|-----------|-----------|-----------|-----------|
| MTTR | MTTR | Recovery | Actual | Target | Size | Timeout | Interval |
| (s) | (s) | Estd | Ios | Redo Blks | Redo Blks | Redo Blks | Redo Blks |
| B | 0 | 0 | 250 | 974 | 1015 | 92160 | 1015 |
| E | 0 | 0 | 292 | 1192 | 2751 | 92160 | 2751 |

Buffer Pool Advisory DB/Inst: AULTDB/aultdb1 Snap: 91-92
 -> Only rows with estimated physical reads >0 are displayed
 -> ordered by Block Size, Buffers For Estimate

| P | Size for Est (M) | Size Factor | Est | | Estimated Physical Reads |
|---|------------------|-------------|----------------------|-------------|--------------------------|
| | | | Buffers for Estimate | Read Factor | |
| D | 128 | .1 | 15,536 | 3.4 | 1,988,649 |
| D | 256 | .2 | 31,072 | 2.5 | 1,497,186 |
| D | 384 | .3 | 46,608 | 2.0 | 1,196,087 |
| D | 512 | .4 | 62,144 | 1.6 | 959,386 |
| D | 640 | .5 | 77,680 | 1.3 | 787,130 |
| D | 768 | .6 | 93,216 | 1.1 | 674,908 |
| D | 896 | .7 | 108,752 | 1.0 | 620,121 |
| D | 1,024 | .8 | 124,288 | 1.0 | 599,692 |
| D | 1,152 | .9 | 139,824 | 1.0 | 593,191 |
| D | 1,280 | 1.0 | 155,360 | 1.0 | 592,402 |
| D | 1,312 | 1.0 | 159,244 | 1.0 | 592,402 |
| D | 1,408 | 1.1 | 170,896 | 1.0 | 592,356 |
| D | 1,536 | 1.2 | 186,432 | 1.0 | 591,798 |
| D | 1,664 | 1.3 | 201,968 | 1.0 | 591,798 |
| D | 1,792 | 1.4 | 217,504 | 1.0 | 591,798 |
| D | 1,920 | 1.5 | 233,040 | 1.0 | 591,798 |
| D | 2,048 | 1.6 | 248,576 | 1.0 | 591,798 |
| D | 2,176 | 1.7 | 264,112 | 1.0 | 591,798 |
| D | 2,304 | 1.8 | 279,648 | 1.0 | 591,798 |
| D | 2,432 | 1.9 | 295,184 | 1.0 | 591,798 |
| D | 2,560 | 2.0 | 310,720 | 1.0 | 591,798 |

PGA Aggr Summary DB/Inst: AULTDB/aultdb1 Snaps: 91-92
-> PGA cache hit % - percentage of W/A (WorkArea) data processed only in-memory

| PGA Cache Hit % | W/A MB Processed | Extra W/A MB Read/Written |
|-----------------|------------------|---------------------------|
| 54.8 | 2,843 | 2,345 |

PGA Aggr Target Stats DB/Inst: AULTDB/aultdb1 Snaps: 91-92

No data exists for this section of the report.

PGA Aggr Target Histogram DB/Inst: AULTDB/aultdb1 Snaps: 91-92
-> Optimal Executions are purely in-memory operations

| Low Optimal | High Optimal | Total Execs | Optimal Execs | 1-Pass Execs | M-Pass Execs |
|-------------|--------------|-------------|---------------|--------------|--------------|
| 2K | 4K | 1,833 | 1,833 | 0 | 0 |
| 64K | 128K | 5 | 5 | 0 | 0 |
| 128K | 256K | 1 | 1 | 0 | 0 |
| 256K | 512K | 6 | 6 | 0 | 0 |
| 512K | 1024K | 439 | 439 | 0 | 0 |
| 1M | 2M | 6 | 6 | 0 | 0 |
| 2M | 4M | 6 | 6 | 0 | 0 |
| 4M | 8M | 14 | 14 | 0 | 0 |
| 8M | 16M | 6 | 6 | 0 | 0 |
| 16M | 32M | 4 | 4 | 0 | 0 |
| 32M | 128M | 3 | 3 | 0 | 0 |
| 128M | 512M | 6 | 0 | 6 | 0 |

PGA Memory Advisory DB/Inst: AULTDB/aultdb1 Snap: 92
-> When using Auto Memory Mgmt, minimally choose a pga\_aggregate\_target value where Estd PGA Overalloc Count is 0

| PGA Target Est (MB) | Size Factr | W/A MB Processed | W/A MB Read/Written to Disk | Estd Cache Hit % | Extra Overallo Count | Estd P Time | Estd PGA |
|---------------------|------------|------------------|-----------------------------|------------------|----------------------|-------------|----------|
| 64 | 0.1 | 3,388.1 | 6,390.6 | 35.0 | 22 | 1.6E+05 | |
| 128 | 0.3 | 3,388.1 | 5,795.7 | 37.0 | 2 | 1.5E+05 | |
| 256 | 0.5 | 3,388.1 | 4,885.0 | 41.0 | 1 | 1.3E+05 | |
| 384 | 0.8 | 3,388.1 | 1,172.5 | 74.0 | 0 | 74,015 | |
| 512 | 1.0 | 3,388.1 | 1,172.5 | 74.0 | 0 | 74,015 | |
| 614 | 1.2 | 3,388.1 | 1,172.5 | 74.0 | 0 | 74,015 | |
| 717 | 1.4 | 3,388.1 | 1,172.5 | 74.0 | 0 | 74,015 | |
| 819 | 1.6 | 3,388.1 | 1,172.5 | 74.0 | 0 | 74,015 | |
| 922 | 1.8 | 3,388.1 | 1,172.5 | 74.0 | 0 | 74,015 | |
| 1,024 | 2.0 | 3,388.1 | 1,172.5 | 74.0 | 0 | 74,015 | |
| 1,536 | 3.0 | 3,388.1 | 1,172.5 | 74.0 | 0 | 74,015 | |
| 2,048 | 4.0 | 3,388.1 | 1,172.5 | 74.0 | 0 | 74,015 | |
| 3,072 | 6.0 | 3,388.1 | 1,172.5 | 74.0 | 0 | 74,015 | |
| 4,096 | 8.0 | 3,388.1 | 1,172.5 | 74.0 | 0 | 74,015 | |

| | PGA Aggr | Auto PGA | PGA Mem | W/A | PGA | W/A | %Auto | %Man | Global Mem |
|---|-----------|-----------|----------|---------|-------|---------|---------|------|------------|
| | Target(M) | Target(M) | Alloc(M) | Used(M) | % Mem | W/A Mem | W/A Mem | | Bound(K) |
| B | 1,628 | 1,434 | 425.37 | 284.23 | 66.82 | 100.00 | 0.00 | | 166,700 |
| E | 1,628 | 1,424 | 315.79 | 177.43 | 56.19 | 100.00 | 0.00 | | 166,700 |

PGA Aggr Target Histogram DB/Inst: AULTDB/aultdb1 Snaps: 91-92
-> Optimal Executions are purely in-memory operations

| Low | High | Total Execs | Optimal Execs | 1-Pass Execs | M-Pass Execs |
|---------|---------|-------------|---------------|--------------|--------------|
| Optimal | Optimal | | | | |
| 2K | 4K | 224,226 | 224,226 | 0 | 0 |
| 64K | 128K | 1,560 | 1,560 | 0 | 0 |
| 128K | 256K | 1,150 | 1,150 | 0 | 0 |
| 256K | 512K | 4,298 | 4,298 | 0 | 0 |
| 512K | 1024K | 7,200,201 | 7,200,201 | 0 | 0 |
| 1M | 2M | 11,248 | 11,248 | 0 | 0 |
| 2M | 4M | 793 | 641 | 152 | 0 |
| 4M | 8M | 255 | 125 | 130 | 0 |
| 8M | 16M | 32 | 32 | 0 | 0 |
| 16M | 32M | 16 | 16 | 0 | 0 |
| 32M | 64M | 51 | 41 | 10 | 0 |
| 64M | 128M | 26 | 26 | 0 | 0 |
| 256M | 512M | 4 | 1 | 3 | 0 |
| 512M | 1024M | 10 | 0 | 10 | 0 |
| 1G | 2G | 7 | 0 | 7 | 0 |

PGA Memory Advisory DB/Inst: AULTDB/aultdb1 Snap: 92
-> When using Auto Memory Mgmt, minimally choose a pga\_aggregate\_target value where Estd PGA Overalloc Count is 0

| PGA Target
Est (MB) | Size
Factr | W/A MB
Processed | Estd Extra
W/A MB Read/
Written to Disk | Estd PGA
Cache Hit % | Estd PGA
Overalloc Count |
|------------------------|---------------|---------------------|---|-------------------------|-----------------------------|
| 625 | 0.1 | 915,596,365.5 | 25,373,976.7 | 97.0 | 2,212 |
| 1,250 | 0.3 | 915,596,365.5 | 12,295,085.9 | 99.0 | 78 |
| 2,500 | 0.5 | 915,596,365.5 | 10,692,609.6 | 99.0 | 26 |
| 3,750 | 0.8 | 915,596,365.5 | 10,131,856.2 | 99.0 | 0 |
| 5,000 | 1.0 | 915,596,365.5 | 9,588,475.5 | 99.0 | 0 |
| 6,000 | 1.2 | 915,596,365.5 | 3,147,080.7 | 100.0 | 0 |
| 7,000 | 1.4 | 915,596,365.5 | 3,113,640.4 | 100.0 | 0 |
| 8,000 | 1.6 | 915,596,365.5 | 3,103,539.7 | 100.0 | 0 |
| 9,000 | 1.8 | 915,596,365.5 | 3,106,428.3 | 100.0 | 0 |
| 10,000 | 2.0 | 915,596,365.5 | 3,088,330.4 | 100.0 | 0 |
| 15,000 | 3.0 | 915,596,365.5 | 3,088,330.4 | 100.0 | 0 |
| 20,000 | 4.0 | 915,596,365.5 | 3,088,330.4 | 100.0 | 0 |
| 30,000 | 6.0 | 915,596,365.5 | 3,088,330.4 | 100.0 | 0 |
| 40,000 | 8.0 | 915,596,365.5 | 3,088,330.4 | 100.0 | 0 |

Shared Pool Advisory DB/Inst: AULTDB/aultdb1 Snap: 92
 --> SP: Shared Pool Est LC: Estimated Library Cache Factr: Factor
 --> Note there is often a 1:Many correlation between a single logical object in the Library Cache, and the physical number of memory objects
 Associated with it. Therefore comparing the number of Lib Cache objects (e.g. in v\$librarycache), with the number of Lib Cache Memory Objects is invalid.

| Shared
Pool
Size(M) | SP
Factr | Est LC
(M) | Est LC
Mem Obj | Est LC | | Est LC | | Est LC
Mem Obj |
|---------------------------|-------------|---------------|-------------------|----------------------|---------------|---------------------|---------------|-------------------|
| | | | | Time
Saved
(s) | Time
Factr | Load
Time
(s) | Load
Factr | |
| 192 | .9 | 3 | 495 | 4,555 | 1.0 | 41 | 1.0 | 12 |
| 224 | 1.0 | 33 | 4,350 | 4,555 | 1.0 | 41 | 1.0 | 96 |
| 256 | 1.1 | 45 | 6,645 | 4,557 | 1.0 | 39 | 1.0 | 96 |
| 288 | 1.3 | 45 | 6,645 | 4,558 | 1.0 | 38 | .9 | 96 |
| 320 | 1.4 | 45 | 6,645 | 4,558 | 1.0 | 38 | .9 | 96 |
| 352 | 1.6 | 45 | 6,645 | 4,558 | 1.0 | 38 | .9 | 96 |
| 384 | 1.7 | 45 | 6,645 | 4,558 | 1.0 | 38 | .9 | 96 |
| 416 | 1.9 | 45 | 6,645 | 4,558 | 1.0 | 38 | .9 | 96 |
| 448 | 2.0 | 45 | 6,645 | 4,558 | 1.0 | 38 | .9 | 96 |

SGA Target Advisory

DB/Inst: AULTDB/aultdb1 Snap: 92

| SGA Target Size (M) | SGA Size Factor | Est DB Time (s) | Est Physical Reads |
|---------------------|-----------------|-----------------|--------------------|
| 396 | 0.3 | 8,538 | 592,206 |
| 792 | 0.5 | 8,536 | 592,206 |
| 1,188 | 0.8 | 8,536 | 592,206 |
| 1,584 | 1.0 | 8,536 | 592,206 |
| 1,980 | 1.3 | 8,536 | 592,206 |
| 2,376 | 1.5 | 8,542 | 592,206 |
| 2,772 | 1.8 | 8,542 | 592,206 |
| 3,168 | 2.0 | 8,542 | 592,206 |

Streams Pool Advisory

DB/Inst: AULTDB/aultdb1 Snap: 92

| Size for Est (MB) | Size Factor | Spill Count | Est Spill Time (s) | Unspill Count | Est Unspill Time (s) |
|-------------------|-------------|-------------|--------------------|---------------|----------------------|
| 32 | 0.13 | 0 | 0 | 0 | 0 |
| 64 | 0.25 | 0 | 0 | 0 | 0 |
| 96 | 0.38 | 0 | 0 | 0 | 0 |
| 128 | 0.50 | 0 | 0 | 0 | 0 |
| 160 | 0.63 | 0 | 0 | 0 | 0 |
| 192 | 0.75 | 0 | 0 | 0 | 0 |
| 224 | 0.88 | 0 | 0 | 0 | 0 |
| 256 | 1.00 | 0 | 0 | 0 | 0 |
| 288 | 1.13 | 0 | 0 | 0 | 0 |
| 320 | 1.25 | 0 | 0 | 0 | 0 |
| 352 | 1.38 | 0 | 0 | 0 | 0 |
| 384 | 1.50 | 0 | 0 | 0 | 0 |
| 416 | 1.63 | 0 | 0 | 0 | 0 |
| 448 | 1.75 | 0 | 0 | 0 | 0 |
| 480 | 1.88 | 0 | 0 | 0 | 0 |
| 512 | 2.00 | 0 | 0 | 0 | 0 |
| 544 | 2.13 | 0 | 0 | 0 | 0 |
| 576 | 2.25 | 0 | 0 | 0 | 0 |
| 608 | 2.38 | 0 | 0 | 0 | 0 |
| 640 | 2.50 | 0 | 0 | 0 | 0 |

Java Pool Advisory

DB/Inst: AULTDB/aultdb1 Snap: 92

| Java Pool Size(M) | JP Factr | Est LC (M) | Est LC | | Est LC | | Est LC | | Est LC Mem |
|-------------------|----------|------------|------------|-----------|-----------|-------|----------|-------|------------|
| | | | Time | Time | Load | Load | Time | Time | |
| Size(M) | Factr | Size (M) | Est LC Mem | Saved Obj | Saved (s) | Factr | Time (s) | Factr | Obj Hits |
| 16 | 1.0 | 2 | 79 | 7 | 1.0 | 41 | 1.0 | 88 | |
| 32 | 2.0 | 4 | 163 | 7 | 1.0 | 41 | 1.0 | 182 | |

Buffer Wait Statistics DB/Inst: AULTDB/aultdb1 Snaps: 91-92
-> ordered by wait time desc, waits desc

| Class | Waits | Total Wait Time (s) | Avg Time (ms) |
|-------------|-------|---------------------|---------------|
| data block | 43 | 4 | 93 |
| undo header | 1 | 0 | 10 |

Enqueue Activity DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> only enqueues with waits are shown
 -> Enqueue stats gathered prior to 10g should not be compared with 10g
 -> ordered by Wait Time desc, Waits desc

Enqueue Type (Request Reason)

| | Requests | Succ Gets | Failed Gets | Waits | Wt Time (s) | Av Wt Time(ms) |
|--|----------|-----------|-------------|-------|-------------|----------------|
| <hr/> | | | | | | |
| BF-BLOOM FILTER (allocation contention) | 2,618 | 2,611 | 7 | 14 | 14 | 982.14 |
| TD-KTF map table enqueue (KTF dump entries) | | | | | | |
| | 9 | 9 | 0 | 9 | 0 | 37.78 |
| PS-PX Process Reservation | 648 | 616 | 32 | 208 | 0 | .96 |
| CF-Controlfile Transaction | | | | | | |
| | 7,661 | 7,660 | 1 | 118 | 0 | 1.27 |
| TM-DML | 5,559 | 5,559 | 0 | 16 | 0 | 3.75 |
| XL-ASM Extent Fault Lock (fault extent map) | | | | | | |
| | 14 | 14 | 0 | 1 | 0 | 60.00 |
| TT-Tablespace | 1,125 | 1,125 | 0 | 30 | 0 | 1.67 |
| HW-Segment High Water Mark | | | | | | |
| | 76 | 76 | 0 | 12 | 0 | 2.50 |
| WF-AWR Flush | 19 | 19 | 0 | 11 | 0 | 1.82 |
| TA-Instance Undo | | | | | | |
| | 12 | 12 | 0 | 8 | 0 | 2.50 |
| KO-Multiple Object Checkpoint (fast object checkpoint) | | | | | | |
| | 81 | 81 | 0 | 27 | 0 | .37 |
| FB-Format Block | 8 | 8 | 0 | 8 | 0 | 1.25 |
| JQ-Job Queue | | | | | | |
| | 60 | 60 | 0 | 2 | 0 | 5.00 |
| PG-Global Parameter | 4 | 4 | 0 | 2 | 0 | 5.00 |
| AF-Advisor Framework (task serialization) | | | | | | |
| | 7 | 7 | 0 | 1 | 0 | 10.00 |
| PI-Remote PX Process Spawn Status | 18 | 18 | 0 | 8 | 0 | .00 |
| RS-Reclaimable Space (prevent aging list update) | | | | | | |
| | 4 | 4 | 0 | 4 | 0 | .00 |
| DR-Distributed Recovery | 2 | 2 | 0 | 2 | 0 | .00 |
| PE-Parameter | | | | | | |
| | 8 | 8 | 0 | 2 | 0 | .00 |
| JS-Job Scheduler (job run lock - synchronize) | | | | | | |
| | 2 | 2 | 0 | 1 | 0 | .00 |
| UL-User-defined | 2 | 2 | 0 | 1 | 0 | .00 |
| US-Undo Segment | | | | | | |
| | 1 | 1 | 0 | 1 | 0 | .00 |
| <hr/> | | | | | | |

```
column parameter1 format a15
column parameter2 format a15
column parameter3 format a15
column lock format a8

Select
    substr(name,1,7) as "lock",parameter1,parameter2,parameter3 from v$event_name
where name like 'enq%'
```

Undo Segment Summary DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> Min/Max TR (mins) - Min and Max Tuned Retention (minutes)
 -> STO - Snapshot Too Old count, OOS - Out of Space count
 -> Undo segment block stats:
 -> uS - unexpired Stolen, uR - unexpired Released, uU - unexpired reUsed
 -> eS - expired Stolen, eR - expired Released, eU - expired reUsed

| Undo TS# | Num Blocks | Undo Transactions | Number of Concurcy | Max Qry Len (s) | Max Tx Concyc | Min/Max TR (mins) | STO OOS | uS/uR/uU/
eS/eR/eU |
|----------|------------|-------------------|--------------------|-----------------|---------------|-------------------|---------|-----------------------|
| 2 | .1 | 1,090 | 3 | 1,725 | 18.8 | 42.8 | 0/0 | 0/0/0/0/0/0 |

Undo Segment Stats DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> Most recent 35 Undostat rows, ordered by Time desc

| End Time | Num Blocks | Undo Transactions | Number of Concurcy | Max Qry Len (s) | Max Tx Concyc | Tun Ret | STO/
OOS | uS/uR/uU/
eS/eR/eU |
|--------------|------------|-------------------|--------------------|-----------------|---------------|---------|-------------|-----------------------|
| 04-Aug 12:56 | 14 | 167 | 3 | 890 | | 29 | 0/0 | 0/0/0/0/0/0 |
| 04-Aug 12:46 | 10 | 141 | 3 | 289 | | 19 | 0/0 | 0/0/0/0/0/0 |
| 04-Aug 12:36 | 10 | 163 | 2 | 1,725 | | 43 | 0/0 | 0/0/0/0/0/0 |
| 04-Aug 12:26 | 18 | 240 | 3 | 1,124 | | 33 | 0/0 | 0/0/0/0/0/0 |
| 04-Aug 12:16 | 9 | 133 | 2 | 901 | | 29 | 0/0 | 0/0/0/0/0/0 |
| 04-Aug 12:06 | 88 | 246 | 3 | 300 | | 19 | 0/0 | 0/0/0/0/0/0 |

Latch Activity DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> "Get Requests", "Pct Get Miss" and "Avg Slps/Miss" are statistics for willing-to-wait latch get requests
 -> "NoWait Requests", "Pct NoWait Miss" are for no-wait latch get requests
 -> "Pct Misses" for both should be very close to 0.0

| Latch Name | Requests | Pct | Avg | Wait | Pct | | |
|--------------------------|-----------|------|-----|------------|--------|----------|--------|
| | | Get | Get | Slps /Miss | (s) | NoWait | NoWait |
| | | Miss | | | | Requests | Miss |
| KJC message pool free li | 14,582 | 0.3 | 0.0 | 0 | 15,463 | 0.1 | |
| gc element | 2,414,376 | 0.0 | 0.3 | 2 | 7,880 | 0.0 | |
| gcs resource hash | 1,861,484 | 0.0 | 0.5 | 1 | 6 | 0.0 | |
| virtual circuit queues | 1 | 0.0 | | 0 | 0 | N/A | |

Latch Sleep Breakdown DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> ordered by misses desc

| Latch Name | Get Requests | Get | | Spin |
|-------------------------|--------------|--------|--------|--------|
| | | Misses | Sleeps | Gets |
| cache buffers chains | 8,492,860 | 21,037 | 3 | 21,034 |
| simulator lru latch | 1,823,879 | 12,065 | 311 | 11,774 |
| cache buffers lru chain | 1,190,948 | 6,096 | 352 | 5,799 |
| gc element | 2,414,376 | 767 | 213 | 582 |
| KJCT flow control latch | 443,643 | 735 | 11 | 725 |

Latch Miss Sources DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> only latches with sleeps are shown
 -> ordered by name, sleeps desc

| Latch Name | Where | NoWait | | Waiter |
|-------------------------|--------------|--------|--------|--------|
| | | Misses | Sleeps | Sleeps |
| cache buffers lru chain | kcbzgws_1 | 0 | 248 | 272 |
| gc element | kclnfnndnewm | 0 | 112 | 18 |
| gcs resource hash | kjbasssume | 0 | 88 | 0 |

Mutex Sleep Summary DB/Inst: AULTDB/aultdb1 Snaps: 91-92

No data exists for this section of the report.

Parent Latch Statistics DB/Inst: AULTDB/aultdb1 Snaps: 91-92

No data exists for this section of the report.

Child Latch Statistics DB/Inst: AULTDB/aultdb1 Snaps: 91-92

No data exists for this section of the report.

Segments by Logical Reads DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> Total Logical Reads: 22,791,070
 -> Captured Segments account for 41.8% of Total

| Owner | Tablespace Name | Object Name | Subobject Name | Obj. Type | Logical Reads | %Total |
|-------|-----------------|---------------|----------------|-----------|---------------|--------|
| TPCH | INDEXES | H_ORDERS_IDX1 | | INDEX | 4,294,720 | 18.84 |
| TPCH | DATA | H_LINEITEM | | TABLE | 2,117,568 | 9.29 |
| TPCH | DATA | H_ORDER | | TABLE | 1,017,136 | 4.46 |
| TPCH | INDEXES | SUPPLIER_IDX1 | | INDEX | 626,848 | 2.75 |
| TPCH | DATA | H_SUPPLIER | | TABLE | 620,432 | 2.72 |

Segments by Physical Reads DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> Total Physical Reads: 9,773,380
 -> Captured Segments account for 39.3% of Total

| Owner | Tablespace Name | Object Name | Subobject Name | Obj. Type | Physical Reads | %Total |
|------------------------------------|-----------------|-------------|----------------|-----------|----------------|--------|
| TPCH | DATA | H_LINEITEM | | TABLE | 2,107,980 | 21.57 |
| TPCH | DATA | H_ORDER | | TABLE | 894,131 | 9.15 |
| ** UNAVAIL ** UNAVAIL ** UNAVAI ** | AILABLE | ** UNDEF | | | 511,994 | 5.24 |
| TPCH | DATA | H_PART | | TABLE | 123,676 | 1.27 |
| TPCH | DATA | H_PARTSUPP | | TABLE | 117,400 | 1.20 |

Segments by Row Lock Waits DB/Inst: AULTDB/aultdb1 Snaps: 91-92

No data exists for this section of the report.

Segments by ITL Waits DB/Inst: AULTDB/aultdb1 Snaps: 91-92

No data exists for this section of the report.

Segments by Buffer Busy Waits DB/Inst: AULTDB/aultdb1 Snaps: 91-92

No data exists for this section of the report.

Segments by Global Cache Buffer Busy DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> % of Capture shows % of GC Buffer Busy for each top segment compared
 -> with GC Buffer Busy for all segments captured by the Snapshot

| Owner | Tablespace Name | Object Name | Subobject Name | Obj. Type | GC Buffer Busy | % of Capture |
|------------------------------------|-----------------|---------------|----------------|-----------|----------------|--------------|
| ** UNAVAIL ** UNAVAIL ** UNAVAI ** | AILABLE | ** UNDEF | | | 9 | 81.82 |
| TPCH | INDEXES | H_ORDERS_IDX1 | | INDEX | 1 | 9.09 |
| TPCH | INDEXES | PARTSUPP_IDX1 | | INDEX | 1 | 9.09 |

Segments by CR Blocks Received DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> Total CR Blocks Received: 361
 -> Captured Segments account for 35.5% of Total

| Owner | Tablespace Name | Object Name | Subobject Name | Obj. Type | CR | |
|--------|-----------------|----------------------|----------------|-----------|-----------------|--------|
| | | | | | Blocks Received | %Total |
| SYS | SYSTEM | JOB\$ | | TABLE | 22 | 6.09 |
| SYS | SYSAUX | SMON_SCN_TIME | | TABLE | 21 | 5.82 |
| SYSMAN | SYSAUX | MGMT_SYSTEM_PERFORMA | | TABLE | 12 | 3.32 |
| SYSMAN | SYSAUX | MGMT_SYSTEM_PERF_LOG | | INDEX | 12 | 3.32 |
| SYSMAN | SYSAUX | MGMT_TASK_QTABLE | | TABLE | 12 | 3.32 |

Segments by Current Blocks Received DB/Inst: AULTDB/aultdb1 Snaps: 91-92
-> Total Current Blocks Received: 95,445
-> Captured Segments account for 99.9% of Total

| Owner | Tablespace Name | Object Name | Subobject Name | Obj. Type | Current | |
|------------|-----------------|---------------|------------------|-----------|-----------------|--------|
| | | | | | Blocks Received | %Total |
| TPCH | INDEXES | H_ORDERS_IDX1 | | INDEX | 65,524 | 68.65 |
| TPCH | DATA | H_ORDER | | TABLE | 24,149 | 25.30 |
| TPCH | DATA | H_SUPPLIER | | TABLE | 2,232 | 2.34 |
| SYS | SYSTEM | TAB\$ | | TABLE | 996 | 1.04 |
| ** UNAVAIL | ** UNAVAIL | ** UNAVA ** | AILABLE ** UNDEF | | 776 | .81 |

Dictionary Cache Stats DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> "Pct Misses" should be very low (< 2% in most cases)
 -> "Final Usage" is the number of cache entries being used

| Cache | Get Requests | Pct Miss | Scan Reqs | Pct Miss | Mod Reqs | Final Usage |
|----------------------|--------------|----------|-----------|----------|----------|-------------|
| dc_awr_control | 63 | 3.2 | 0 | N/A | 0 | 1 |
| dc_database_links | 2 | 0.0 | 0 | N/A | 0 | 1 |
| dc_files | 8 | 0.0 | 0 | N/A | 0 | 8 |
| dc_global_oids | 2,826 | 0.2 | 0 | N/A | 0 | 133 |
| dc_histogram_data | 1,151 | 11.6 | 0 | N/A | 0 | 750 |
| dc_histogram_defs | 3,213 | 5.6 | 0 | N/A | 0 | 3,460 |
| dc_object_grants | 484 | 0.0 | 0 | N/A | 0 | 17 |
| dc_objects | 7,172 | 1.0 | 0 | N/A | 17 | 2,203 |
| dc_profiles | 64 | 0.0 | 0 | N/A | 0 | 1 |
| dc_rollback_segments | 850 | 0.0 | 0 | N/A | 0 | 22 |
| dc_segments | 1,020 | 5.9 | 0 | N/A | 4 | 728 |
| dc_sequences | 13 | 30.8 | 0 | N/A | 13 | 0 |
| dc tablespaces | 9,757 | 0.0 | 0 | N/A | 0 | 9 |
| dc_users | 13,294 | 0.0 | 0 | N/A | 0 | 142 |
| global database name | 4,485 | 0.0 | 0 | N/A | 0 | 1 |
| outstanding_alerts | 52 | 69.2 | 0 | N/A | 2 | 1 |

Dictionary Cache Stats (RAC) DB/Inst: AULTDB/aultdb1 Snaps: 91-92

| Cache | GES Requests | GES Conflicts | GES Releases |
|--------------------|--------------|---------------|--------------|
| dc_awr_control | 2 | 2 | 0 |
| dc_global_oids | 5 | 0 | 0 |
| dc_histogram_defs | 181 | 0 | 0 |
| dc_objects | 71 | 0 | 0 |
| dc_segments | 68 | 5 | 0 |
| dc_sequences | 26 | 5 | 0 |
| dc tablespaces | 1 | 0 | 0 |
| dc_users | 5 | 0 | 0 |
| outstanding_alerts | 100 | 36 | 0 |

Library Cache Activity DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> "Pct Misses" should be very low

| Namespace | Get Requests | Pct Miss | Pin Requests | Pct Miss | Reloads | Invali-dations |
|-----------------|--------------|----------|--------------|----------|---------|----------------|
| BODY | 1,514 | 0.0 | 1,858 | 0.2 | 4 | 0 |
| CLUSTER | 44 | 0.0 | 16 | 0.0 | 0 | 0 |
| INDEX | 2 | 0.0 | 2 | 0.0 | 0 | 0 |
| JAVA DATA | 2 | 0.0 | 0 | N/A | 0 | 0 |
| SQL AREA | 2,246 | 1.5 | 17,091 | 2.5 | 121 | 6 |
| TABLE/PROCEDURE | 12,745 | 0.1 | 16,155 | 1.4 | 166 | 0 |
| TRIGGER | 376 | 0.0 | 423 | 0.0 | 0 | 0 |

Library Cache Activity (RAC) DB/Inst: AULTDB/aultdb1 Snaps: 91-92

| Namespace | GES Lock Requests | GES Pin Requests | GES Pin Releases | GES Inval Requests | GES Invali-dations |
|-----------------|-------------------|------------------|------------------|--------------------|--------------------|
| CLUSTER | 16 | 16 | 0 | 0 | 0 |
| INDEX | 2 | 2 | 0 | 0 | 0 |
| TABLE/PROCEDURE | 4,553 | 15,492 | 0 | 0 | 0 |

Dictionary Cache Stats DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> "Pct Misses" should be very low (< 2% in most cases)
 -> "Final Usage" is the number of cache entries being used

| Cache | Get Requests | Pct Miss | Scan Reqs | Pct Miss | Mod Reqs | Final Usage |
|----------------------|--------------|----------|-----------|----------|----------|-------------|
| dc_global_oids | 2,826 | 0.2 | 0 | N/A | 0 | 133 |
| dc_histogram_data | 1,151 | 11.6 | 0 | N/A | 0 | 750 |
| dc_histogram_defs | 3,213 | 5.6 | 0 | N/A | 0 | 3,460 |
| dc_objects | 7,172 | 1.0 | 0 | N/A | 17 | 2,203 |
| dc_segments | 1,020 | 5.9 | 0 | N/A | 4 | 728 |
| dc tablespaces | 9,757 | 0.0 | 0 | N/A | 0 | 9 |
| dc_users | 13,294 | 0.0 | 0 | N/A | 0 | 142 |
| global database name | 4,485 | 0.0 | 0 | N/A | 0 | 1 |

Dictionary Cache Stats (RAC) DB/Inst: AULTDB/aultdb1 Snaps: 91-92

| Cache | GES Requests | GES Conflicts | GES Releases |
|-------|--------------|---------------|--------------|
| | | | |

Library Cache Activity DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> "Pct Misses" should be very low

| Namespace | Get Requests | Pct Miss | Pin Requests | Pct Miss | Reloading | Inval- |
|-----------------|--------------|----------|--------------|----------|-----------|---------|
| | | | | | | dations |
| BODY | 1,514 | 0.0 | 1,858 | 0.2 | 4 | 0 |
| SQL AREA | 2,246 | 1.5 | 17,091 | 2.5 | 121 | 6 |
| TABLE/PROCEDURE | 12,745 | 0.1 | 16,155 | 1.4 | 166 | 0 |

Library Cache Activity (RAC) DB/Inst: AULTDB/aultdb1 Snaps: 91-92

| Namespace | GES Lock Requests | GES Pin Requests | GES Pin Releases | GES Inval Requests | GES Inval dations |
|-----------------|-------------------|------------------|------------------|--------------------|-------------------|
| | | | | | |
| TABLE/PROCEDURE | 4,553 | 15,492 | 0 | 0 | 0 |

Memory Dynamic Components DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> Min/Max sizes since instance startup
 -> Oper Types/Modes: INITializing, GROW, SHRink, STAtic, IMMEDIATE, DEFERred
 -> ordered by Component

| Component | Begin Snap Size (Mb) | Current Size (Mb) | Min Size (Mb) | Max Size (Mb) | Oper Count | Last Op Typ/Mod |
|-----------------|----------------------|-------------------|---------------|---------------|------------|-----------------|
| ASM Buffer Cach | .00 | .00 | .00 | .00 | 0 | STA/ |
| DEFAULT 16K buf | .00 | .00 | .00 | .00 | 0 | STA/ |
| DEFAULT 2K buff | .00 | .00 | .00 | .00 | 0 | STA/ |
| DEFAULT 32K buf | .00 | .00 | .00 | .00 | 0 | STA/ |
| DEFAULT 4K buff | .00 | .00 | .00 | .00 | 0 | STA/ |
| DEFAULT 8K buff | .00 | .00 | .00 | .00 | 0 | STA/ |
| DEFAULT buffer | 1,312.00 | 1,312.00 | 1,296.00 | 1,328.00 | 2 | GRO/DEF |
| KEEP buffer cac | .00 | .00 | .00 | .00 | 0 | STA/ |
| PGA Target | 512.00 | 512.00 | 512.00 | 512.00 | 0 | STA/ |
| RECYCLE buffer | .00 | .00 | .00 | .00 | 0 | STA/ |
| SGA Target | 1,584.00 | 1,584.00 | 1,584.00 | 1,584.00 | 0 | STA/ |
| Shared IO Pool | .00 | .00 | .00 | .00 | 0 | STA/ |
| java pool | 16.00 | 16.00 | 16.00 | 16.00 | 0 | STA/ |
| large pool | 16.00 | 16.00 | 16.00 | 16.00 | 0 | STA/ |
| shared pool | 224.00 | 224.00 | 208.00 | 240.00 | 2 | SHR/DEF |
| streams pool | .00 | .00 | .00 | .00 | 0 | STA/ |

Memory Resize Operations Summary DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> Resizes, Grows, Shrinks - Operations captured by AWR
 if there are operations on the same component for the same
 operation\_type, target\_size, and with the same start\_time
 only one operation is captured
 -> ordered by Component

| Component | Min Size (Mb) | Max Size (Mb) | Avg Size (Mb) | Re-Sizes | Grows | Shrink |
|----------------|---------------|---------------|---------------|----------|-------|--------|
| DEFAULT buffer | 1,296.00 | 1,312.00 | 1,304.00 | 2 | 1 | 1 |
| shared pool | 224.00 | 240.00 | 232.00 | 2 | 1 | 1 |

Memory Resize Ops DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> Oper Types/Modes: INITializing, GROW, SHRink, STAtic, IMMEDIATE, DEFERred
 Delta : change in size of the component
 Target Delta: displayed only if final size <> target\_size
 -> Status: COMplete/CANcelled/INActive/PENding/ERRor
 -> ordered by start\_time desc, component

| Start | Ela (s) | Component | Oper Typ/Mod | Init Size (M) | Target Delta | Final Delta (M) | Sta |
|----------------|---------|-----------|--------------|---------------|--------------|-----------------|-----------|
| 08/04 12:39:06 | 0 | bufcache | GRO/DEF | 1,296 | 16 | N/A | 1,312 COM |
| 08/04 12:39:06 | 0 | shared | SHR/DEF | 240 | -16 | N/A | 224 COM |
| 08/04 12:02:59 | 2 | bufcache | SHR/DEF | 1,312 | -16 | N/A | 1,296 COM |
| 08/04 12:02:59 | 2 | shared | GRO/DEF | 224 | 16 | N/A | 240 COM |

Process Memory Summary DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> B: Begin snap E: End snap
 -> All rows below contain absolute values (i.e. not diffed over the interval)
 -> Max Alloc is Maximum PGA Allocation size at snapshot time
 -> Hist Max Alloc is the Historical Max Allocation for still-connected processes
 -> ordered by Begin/End snapshot, Alloc (MB) desc

| Category | Alloc
(MB) | Used
(MB) | Hist | | | | | |
|----------|---------------|--------------|--------------|------------------|--------------|-------|-------------|--------------|
| | | | Avg
Alloc | Std Dev
Alloc | Max
Alloc | Alloc | Num
Proc | Num
Alloc |
| | | | | | | | | |
| B Other | 153.1 | N/A | 3.5 | 4.9 | 24 | 24 | 44 | 44 |
| Freeable | 13.3 | .0 | .9 | .9 | 4 | N/A | 14 | 14 |
| JAVA | 1.6 | 1.6 | .8 | 1.2 | 2 | 2 | 2 | 2 |
| SQL | 1.2 | .5 | .1 | .1 | 0 | 3 | 20 | 13 |
| PL/SQL | .2 | .1 | .0 | .0 | 0 | 0 | 42 | 42 |
| E Other | 175.3 | N/A | 3.5 | 4.6 | 24 | 24 | 50 | 50 |
| Freeable | 101.5 | .0 | 5.3 | 10.1 | 28 | N/A | 19 | 19 |
| SQL | 23.2 | 22.5 | .9 | 1.7 | 5 | 166 | 26 | 19 |
| JAVA | 1.6 | 1.6 | .8 | 1.2 | 2 | 2 | 2 | 2 |
| PL/SQL | .2 | .1 | .0 | .0 | 0 | 0 | 48 | 48 |

SGA Memory Summary DB/Inst: AULTDB/aultdb1 Snaps: 91-92

| SGA regions | Begin Size (Bytes) | End Size (Bytes) |
|------------------|--------------------|------------------|
| | | (if different) |
| Database Buffers | | 1,375,731,712 |
| Fixed Size | | 1,300,968 |
| Redo Buffers | | 10,858,496 |
| Variable Size | | 671,090,200 |
| sum | | 2,058,981,376 |

SGA breakdown difference

DB/Inst: AULTDB/aultdb1 Snaps: 91-92

-> ordered by Pool, Name

-> N/A value for Begin MB or End MB indicates the size of that Pool/Name was insignificant, or zero in that snapshot

| Pool | Name | Begin MB | End MB | % Diff |
|--------|---------------------------|----------|---------|--------|
| java | free memory | 7.7 | 7.7 | 0.00 |
| java | joxlod exec hp | 8.1 | 8.1 | 0.00 |
| java | joxs heap | .3 | .3 | 0.00 |
| large | ASM map operations hashta | .2 | .2 | 0.00 |
| large | PX msg pool | .3 | .4 | 41.67 |
| large | free memory | 15.5 | 15.4 | -0.79 |
| shared | ASH buffers | 4.0 | 4.0 | 0.00 |
| shared | CCursor | 3.4 | 4.0 | 15.27 |
| shared | Heap0: KGL | 4.0 | 4.1 | 1.74 |
| shared | KCB Table Scan Buffer | 4.0 | 4.0 | 0.00 |
| shared | KGL buckets | 3.0 | 3.0 | 0.00 |
| shared | KGL handle | 6.6 | 6.6 | 0.26 |
| shared | KGLS heap | 3.8 | 4.6 | 20.93 |
| shared | KQR L PO | N/A | 2.3 | N/A |
| shared | KQR M PO | 2.8 | 3.0 | 5.87 |
| shared | KSF D SGA I/O b | 4.0 | 4.0 | 0.00 |
| shared | PCursor | 2.6 | 2.6 | 1.61 |
| shared | PL/SQL DIANA | 11.4 | 11.4 | 0.00 |
| shared | PL/SQL MPCODE | 12.8 | 13.0 | 1.53 |
| shared | db_block_hash_buckets | 5.9 | 5.9 | 0.00 |
| shared | dbwriter coalesce buffer | 4.0 | 4.0 | 0.00 |
| shared | free memory | 41.4 | 32.7 | -20.92 |
| shared | gcs resources | 20.2 | 20.2 | 0.00 |
| shared | gcs shadows | 15.7 | 15.7 | 0.00 |
| shared | ges big msg buffers | 4.1 | 4.1 | 0.00 |
| shared | gesblFilter_seg | N/A | 4.0 | N/A |
| shared | row cache | 3.6 | 3.6 | 0.00 |
| shared | sql area | 14.1 | 16.6 | 17.03 |
| shared | type object de | 2.7 | 2.7 | 0.02 |
| | buffer_cache | 1,312.0 | 1,312.0 | 0.00 |
| | fixed_sga | 1.2 | 1.2 | 0.00 |
| | log_buffer | 10.4 | 10.4 | 0.00 |

Streams CPU/IO Usage DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> Streams processes ordered by CPU usage
 -> CPU and I/O Time in micro seconds

| Session Type | CPU Time | User I/O Time | Sys I/O Time |
|---------------------------|-----------|---------------|--------------|
| STREAMS Capture | 2,128,000 | 0 | 0 |
| STREAMS Apply Reader | 609,945 | 2,050 | 1,341 |
| Logminer Builder | 185,835 | 0 | 0 |
| Logminer Preparer | 175,559 | 0 | 0 |
| QMON Slaves | 132,888 | 0 | 0 |
| Logminer Reader | 97,009 | 0 | 887,687 |
| STREAMS Apply Server | 35,580 | 0 | 0 |
| STREAMS Apply Coordinator | 747 | 0 | 0 |
| QMON Coordinator | 454 | 0 | 0 |
| Propagation Sender | 0 | 0 | 0 |

Streams Capture DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> Lag Change should be small or negative (in seconds)

| Capture Name | Captured | | Enqueued | | Pct | Pct | Pct | Pct |
|--------------|------------|------------|------------|---------------|--------------|---------------|------------|-----|
| | Per Second | Per Second | Lag Change | RuleEval Time | Enqueue Time | RedoWait Time | Pause Time | |
| STREAM_CAP | 65 | 39 | 93 | 0 | 23 | 0 | 0 | 71 |

Streams Apply /Inst: AULTDB/aultdb1 Snaps: 91-92
 -> Pct DB is the percentage of all DB transactions that this apply handled
 -> WDEP is the wait for dependency
 -> WCMT is the wait for commit
 -> RBK is rollbacks
 -> MPS is messages per second
 -> TPM is time per message in milli-seconds
 -> Lag Change should be small or negative (in seconds)

| Apply Name | Applied TPS | Pct DB | Pct WDEP | Pct WCMT | Pct RBK | Applied MPS | Dequeue TPM | Apply TPM | Lag Change |
|------------|-------------|--------|----------|----------|---------|-------------|-------------|-----------|------------|
| STREAM_APP | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Buffered Queues /Inst: AULTDB/aultdb1 Snaps: 91-92
 -> The Spill Rate should be very close to zero
 -> The Diff in Percentage Spilled should be very close to zero or negative

| Queue Schema and Name | Incoming per second | Outgoing per second | Spilled per second | Diff Pct Spilled |
|-----------------------|---------------------|---------------------|--------------------|------------------|
| STRMADMIN.REP_CAPTURE | 39 | 39 | 0 | 0 |
| STRMADMIN.REP_DEST_QU | 0 | 0 | 0 | 0 |

Buffered Subscribers /Inst: AULTDB/aultdb1 Snaps: 91-92
 -> All Subscribers should have a zero spill rate

| Subscriber Name | Incoming per second | Outgoing per second | Spilled per second |
|----------------------|---------------------|---------------------|--------------------|
| STREAM_APP | 793 | 793 | 6 |
| STREAM_APP | 0 | 0 | 0 |
| PROXY: "STRMADMIN"." | 391 | 391 | 0 |
| PROXY: CATLIBRAR.ASU | 0 | 0 | 0 |
| PROXY: CATLIBRAR.ASU | -793 | -793 | -6 |

Rule Set /Inst: AULTDB/aultdb1 Snaps: 91-92
 -> Rule Sets ordered by Evaluations

| Ruleset Name | Evals | Fast Evals | SQL Execs | CPU Time | Elapsed Time |
|-------------------------|-------|------------|-----------|----------|--------------|
| STRMADMIN.RULESETS\$_10 | 3,174 | 0 | 3,174 | 1,565 | 2,784 |
| SYS.ALERT_QUE_R | 2 | 0 | 0 | 0 | 0 |
| STRMADMIN.RULESETS\$_6 | 0 | 0 | 0 | 0 | 0 |
| STRMADMIN.RULESETS\$_3 | 0 | 0 | 0 | 0 | 0 |
| STRMADMIN.RULESETS\$_8 | 0 | 0 | 0 | 0 | 0 |

Resource Limit Stats Inst: AULTDB/aultdb1 Snaps: 91-92
-> only rows with Current or Maximum Utilization > 80% of Limit are shown
-> ordered by resource name

| Resource Name | Current Utilization | Maximum Utilization | Initial Allocation | Limit |
|---------------|---------------------|---------------------|--------------------|-------|
| sessions | 51 | 65 | 150 | 150 |

init.ora Parameters DB/Inst: AULTDB/aultdb1 Snaps: 91-92
-> if IP/Public/Source at End snap is different a '\*' is displayed

| Parameter Name | Begin value | End value
(if different) |
|-----------------------------|------------------------------------|-----------------------------|
| audit_file_dest | /home/oracle/app/product/oracle/a | |
| audit_trail | DB | |
| cluster_database | TRUE | |
| cluster_database_instances | 2 | |
| compatible | 11.1.0.0.0 | |
| control_files | +DATA2/aultdb/controlfile/current | |
| db_block_size | 8192 | |
| db_create_file_dest | +DATA2 | |
| db_domain | | |
| db_name | aultdb | |
| db_recovery_file_dest | +DATA2 | |
| db_recovery_file_dest_size | 2147483648 | |
| diagnostic_dest | /home/oracle/app/product/oracle | |
| dispatchers | (PROTOCOL=TCP) (SERVICE=aultdbXDB) | |
| instance_number | 1 | |
| memory_target | 2197815296 | |
| open_cursors | 300 | |
| processes | 150 | |
| remote_listener | LISTENERS_AULTDB | |
| remote_login_passwordfile | EXCLUSIVE | |
| spfile | +DATA/aultdb/spfileaultdb.ora | |
| star_transformation_enabled | TRUE | |
| thread | 1 | |
| undo_tablespace | UNDOTBS1 | |

Global Enqueue Statistics

DB/Inst: AULTDB/aultdb1 Snaps: 91-92

| Statistic | Total | per Second | per Trans |
|------------------------------------|---------|------------|-----------|
| acks for commit broadcast(actual) | 216 | 0.1 | 0.2 |
| acks for commit broadcast(logical) | 242 | 0.1 | 0.2 |
| broadcast msgs on commit(actual) | 700 | 0.2 | 0.6 |
| broadcast msgs on commit(logical) | 701 | 0.2 | 0.6 |
| broadcast msgs on commit(wasted) | 53 | 0.0 | 0.0 |
| gcs assume no cvt | 45,685 | 12.6 | 38.9 |
| gcs blocked converts | 205 | 0.1 | 0.2 |
| gcs blocked cr converts | 238 | 0.1 | 0.2 |
| gcs compatible cr basts (local) | 93,290 | 25.8 | 79.4 |
| gcs dbwr flush pi msgs | 36 | 0.0 | 0.0 |
| gcs dbwr write request msgs | 141 | 0.0 | 0.1 |
| gcs immediate (compatible) conver | 87 | 0.0 | 0.1 |
| gcs immediate (null) converts | 324 | 0.1 | 0.3 |
| gcs immediate cr (compatible) con | 11,512 | 3.2 | 9.8 |
| gcs immediate cr (null) converts | 213,759 | 59.2 | 181.9 |
| gcs indirect ast | 42,995 | 11.9 | 36.6 |
| gcs indirect fg ast | 42,995 | 11.9 | 36.6 |
| gcs msgs process time(ms) | 15,443 | 4.3 | 13.1 |
| gcs msgs received | 549,546 | 152.1 | 467.7 |
| gcs new served by master | 102 | 0.0 | 0.1 |
| gcs pings refused | 8 | 0.0 | 0.0 |
| gcs retry convert request | 16,809 | 4.7 | 14.3 |
| gcs side channel msgs actual | 5,444 | 1.5 | 4.6 |
| gcs side channel msgs logical | 146,320 | 40.5 | 124.5 |
| ges msgs process time(ms) | 344 | 0.1 | 0.3 |
| ges msgs received | 15,248 | 4.2 | 13.0 |
| global posts queue time | 318 | 0.1 | 0.3 |
| global posts queued | 59 | 0.0 | 0.1 |
| global posts requested | 63 | 0.0 | 0.1 |
| global posts sent | 59 | 0.0 | 0.1 |

| | | | |
|-----------------------------------|---------|-------|-------|
| implicit batch messages received | 26,591 | 7.4 | 22.6 |
| implicit batch messages sent | 28,441 | 7.9 | 24.2 |
| messages flow controlled | 2,047 | 0.6 | 1.7 |
| messages queue sent actual | 68,909 | 19.1 | 58.6 |
| messages queue sent logical | 142,750 | 39.5 | 121.5 |
| messages received actual | 236,086 | 65.3 | 200.9 |
| messages received logical | 564,794 | 156.3 | 480.7 |
| messages sent directly | 134,160 | 37.1 | 114.2 |
| messages sent indirectly | 245,700 | 68.0 | 209.1 |
| messages sent not implicit batch | 40,468 | 11.2 | 34.4 |
| messages sent pbatched | 306,413 | 84.8 | 260.8 |
| msgs causing lmd to send msgs | 5,845 | 1.6 | 5.0 |
| msgs causing lms(s) to send msgs | 19,392 | 5.4 | 16.5 |
| msgs received queue time (ms) | 41,000 | 11.3 | 34.9 |
| msgs received queued | 564,808 | 156.3 | 480.7 |
| msgs sent queue time (ms) | 146,886 | 40.7 | 125.0 |
| msgs sent queue time on ksxp (ms) | 322,477 | 89.2 | 274.4 |
| msgs sent queued | 139,264 | 38.5 | 118.5 |
| msgs sent queued on ksxp | 243,401 | 67.4 | 207.1 |
| process batch messages received | 50,530 | 14.0 | 43.0 |
| process batch messages sent | 50,185 | 13.9 | 42.7 |

Global CR Served Stats

DB/Inst: AULTDB/aultdb1 Snaps: 91-92

| Statistic | Total |
|------------------------|-------|
| CR Block Requests | 486 |
| CURRENT Block Requests | 36 |
| Data Block Requests | 486 |
| Undo Block Requests | 0 |
| TX Block Requests | 7 |
| Current Results | 522 |
| Fairness Down Converts | 78 |
| Fairness Clears | 10 |
| Flushes | 4 |

Global CURRENT Served Stats DB/Inst: AULTDB/aultdb1 Snaps: 91-92

-> Pins = CURRENT Block Pin Operations

-> Flushes = Redo Flush before CURRENT Block Served Operations

-> Writes = CURRENT Block Fusion Write Operations

| Statistic | Total | % <1ms | % <10ms | % <100ms | % <1s | % <10s |
|-----------|--------|--------|---------|----------|-------|--------|
| Pins | 93,484 | 99.95 | 0.00 | 0.05 | 0.00 | 0.00 |
| Flushes | 1 | 0.00 | 0.00 | 100.00 | 0.00 | 0.00 |
| Writes | 43 | 0.00 | 4.65 | 74.42 | 18.60 | 2.33 |

Global Cache Transfer Stats DB/Inst: AULTDB/aultdb1 Snaps: 91-92

-> Immediate(Immed) - Block Transfer NOT impacted by Remote Processing Delays

-> Busy (Busy) - Block Transfer impacted by Remote Contention

-> Congested (Congst) - Block Transfer impacted by Remote System Load

-> ordered by CR + Current Blocks Received desc

| Inst Block
No Class | CR | | | | Current | | | |
|------------------------|--------------------|------------|-----------|-------------|--------------------|------------|-----------|-------------|
| | Blocks
Received | %
Immed | %
Busy | %
Congst | Blocks
Received | %
Immed | %
Busy | %
Congst |
| 2 data block | 150 | 98.0 | 2.0 | .0 | 95,402 | 99.8 | .0 | .2 |
| 2 undo header | 200 | 100.0 | .0 | .0 | 3 | 100.0 | .0 | .0 |
| 2 Others | 10 | 100.0 | .0 | .0 | 24 | 100.0 | .0 | .0 |

Global Cache Transfer Times (ms) DB/Inst: AULTDB/aultdb1 Snaps: 91-92

-> Avg Time - average time of all blocks (Immed, Busy, Congst) in ms

-> Immed, Busy, Congst - Average times in ms

-> ordered by CR + Current Blocks Received desc

| Inst Block
No Class | CR Avg Time (ms) | | | | Current Avg Time (ms) | | | |
|------------------------|------------------|-------|------|--------|-----------------------|-------|------|--------|
| | All | Immed | Busy | Congst | All | Immed | Busy | Congst |
| 2 data blo | 2.5 | 1.9 | 27.6 | N/A | 1.8 | 1.7 | N/A | 9.1 |
| 2 undo hea | 1.4 | 1.4 | N/A | N/A | 1.6 | 1.6 | N/A | N/A |
| 2 others | 1.4 | 1.4 | N/A | N/A | 1.4 | 1.4 | N/A | N/A |
| 2 undo blo | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A |

Global Cache Transfer (Immediate) DB/Inst: AULTDB/aultdb1 Snaps: 91-92

-> Immediate(Immed) - Block Transfer NOT impacted by Remote Processing Delays

-> % of Blocks Received requiring 2 or 3 hops

-> ordered by CR + Current Blocks Received desc

| Src Block
Inst Class | CR | | | | Current | | | |
|-------------------------|----------------|--------------------|-----------|-----------|----------------|--------------------|-----------|-----------|
| | Blocks
Lost | Blocks
Received | %
2hop | %
3hop | Blocks
Lost | Blocks
Received | %
2hop | %
3hop |
| 2 data blo | 0 | 147 | 100.0 | 0.0 | 95,204 | 100.0 | 0.0 | 0.0 |
| 2 undo hea | 0 | 200 | 100.0 | 0.0 | 3 | 100.0 | 0.0 | 0.0 |
| 2 others | 0 | 10 | 100.0 | 0.0 | 24 | 100.0 | 0.0 | 0.0 |
| 2 undo blo | 0 | 0 | N/A | N/A | 0 | N/A | N/A | N/A |

Global Cache Times (Immediate) DB/Inst: AULTDB/aultdb1 Snaps: 91-92

-> Blocks Lost, 2-hop and 3-hop Average times in (ms)

-> ordered by CR + Current Blocks Received desc

| Src Block
Inst Class | Lost
Time | CR Avg Time (ms) | | | Current Avg Time (ms) | | |
|-------------------------|--------------|------------------|------|------|-----------------------|------|------|
| | | Immed | 2hop | 3hop | Immed | 2hop | 3hop |
| 2 data blo | | 1.9 | 1.9 | N/A | 1.7 | 1.7 | N/A |
| 2 undo hea | | 1.4 | 1.4 | N/A | 1.6 | 1.6 | N/A |
| 2 others | | 1.4 | 1.4 | N/A | 1.4 | 1.4 | N/A |
| 2 undo blo | | N/A | N/A | N/A | N/A | N/A | N/A |

Interconnect Ping Latency Stats DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> Ping latency of the roundtrip of a message from this instance to target in
 -> The target instance is identified by an instance number.
 -> Average and standard deviation of ping latency is given in milliseconds
 -> for message sizes of 500 bytes and 8K.
 -> Note that latency of a message from the instance to itself is used as
 -> control, since message latency can include wait for CPU

| Instance | Count | Target 500B Pin Avg Latency | Stddev | 8K Ping Avg Latency | Stddev |
|----------|-------|-----------------------------|----------|---------------------|--------|
| | | 500B msg | 500B msg | | 8K msg |
| 1 | 360 | .64 | 1.04 | 360 | .63 |
| 2 | 360 | 1.39 | 2.43 | 360 | 2.16 |

Interconnect Throughput by Client DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> Throughput of interconnect usage by major consumers.
 -> All throughput numbers are megabytes per second

| Used By | Send | Receive |
|----------------|------------|------------|
| | Mbytes/sec | Mbytes/sec |
| Global Cache | .20 | .21 |
| Parallel Query | .35 | .39 |
| DB Locks | .03 | .03 |
| DB Streams | .00 | .00 |
| Other | .00 | .00 |

Interconnect Device Statistics DB/Inst: AULTDB/aultdb1 Snaps: 91-92
 -> Throughput and errors of interconnect devices (at OS level).
 -> All throughput numbers are megabytes per second

| Device Name | IP Address | Public Source | | | | |
|-------------|------------|---------------|---------------------------|---------|--------|---------|
| | | Send | Send | Send | Buffer | Carrier |
| Mbytes/sec | Errors | Dropped | Overrun | Lost | | |
| | | Receive | Receive | Receive | Buffer | Frame |
| Receive | Receive | Receive | Buffer | Frame | | |
| Mbytes/sec | Errors | Dropped | Overrun | Errors | | |
| eth0 | 11.1.1.1 | NO | Oracle Cluster Repository | | | |
| | .77 | 0 | 0 | 0 | 0 | |
| | .82 | 1 | 0 | 0 | 1 | |

```
$ crsctl check has
CRS-4638: Oracle High Availability Services is online

$ crsctl check crs
CRS-4638: Oracle High Availability Services is online
CRS-4537: Cluster Ready Services is online
CRS-4529: Cluster Synchronization Services is online
CRS-4533: Event Manager is online
```

```
$ crsctl query crs activeversion
Oracle Clusterware active version on the cluster is [11.2.0.4.0]

$ crsctl query crs releaseversion
Oracle High Availability Services release version on the local node is [11.2.0.4.0]

$ crsctl query crs softwareversion
Oracle Clusterware version on node [oe01db01] is [11.2.0.4.0]

$ crsctl query crs softwareversion -all
Oracle Clusterware version on node [oe01db01] is [11.2.0.4.0]
Oracle Clusterware version on node [oe01db02] is [11.2.0.4.0]
Oracle Clusterware version on node [oe01db03] is [11.2.0.4.0]
Oracle Clusterware version on node [oe01db04] is [11.2.0.4.0]
Oracle Clusterware version on node [oe01db05] is [11.2.0.4.0]
Oracle Clusterware version on node [oe01db06] is [11.2.0.4.0]
Oracle Clusterware version on node [oe01db07] is [11.2.0.4.0]
Oracle Clusterware version on node [oe01db08] is [11.2.0.4.0]

$ crsctl query css votedisk
## STATE File Universal Id           File Name           Disk group
--- -----
1. ONLINE 7cb8478916a84f10bf7dbb336ca68601 (o/192.168.10.5/DBFS_DG_CD_02_oe01cel01) [DBFS_DG]
2. ONLINE 8eb8c6e255534f84bfb1da0194b845bb (o/192.168.10.6/DBFS_DG_CD_02_oe01cel02) [DBFS_DG]
3. ONLINE 7f378e868c094fb0bfc38af465fc64f4 (o/192.168.10.7/DBFS_DG_CD_02_oe01cel03) [DBFS_DG]
Located 3 voting disk(s).

$ crsctl query crs administrator
CRS Administrator List: *
```

```
$ crsctl status serverpool -p
NAME=Free
IMPORTANCE=0
MIN_SIZE=0
MAX_SIZE=-1
SERVER_NAMES=
PARENT_POOLS=
EXCLUSIVE_POOLS=
ACL=owner:oracle:rwx,pgrp:oinstall:rwx,other::r-x

NAME=Generic
IMPORTANCE=0
MIN_SIZE=0
MAX_SIZE=-1
SERVER_NAMES=oe01db01 oe01db02
PARENT_POOLS=
EXCLUSIVE_POOLS=
ACL=owner:oracle:r-x,pgrp:oinstall:r-x,other::r-x

NAME=ora.dbm
IMPORTANCE=1
MIN_SIZE=0
MAX_SIZE=-1
SERVER_NAMES=oe01db01 oe01db02
PARENT_POOLS=Generic
EXCLUSIVE_POOLS=
ACL=owner:oracle:rwx,pgrp:oinstall:rwx,other::r--

NAME=ora.sri
IMPORTANCE=1
MIN_SIZE=0
MAX_SIZE=-1
SERVER_NAMES=oe01db01 oe01db02
PARENT_POOLS=Generic
EXCLUSIVE_POOLS=
ACL=owner:oracle:rwx,pgrp:oinstall:rwx,other::r--

$ crsctl status serverpool ora.dbm -p
NAME=ora.dbm
IMPORTANCE=1
MIN_SIZE=0
MAX_SIZE=-1
SERVER_NAMES=oe01db01 oe01db02
PARENT_POOLS=Generic
EXCLUSIVE_POOLS=
ACL=owner:oracle:rwx,pgrp:oinstall:rwx,other::r--

$ crsctl get cluster mode status
Cluster is running in "standard" mode

$ crsctl get node role config
Node 'exaldb01' configured role is 'hub'
```

```
$ srvctl status server -n oe01db01,oe01db02,oe01db03,oe01db04
Server name: oe01db01
Server state: ONLINE
Server name: oe01db02
Server state: ONLINE
Server name: oe01db03
Server state: ONLINE
Server name: oe01db04
Server state: ONLINE

$ srvctl status database -d dbm
Instance dbm1 is running on node oe01db01
Instance dbm2 is running on node oe01db02

$ srvctl status instance -d dbm -i dbm1
Instance dbm1 is running on node oe01db01

$ srvctl status nodeapps
VIP oe01db01-vip is enabled
VIP oe01db01-vip is running on node: oe01db01
VIP oe01db02-vip is enabled
VIP oe01db02-vip is running on node: oe01db02
VIP oe01db03-vip is enabled
VIP oe01db03-vip is running on node: oe01db03
VIP oe01db04-vip is enabled
VIP oe01db04-vip is running on node: oe01db04
VIP oe01db05-vip is enabled
Network is enabled
Network is running on node: oe01db01
Network is running on node: oe01db02
Network is running on node: oe01db03
Network is running on node: oe01db04
GSD is disabled
GSD is not running on node: oe01db01
GSD is not running on node: oe01db02
GSD is not running on node: oe01db03
GSD is not running on node: oe01db04
ONS is enabled
ONS daemon is running on node: oe01db01
ONS daemon is running on node: oe01db02
ONS daemon is running on node: oe01db03
ONS daemon is running on node: oe01db04

$ srvctl status nodeapps -n oe01db01
VIP oe01db01-vip is enabled
VIP oe01db01-vip is running on node: oe01db01
Network is enabled
Network is running on node: oe01db01
GSD is disabled
GSD is not running on node: oe01db01
ONS is enabled
ONS daemon is running on node: oe01db01

$ srvctl status asm
ASM is running on oe01db01, oe01db02, oe01db03, oe01db04

$ srvctl status diskgroup -g DATA1
Disk Group DATA1 is running on oe01db01, oe01db02, oe01db03, oe01db04

$ srvctl status listener
Listener LISTENER is enabled
Listener LISTENER is running on node(s): oe01db01, oe01db02, oe01db03, oe01db04
```

```
$ srvctl status listener -n oe01db01
Listener LISTENER is enabled on node(s): oe01db01
Listener LISTENER is running on node(s): oe01db01

$ srvctl status scan
SCAN VIP scan1 is enabled
SCAN VIP scan1 is running on node oe01db02
SCAN VIP scan2 is enabled
SCAN VIP scan2 is running on node oe01db01
SCAN VIP scan3 is enabled
SCAN VIP scan3 is running on node oe01db03

$ srvctl status scan -i 1
SCAN VIP scan1 is enabled
SCAN VIP scan1 is running on node oe01db02

$ srvctl status scan_listener
SCAN Listener LISTENER_SCAN1 is enabled
SCAN listener LISTENER_SCAN1 is running on node oe01db02
SCAN Listener LISTENER_SCAN2 is enabled
SCAN listener LISTENER_SCAN2 is running on node oe01db01
SCAN Listener LISTENER_SCAN3 is enabled
SCAN listener LISTENER_SCAN3 is running on node oe01db03

$ srvctl status scan_listener -i 1
SCAN Listener LISTENER_SCAN1 is enabled
SCAN listener LISTENER_SCAN1 is running on node oe01db02

$ srvctl status vip -n oe01db01
VIP oe0101-vip is enabled
VIP oe0101-vip is running on node: oe01db01

$ srvctl status vip -i oe0101-vip-vip
PRKO-2167 : VIP oe0101-vip-vip does not exist.
```

```
$ srvctl config database -d dbm
Database unique name: dbm
Database name: dbm
Oracle home: /u01/app/oracle/product/11.2.0.4/dbhome_1
Oracle user: oracle
Spfile: +DATA_OE01/dbm/spfiledbm.ora
Domain: at-rockside.lab
Start options: open
Stop options: immediate
Database role: PRIMARY
Management policy: AUTOMATIC
Server pools: dbm
Database instances: dbm1,dbm2
Disk Groups: DATA_OE01,RECO_OE01
Mount point paths:
Services:
Type: RAC
Database is administrator managed

$ srvctl config nodeapps -n oe01db01
-n <node_name> option has been deprecated.
Network exists: 1/174.17.40.0/255.255.255.0/bondeth0, type static
VIP exists: /oe0101-vip/174.17.40.4/174.17.40.0/255.255.255.0/bondeth0,
           hosting node oe01db01
GSD exists
ONS exists: Local port 6100, remote port 6200, EM port 2016

$ srvctl config listener -l LISTENER -a
Name: LISTENER
Network: 1, Owner: oracle
Home: <CRS home>
/u01/app/11.2.0.4/grid on node(s) oe01db02,oe01db01
End points: TCP:1521
```

```
$ ocrconfig -showbackup

oe01db01 2014/09/08 14:41:23 /u01/app/11.2.0.3/grid/cdata/oe01-cluster/backup00.ocr
oe01db01 2014/09/08 10:41:23 /u01/app/11.2.0.3/grid/cdata/oe01-cluster/backup01.ocr
oe01db01 2014/09/08 06:41:23 /u01/app/11.2.0.3/grid/cdata/oe01-cluster/backup02.ocr
oe01db01 2014/09/07 02:41:21 /u01/app/11.2.0.3/grid/cdata/oe01-cluster/day.ocr
oe01db01 2014/08/28 14:41:06 /u01/app/11.2.0.4/grid/cdata/oe01-cluster/week.ocr
oe01db01 2013/02/26 17:20:00 /u01/app/11.2.0.4/grid/cdata/oe01-cluster/backup_20130226_172000.ocr

$ olsnodes -s
oe01db01 Active
oe01db02 Active

$ olsnodes -n
oe01db01 1
oe01db02 2

$ ocrcheck
Status of Oracle Cluster Registry is as follows :
Version : 3
Total space (kbytes) : 262120
Used space (kbytes) : 3036
Available space (kbytes) : 259084
ID : 1278623030
Device/File Name : +DBFS_DG
Device/File integrity check succeeded
Device/File not configured
Device/File not configured
Device/File not configured
Cluster registry integrity check succeeded
ocrcheck Logical corruption check succeeded

$ ocrcheck -local
Status of Oracle Local Registry is as follows :
Version : 3
Total space (kbytes) : 262120
Used space (kbytes) : 2684
Available space (kbytes) : 259436
ID : 957270436
Device/File Name : /u01/app/11.2.0.4/grid/cdata/oe01db01.olr

Device/File integrity check succeeded
Local registry integrity check succeeded
Logical corruption check succeeded
```

```
DECLARE
    sqlstmt  CLOB;
    taskname VARCHAR2(30);
BEGIN
    sqlstmt := 'select r.region,s.totalsales from sales s, regions r where
               s.region_no = r.region_no group by r.region_no';

    taskname := DBMS_SQLTUNE.CREATE_TUNING_TASK (
        sql_text      => sqlstmt,
        bind_list    => sql_binds(anydata.ConvertNumber(100)),
        user_name    => 'tfm',
        scope        => 'COMPREHENSIVE',
        time_limit   => 60,
        task_name    => 'totalsales_sql_sta_task',
        description  => 'Individual Query - SQL Tuning Advisor Task');
END;
/
BEGIN
    DBMS_SQLTUNE.EXECUTE_TUNING_TASK(task_name=>'totalsales_sql_sta_task');
END;
/
SET LONGCHUNKSIZE 1500
SET LINESIZE 250
SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK( 'totalsales_sql_sta_task' ) FROM DUAL;
```

```

DECLARE
    taskname varchar2(30)      := 'SQL_ACCESS_9940';
    task_desc varchar2(256)     := 'SQL Access Advisor';
    task_or_template varchar2(30) := 'SQLACCESS_EMTASK';
    task_id number              := 0;
    num_found number;
    sts_name varchar2(256)      := 'STS_CUSTOM_9940';
    sts_owner varchar2(30)       := 'SYS';
BEGIN
Create the SQL Access Advisor Task dbms_advisor.create_task(DBMS_ADVISOR.SQLACCESS_ADVISOR,
task_id,taskname,task_desc,task_or_template);

-- Reset the SQL Access Advisor Task
dbms_advisor.reset_task(taskname);

-- Delete Previous STS Workload Task Link
select count(*) into num_found from user_advisor_sqla_wk_map where
task_name = taskname and workload_name = sts_name;
IF num_found > 0 THEN
dbms_advisor.delete_sts_ref(taskname, sts_owner, sts_name);
END IF;

-- Link STS Workload to Task
dbms_advisor.add_sts_ref(taskname,sts_owner, sts_name);

-- Set the STS Workload Parameters
dbms_advisor.set_task_parameter(taskname,'VALID_ACTION_LIST',DBMS_ADVISOR.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname,'VALID_MODULE_LIST',DBMS_ADVISOR.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname,'SQL_LIMIT',DBMS_ADVISOR.ADVISOR_UNLIMITED);
dbms_advisor.set_task_parameter(taskname,'VALID_USERNAME_LIST',DBMS_ADVISOR.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname,'VALID_TABLE_LIST',DBMS_ADVISOR.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname,'INVALID_TABLE_LIST',
DBMS_ADVISOR.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname,'INVALID_ACTION_LIST',
DBMS_ADVISOR.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname,'INVALID_USERNAME_LIST',
DBMS_ADVISOR.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname,'INVALID_MODULE_LIST',
DBMS_ADVISOR.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname,'VALID_SQLSTRING_LIST',
DBMS_ADVISOR.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname,'INVALID_SQLSTRING_LIST','@!');

/* Set Task Parameters */
dbms_advisor.set_task_parameter(taskname,'ANALYSIS_SCOPE','ALL');
dbms_advisor.set_task_parameter(taskname,'RANKING_MEASURE','PRIORITY,OPTIMIZER_COST');
dbms_advisor.set_task_parameter(taskname,'DEF_PARTITION_TABLESPACE',
DBMS_ADVISOR.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname,'TIME_LIMIT',10000);
dbms_advisor.set_task_parameter(taskname,'MODE','COMPREHENSIVE');
dbms_advisor.set_task_parameter(taskname,'STORAGE_CHANGE',DBMS_ADVISOR.ADVISOR_UNLIMITED);
dbms_advisor.set_task_parameter(taskname,'DML_VOLATILITY','TRUE');
dbms_advisor.set_task_parameter(taskname,'WORKLOAD_SCOPE','PARTIAL');
dbms_advisor.set_task_parameter(taskname,'DEF_INDEX_TABLESPACE',
DBMS_ADVISOR.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname,'DEF_INDEX_OWNER',DBMS_ADVISOR.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname,'DEF_MVIEW_TABLESPACE',
DBMS_ADVISOR.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname,'DEF_MVIEW_OWNER',DBMS_ADVISOR.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname,'DEF_MVLOG_TABLESPACE',
DBMS_ADVISOR.ADVISOR_UNUSED);
dbms_advisor.set_task_parameter(taskname,'CREATION_COST','TRUE');
dbms_advisor.set_task_parameter(taskname,'JOURNALING','4');
dbms_advisor.set_task_parameter(taskname,'DAYS_TO_EXPIRE','30');

-- Execute the SQL Access Advisor Task
dbms_advisor.execute_task(taskname);
END;

```

```
$ expdp dumpfile=uatdblinks.dmp directory=clone_save full=y  
include=db_link userid =\"/ as sysdba\"
```

```
$ expdp dumpfile=uatdblinks.dmp directory=clone_save  
schemas=APPS,XX,XXL include=db_link userid ="/ as sysdba\"
```

```
full=y
include=db_link:"IN (SELECT db_link
                      from dba_db_links Where Owner = 'PUBLIC')"
```

```
full=y
include=DATABASE_EXPORT/SCHEMA/PUBLIC_SYNONYM/SYNONYM:"IN (SELECT
    synonym_name from dba_synonyms
    where Owner = 'PUBLIC' and table_owner = 'HR'))"
```

```
$ impdp dumpfile=pubdblinks.dmp directory=clone_save  
      full=y sqlfile=testfile.txt
```

```
$ cat testfile.txt
-- CONNECT SYS
ALTER SESSION SET EVENTS '10150 TRACE NAME CONTEXT FOREVER, LEVEL 1';
ALTER SESSION SET EVENTS '10904 TRACE NAME CONTEXT FOREVER, LEVEL 1';
ALTER SESSION SET EVENTS '25475 TRACE NAME CONTEXT FOREVER, LEVEL 1';
ALTER SESSION SET EVENTS '10407 TRACE NAME CONTEXT FOREVER, LEVEL 1';
ALTER SESSION SET EVENTS '10851 TRACE NAME CONTEXT FOREVER, LEVEL 1';
ALTER SESSION SET EVENTS '22830 TRACE NAME CONTEXT FOREVER, LEVEL 192 ';
-- new object type path: DATABASE_EXPORT/SCHEMA/DB_LINK
CREATE PUBLIC DATABASE LINK "CZPRD.BT.NET"
    USING 'czprd';
CREATE PUBLIC DATABASE LINK "ERP_ARCHIVE_ONLY_LINK.BT.NET"
    CONNECT TO "AM_STR_HISTORY_READ" IDENTIFIED BY VALUES
'05EDC7F2F211FF3A79CD5A526EF812A0FCC4527A185146A206C88098BB1FF8F0B1'
    USING 'ILMPRD1';
```

```
SQL> SELECT object_path, comments
  FROM database_export_objects
 WHERE object_path like '%CONSTRAINT%'
 AND object_path not like '%/%';
```

OBJECT\_PATH

COMMENTS

CONSTRAINT

Constraints (including referential constraints)

REF\_CONSTRAINT

Referential constraints

```
full=y  
exclude=schema:"in ('HR','SH')"
```

```
full=y
include=table:"IN (select table_name from dba_tables
      where table_name like '%TEMP'
      and owner not in ('SYSTEM', 'APPS'))"
```

```
schemas=GL  
query=GL_TRANSACTIONS: "where account_date > add_months(sysdate, -6)"  
query=GL.GL_LINES: " where transaction_date > add_months(sysdate, -6)"
```

```
schemas=GL
include=table:"like '%TRANSACTION%'"
query="where account_date > add_months(sysdate, -6)"
```

```
schemas=GL  
exclude=table:"like '%TRANSACTION'"
```

```
SQL> desc emp_dept
Name          Null?    Type
-----
DEPTNO        NOT NULL NUMBER(2)
DNAME          VARCHAR2(14)
ENAME          VARCHAR2(10)

SQL> select object_type from user_objects where object_name = 'EMP_DEPT';

OBJECT_TYPE
-----
VIEW
```

views\_as\_tables=scott.emp\_dept

```
$ expdp parfile=p1.txt
Export: Release 12.1.0.2.0 - Production on Wed Apr 8 22:06:49 2015
Copyright (c) 1982, 2014, Oracle and/or its affiliates. All rights reserved.
Connected to: Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options
Starting "SYSTEM"."SYS_EXPORT_TABLE_01": system/********@orcl parfile=p1.txt
Estimate in progress using BLOCKS method...
Processing object type TABLE_EXPORT/VIEWS_AS_TABLES/TABLE_DATA
Total estimation using BLOCKS method: 16 KB
Processing object type TABLE_EXPORT/VIEWS_AS_TABLES/TABLE
. . exported "SCOTT"."EMP_DEPT"           6.234 KB      14 rows
Master table "SYSTEM"."SYS_EXPORT_TABLE_01" successfully loaded/unloaded
*****
Dump file set for SYSTEM.SYS_EXPORT_TABLE_01 is:
/home/oracle/app/oracle/admin/cdb1/dpdump/x1.dmp
Job "SYSTEM"."SYS_EXPORT_TABLE_01" successfully completed at
Wed Apr 8 22:06:59 2015 elapsed 0 00:00:08
```

```
$ impdp parfile=p1.txt
Import: Release 12.1.0.2.0 - Production on Wed Apr 8 22:09:42 2015
Copyright (c) 1982, 2014, Oracle and/or its affiliates. All rights reserved.
Connected to: Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options
Master table "SYSTEM"."SYS_IMPORT_FULL_01" successfully loaded/unloaded
Starting "SYSTEM"."SYS_IMPORT_FULL_01": system/********@orcl parfile=p1.txt
Processing object type TABLE_EXPORT/VIEWS_AS_TABLES/TABLE
Processing object type TABLE_EXPORT/VIEWS_AS_TABLES/TABLE_DATA
. . imported "MIKE"."EMP_DEPT"          6.234 KB    14 rows
Job "SYSTEM"."SYS_IMPORT_FULL_01" successfully completed at
                           Wed Apr 8 22:09:47 2015 elapsed 0 00:00:03
```

```
SQL> desc emp_dept
Name          Null?    Type
-----
DEPTNO           NUMBER(2)
DNAME            VARCHAR2(14)
ENAME             VARCHAR2(10)
SQL> select object_type from user_objects where object_name = 'EMP_DEPT';

OBJECT_TYPE
-----
TABLE
```

TABLES=HR.ALL\_EMPLOYEES
REMAP\_DATA=HR.ALL\_EMPLOYEES.SSN:HR.HR\_UTILS.SSN\_SCRAMBLE

```
TABLES=SALES.DAILY_SALES
REMAP_TABLE=SALES.DAILY_SALES:DAILY_SALES_031515
REMAP_TABLESPACE:SALES_DATA:SALES_BKUP
```

SCHEMAS=SALES

REMAP\_SCHEMA=SALES:MIKE

TRANSFORM=SEGMENT\_ATTRIBUTES:N

SCHEMAS=SALES
REMAP\_SCHEMA=SALES:MIKE
TRANSFORM=SEGMENT\_ATTRIBUTES:N:TABLE
TRNASFORM=STORAGE:N:INDEX

SCHEMAS=DWH
LOGTIME=ALL
METRICS=YES
PARALLEL=8
DUMPFILE=dwh\_exp\_%U.dmp
DIRECTORY=SCRATCH\_AREA
STATUS=180
EXCLUDE=INDEX
TRANSFORM=SEGMENT\_ATTRIBUTES:N

SCHEMAS=DWH
LOGTIME=ALL
METRICS=YES
PARALLEL=8
DUMPFILE=dwh\_exp\_%U.dmp
DIRECTORY=SCRATCH\_AREA
INCLUDE=INDEX
TRANSFORM=SEGMENT\_ATTRIBUTES:N

```
DBMS_STATS.GATHER_SCHEMA_STATS('DWH') ;
```

```

CREATE OR REPLACE PROCEDURE XX_BACKUP_TABLES (
    schema_name VARCHAR2 DEFAULT USER)
IS
/*****
 NAME:      XX_BACKUP_TABLES
 PURPOSE:   Backup tables under schema
*****/
BEGIN
-- Export all the tables belonging to schema to file system

DECLARE
    dpumpfile    NUMBER;
    dpumpstatus   VARCHAR2 (200);
    tempvar       VARCHAR2 (200);
BEGIN
    EXECUTE IMMEDIATE
        'CREATE OR REPLACE DIRECTORY BACKUP_DIR AS ''/u01/app/oracle/expdp''';

    BEGIN
        -- verify if the Data Pump job table exist.
        -- drop if exist.
        SELECT table_name
            INTO tempvar
            FROM user_tables
           WHERE table_name = 'BACKUP_TABLE_EXP';

        EXECUTE IMMEDIATE 'DROP TABLE BACKUP_TABLE_EXP';
    EXCEPTION
        WHEN NO_DATA_FOUND
        THEN
            NULL;
    END;

    -- define job
    dpumpfile :=
        DBMS_DATAPUMP.open (OPERATION    => 'EXPORT',
                            JOB_MODE     => 'SCHEMA',
                            JOB_NAME     => 'BACKUP_TABLE_EXP');

```

```

-- add dump file name
DBMS_DATAPUMP.add_file (
    HANDLE      => dpumpfile,
    FILENAME   => 'BACKUP_tabs.dmp',
    DIRECTORY  => 'BACKUP_DIR',
    FILETYPE   => DBMS_DATAPUMP.KU$_FILE_TYPE_DUMP_FILE,
    REUSEFILE  => 1);

-- add log file name
DBMS_DATAPUMP.add_file (
    HANDLE      => dpumpfile,
    FILENAME   => 'BACKUP_tabs.log',
    DIRECTORY  => 'BACKUP_DIR',
    FILETYPE   => DBMS_DATAPUMP.KU$_FILE_TYPE_LOG_FILE);

-- add filters to export only one schema
DBMS_DATAPUMP.metadata_filter (HANDLE  => dpumpfile,
                                NAME     => 'SCHEMA_LIST',
                                VALUE    => schema_name);

-- start the export job
DBMS_DATAPUMP.start_job (HANDLE => dpumpfile);

-- wait until the job completes
DBMS_DATAPUMP.wait_for_job (HANDLE => dpumpfile, JOB_STATE => dpumpstatus);
-- close the job
DBMS_DATAPUMP.detach (HANDLE => dpumpfile);

EXCEPTION
  WHEN OTHERS
  THEN
    DBMS_DATAPUMP.detach (HANDLE => dpumpfile);
    RAISE;
END;

END XX_BACKUP_TABLES;
/

```

TRANSFORM=DISABLE\_ARCHIVE\_LOGGING:Y

```
SET LINESIZE 80
COL endian_format  FORMAT A12      HEADING "Endian|Format"
COL platform_name  FORMAT A60      HEADING "Platform Name" WRAP
TTITLE "Platforms Currently Supported for TTS / TDB Operations|(from
V$TRANSPORTABLE_PLATFORM)"
SELECT
    endian_format
    ,platform_name
  FROM v$transportable_platform
 ORDER BY endian_format, platform_name;
TTITLE OFF
```

```
Platforms Currently Supported for TTS / TDB Operations
  (from V$TRANSPORTABLE_PLATFORM)
```

| Endian Format | Platform Name |
|---------------|-----------------------------------|
| Big | AIX-Based Systems (64-bit) |
| Big | Apple Mac OS |
| Big | HP-UX (64-bit) |
| Big | HP-UX IA (64-bit) |
| Big | IBM Power Based Linux |
| Big | IBM zSeries Based Linux |
| Big | Solaris[tm] OE (32-bit) |
| Big | Solaris[tm] OE (64-bit) |
| Little | Apple Mac OS (x86-64) |
| Little | HP IA Open VMS |
| Little | HP Open VMS |
| Little | HP Tru64 UNIX |
| Little | Linux IA (32-bit) |
| Little | Linux IA (64-bit) |
| Little | Linux x86 64-bit |
| Little | Microsoft Windows IA (32-bit) |
| Little | Microsoft Windows IA (64-bit) |
| Little | Microsoft Windows x86 64-bit |
| Little | Solaris Operating System (x86) |
| Little | Solaris Operating System (x86-64) |

```
20 rows selected.
```

```
SET LINESIZE 60
TTITLE "Current Database Platform Endianess| (from V$DATABASE +
V$TRANSPORTABLE_PLATFORM)"
COL name          FORMAT A16      HEADING "Database Name"
COL endian_format FORMAT A12      HEADING "Endian|Format"
SELECT
  D.name
 ,TP.endian_format
FROM
  v$transportable_platform TP
 ,v$database D
WHERE TP.platform_name = D.platform_name;
TTITLE OFF
```

```
        Current Database Platform Endianess
        (from V$DATABASE + V$TRANSPORTABLE_PLATFORM)
```

| Database Name | Endian Format |
|---------------|---------------|
| DB11203 | Little |

```
-----
-- Using DBMS_TDB to validate if a database is ready for transport
-----
SET SERVEROUTPUT ON
DECLARE
    db_check BOOLEAN;
BEGIN

-----
-- Can this database be transported to
-- the specified destination platform?
-----
db_check := DBMS_TDB.CHECK_DB(
    target_platform_name => 'Linux IA (32-bit)'
    ,skip_option => DBMS_TDB.SKIP_OFFLINE
);
IF db_check
    THEN DBMS_OUTPUT.PUT_LINE('Database can be transferred to destination platform.');
    ELSE DBMS_OUTPUT.PUT_LINE('Warning!!! Database CANNOT be transported to
destination platform.');
END IF;

-----
-- Are there any directories or external objects that need to be
-- transferred separately after these tablespace(s) have been
-- transported to the destination platform?
-----
db_check := DBMS_TDB.CHECK_EXTERNAL;
IF db_check
    THEN DBMS_OUTPUT.PUT_LINE('Database can be transferred to destination platform.');
    ELSE DBMS_OUTPUT.PUT_LINE('Warning!!! Database CANNOT be transported to
destination platform.');
END IF;

END;
/
```

```
BEGIN
    DBMS_TTS.TRANSPORT_SET_CHECK(
        ts_list => 'TPCH_DATA,TPCH_IDX'
        ,incl_constraints => TRUE
        ,full_check => TRUE
    );
END;
/
SELECT * FROM transport_setViolations;
no rows selected
```

```
-----  
-- Prepare the source database  
-----  
  
$> mkdir /home/oracle/TTS  
  
DROP DATABASE LINK db10205;  
CREATE DATABASE LINK db10205  
    CONNECT TO system IDENTIFIED BY "oracle_4U"  
    USING 'db10205';  
  
CREATE OR REPLACE DIRECTORY db10205_dbf  
    AS '/u02/app/oracle/oradata/db10205';  
CREATE OR REPLACE DIRECTORY ttsfiles  
    AS '/home/oracle/TTS';  
GRANT READ, WRITE ON DIRECTORY ttsfiles TO PUBLIC;  
  
-----  
-- Prepare the destination database  
-----  
  
$> mkdir /home/oracle/TTS  
  
CREATE OR REPLACE DIRECTORY db12010_data  
    AS '+DATA/ORCL/DATAFILE';  
CREATE OR REPLACE DIRECTORY ttsfiles  
    AS '/home/oracle/TTS';  
GRANT READ, WRITE ON DIRECTORY ttsfiles TO PUBLIC;
```

```
ALTER TABLESPACE tpch_data READ ONLY;
```

```
Tablespace altered.
```

```
ALTER TABLESPACE tpch_idx READ ONLY;
```

```
Tablespace altered.
```

```
JOB_NAME = TTS_1
DIRECTORY = TTSFILES
DUMPFILE = tts_1.dmp
LOGFILE = tts_1.log
TRANSPORT_TABLESPACES = "tpch_data, tpch_idx"
TRANSPORT_FULL_CHECK = TRUE
```

```
$> expdp system/oracle_4U PARFILE=/home/oracle/TTS/TTS_1.dpectl

Export: Release 10.2.0.1.0 - 64bit Production on Friday, 16 January, 2015 18:01:30

Copyright (c) 2003, 2005, Oracle. All rights reserved.

Connected to: Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - 64bit Production
With the Partitioning, OLAP and Data Mining options
Starting "SYS"."SYS_EXPORT_TRANSPORTABLE_01":
userid="/*****@(DESCRIPTION=(ADDRESS=(PROTOCOL=beq)(PROGRAM=/u01/app/oracle/product
/10.2.0/db_1/bin/oracle)(ARGV0=oraclefxk1)(ARGS=\(DESCRIPTION=\(LOCAL=YES\)\)(ADDRESS=
\PROTOCOL=beq\)\)\))(ENVS=ORACLE_SID=fxk1)(CONNECT_DATA=(SID=fxk1)))
AS SYSDBA" transport tablespaces=
TPCH_DATA, TPCH_IDX dumpfile=tts_2.dmp directory=TTSFILES logfile=tts_2.log
Processing object type TRANSPORTABLE_EXPORT/TABLE
Processing object type TRANSPORTABLE_EXPORT/GRANT/OWNER_GRANT/OBJECT_GRANT
Processing object type TRANSPORTABLE_EXPORT/INDEX
Processing object type TRANSPORTABLE_EXPORT/CONSTRAINT/CONSTRAINT
Processing object type TRANSPORTABLE_EXPORT/INDEX_STATISTICS
Processing object type TRANSPORTABLE_EXPORT/CONSTRAINT/REF_CONSTRAINT
Processing object type TRANSPORTABLE_EXPORT/TABLE_STATISTICS
Processing object type TRANSPORTABLE_EXPORT/POST_INSTANCE/PLUGTS_BLK
Master table "SYS"."SYS_EXPORT_TRANSPORTABLE_01" successfully loaded/unloaded
```

```
BEGIN
    DBMS_FILE_TRANSFER.PUT_FILE(
        source_directory_object => 'DB10205_DBF'
        ,source_file_name => 'tpch_data.dbf'
        ,destination_directory_object => 'DB12010_DATA'
        ,destination_file_name => 'TPCH_DATA.DBF'
        ,destination_database => 'DB12010'
    );
    DBMS_FILE_TRANSFER.PUT_FILE(
        source_directory_object => 'DB10205_DBF'
        ,source_file_name => 'tpch_idx.dbf'
        ,destination_directory_object => 'DB12010_DATA'
        ,destination_file_name => 'TPCH_IDX.DBF'
        ,destination_database => 'DB12010'
    );
    DBMS_FILE_TRANSFER.PUT_FILE(
        source_directory_object => 'TTSFILES'
        ,source_file_name => 'tts_1.dmp'
        ,destination_directory_object => 'TTSFILES'
        ,destination_file_name => 'tts_1.dmp'
        ,destination_database => 'DB12010'
    );
END;
/
```

```
JOB_NAME = TTS_1
DIRECTORY = TTSFILES
DUMPFILE = tts_1.dmp
LOGFILE = tts_1.log
TRANSPORT_DATAFILES = '+DATA/ORCL/DATAFILE/TPCH_DATA.DBF', '+DATA/ORCL/DATAFILE/TPCH_IDX.DBF'
```

```
$> impdp system/oracle_4U PARFILE=tts_1.dpictl
>> Results of successful transportable tablespace import at destination
>> (from destination database's alert log)

.
.
Fri Jan 16 18:32:18 2015
DM00 started with pid=36, OS id=2269, job SYSTEM.TTS_1
Fri Jan 16 18:32:18 2014
DW00 started with pid=37, OS id=2271, wid=1, job SYSTEM.TTS_1
Plug in tablespace TPCH_IDX with datafile
 '+DATA/ORCL/DATAFILE/TPCH_IDX.DBF'
Plug in tablespace TPCH_DATA with datafile
 '+DATA/ORCL/DATAFILE/TPCH_DATA.DBF'
.
```

```
SQL> ALTER TABLESPACE tpch_data READ WRITE;
Tablespace altered.
SQL> ALTER TABLESPACE tpch_idx READ WRITE;
Tablespace altered.
```

```
-----  
-- On source database:  
-----  
SQL> CREATE OR REPLACE DIRECTORY ora10g_dbf  
      AS '+DATA/ORA10G/DATAFILE';  
SQL> GRANT READ, WRITE ON DIRECTORY ora10g_dbf TO PUBLIC;  
  
SQL> CREATE OR REPLACE DIRECTORY xttsfiles  
      AS '/home/oracle/XTTSFILES';  
SQL> GRANT READ, WRITE ON DIRECTORY xttsfiles TO PUBLIC;  
  
-----  
-- On destination database:  
-----  
SQL> CREATE OR REPLACE DIRECTORY ora12c_dbf  
      AS '+DATA/ORA12010/DATAFILE';  
SQL> GRANT READ, WRITE ON DIRECTORY ora12c_dbf TO PUBLIC;  
  
SQL> CREATE OR REPLACE DIRECTORY xttsfiles  
      AS '/home/oracle/XTTSFILES';  
SQL> GRANT READ, WRITE ON DIRECTORY xttsfiles TO PUBLIC;  
SQL> DROP DATABASE LINK ora10g;  
SQL> CREATE DATABASE LINK ora10g  
      CONNECT TO system IDENTIFIED BY oracle_4U  
      USING 'ORA10G';
```

```
$> unzip -d /home/oracle/XTTS rman_xttconvert_1.4.zip

# Editing xtt.properties configuration file on source and destination:

-- Do this on source and destination platforms
$> unzip -d /home/oracle/XTTS rman_xttconvert_1.4.zip
-- Edit xtt.properties file on source and destination platforms

# xtt.properties
# Parameters for Cross-Platform Transportable Tablespace (XTTS) demonstrations
# Tablespace(s) for XTTS transport:
 tablespaces=AP_DATA,AP_IDX
 # Source database parameters:
 platformid=2
 srkdir=ORA10G_DBF
 srclink=ORA10G
 dfcopydir=/home/oracle/XTTSFILES
 backupformat=/home/oracle/XTTSFILES
 # Destination database parameters:
 dstdir=ORA12c_DBF
 stageondest=/home/oracle/XTTSFILES
 storageondest=+DATA
 backupondest=+FRA
 asm_home=/u01/app/oracle/product/12.1.0/grid
 asm_sid=+ASM
 parallel=4
 rollparallel=2
```

```
EXEC DBMS_TTS.TRANSPORT_SET_CHECK('TPCH_DATA,TPCH_IDX',TRUE,TRUE);
PL/SQL procedure successful.
```

```
SELECT * FROM transport_setViolations;
No rows returned
```

```
SQL> ALTER TABLESPACE tpch_data READ ONLY;
Tablespace altered.
```

```
SQL> ALTER TABLESPACE tpch_idx READ ONLY;
Tablespace altered.
```

```
$> expdp system/oracle_4U parfile=/home/oracle/fte_db11203.dpectl
```

Export: Release 11.2.0.4.0 - Production on Fri Jan 16 20:23:34 2015

Copyright (c) 1982, 2011, Oracle and/or its affiliates. All rights reserved.

Connected to: Oracle Database 11g Enterprise Edition Release 11.2.0.4.0 - 64bit Production
With the Partitioning, Automatic Storage Management, OLAP, Data Mining
and Real Application Testing options

Starting "SYSTEM"."SYS\_EXPORT\_FULL\_01": system/\*\*\*\*\*\*\*\*\* parfile=/home/oracle/fte\_oral1203.dpectl
Estimate in progress using BLOCKS method...

Processing object type DATABASE\_EXPORT/EARLY\_OPTIONS/VIEWS\_AS\_TABLES/TABLE\_DATA

Processing object type DATABASE\_EXPORT/NORMAL\_OPTIONS/TABLE\_DATA

Processing object type DATABASE\_EXPORT/NORMAL\_OPTIONS/VIEWS\_AS\_TABLES/TABLE\_DATA

Processing object type DATABASE\_EXPORT/SCHEMA/TABLE/TABLE\_DATA

Total estimation using BLOCKS method: 64.93 MB

Processing object type DATABASE\_EXPORT/PRE\_SYSTEM\_IMPCALLOUT/MARKER

...

<< many lines omitted for sake of brevity >>

...

. . . exported "SYSTEM"."REPCAT\$\_TEMPLATE\_SITES" 0 KB 0 rows

. . . exported "SYSTEM"."REPCAT\$\_TEMPLATE\_TARGETS" 0 KB 0 rows

. . . exported "SYSTEM"."REPCAT\$\_USER\_AUTHORIZATIONS" 0 KB 0 rows

. . . exported "SYSTEM"."REPCAT\$\_USER\_PARM\_VALUES" 0 KB 0 rows

. . . exported "SYSTEM"."SQLPLUS\_PRODUCT\_PROFILE" 0 KB 0 rows

Master table "SYSTEM"."SYS\_EXPORT\_FULL\_01" successfully loaded/unloaded

\*\*\*\*\*

Dump file set for SYSTEM.SYS\_EXPORT\_FULL\_01 is:

/u01/app/oracle/admin/oral1g/dpdump/fte\_db11203.dmp

\*\*\*\*\*

Datafiles required for transportable tablespace TPCH\_DATA:

+DATA/oral1g/datafile/tpch\_data.258.858108115

Datafiles required for transportable tablespace TPCH\_IDX:

+DATA/oral1g/datafile/tpch\_idx.257.858108117

Job "SYSTEM"."SYS\_EXPORT\_FULL\_01" successfully completed ...

```
CREATE OR REPLACE DIRECTORY dpdir
AS '/u01/app/oracle/admin/cdb122/dpdump';
GRANT READ ON DIRECTORY dpdir TO PUBLIC;
GRANT WRITE ON DIRECTORY dpdir TO PUBLIC;
```

```
$> impdp system/oracle_4U@db12010 parfile=/home/oracle/fti_db12010.dpictl
```

```
DIRECTORY=DPDIR
DUMPFILE=fte_db11203.dmp
LOGFILE=fti_db12010.log
FULL=Y
TRANSPORT_DATAFILES='/u01/app/oracle/oradata/db12010/tpch_data.dbf',
'/u01/app/oracle/oradata/db12010/tpch_idx.dbf'
```

```
Import: Release 12.1.0.1.0 - Production on Fri Jan 16 21:14:03 2015
```

```
Copyright (c) 1982, 2013, Oracle and/or its affiliates. All rights reserved.
```

```
Connected to: Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options
Master table "SYSTEM"."SYS_IMPORT_FULL_01" successfully loaded/unloaded
Source timezone version is +00:00 and target timezone version is -07:00.
Starting "SYSTEM"."SYS_IMPORT_FULL_01": system/********@db12010 parfile=/home/oracle/fti_db12010.
dpctl
Processing object type DATABASE_EXPORT/PRE_SYSTEM_IMPCALLOUT/MARKER
Processing object type DATABASE_EXPORT/PRE_INSTANCE_IMPCALLOUT/MARKER
Processing object type DATABASE_EXPORT/TABLESPACE
.
.
.
<< many lines omitted for sake of brevity >>
.
.
```

```
SQL> CREATE BIGFILE TEMPORARY TABLESPACE biglmtt1
  TEMPFILE '+DATA'
  SIZE 256M
  AUTOEXTEND ON
  TABLESPACE GROUP ttg_1;
```

```
Tablespace created.
```

```
SQL> CREATE BIGFILE TEMPORARY TABLESPACE biglmtt2
  TEMPFILE '+DATA2'
  SIZE 256M
  AUTOEXTEND ON
  EXTENT MANAGEMENT LOCAL
  UNIFORM SIZE 64M;
```

Tablespace created.

```
ALTER TABLESPACE biglmtt2
  TABLESPACE GROUP ttg_1;
```

Tablespace altered.

```
SQL> SELECT * FROM dba_tablespace_groups;
```

| GROUP_NAME | TABLESPACE_NAME |
|------------|-----------------|
| TTG_1 | BIGLMTT1 |
| TTG_1 | BIGLMTT2 |

```
ALTER USER ap TEMPORARY TABLESPACE ttg_1;
```

User altered.

```
SQL> CREATE TEMPORARY TABLESPACE gtt_lmtt
  TEMPFILE '+DATA2'
  SIZE 64M
  AUTOEXTEND ON
  EXTENT MANAGEMENT LOCAL
  UNIFORM SIZE 1M;
```

```
Tablespace created.
```

```
SQL> CREATE GLOBAL TEMPORARY TABLE ap.gtt_load_revenue(
  acct_nbr      NUMBER(5)
,acct_desc     VARCHAR2(32)
,acct_balance  NUMBER(8,2)
)
ON COMMIT PRESERVE ROWS
TABLESPACE gtt_lmtt;
```

Table created.

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

INSERT INTO gl.gtt_load_revenue VALUES (12300, 'GL Account 12300', 123.00);
INSERT INTO gl.gtt_load_revenue VALUES (12301, 'GL Account 12301', 123.01);
INSERT INTO gl.gtt_load_revenue VALUES (12302, 'GL Account 12302', 123.02);
INSERT INTO gl.gtt_load_revenue VALUES (12303, 'GL Account 12303', 123.03);
INSERT INTO gl.gtt_load_revenue VALUES (12304, 'GL Account 12304', 123.04);
INSERT INTO gl.gtt_load_revenue VALUES (12305, 'GL Account 12305', 123.05);
INSERT INTO gl.gtt_load_revenue VALUES (12306, 'GL Account 12306', 123.06);
INSERT INTO gl.gtt_load_revenue VALUES (12307, 'GL Account 12307', 123.07);
INSERT INTO gl.gtt_load_revenue VALUES (12308, 'GL Account 12308', 123.08);
INSERT INTO gl.gtt_load_revenue VALUES (12309, 'GL Account 12309', 123.09);
. . .
<< many other transactions! >>
. . .
```

```

SET LINESIZE 90
SET PAGESIZE 20000
COL begin_time      FORMAT A20          HEADING "Begin Time"
COL end_time        FORMAT A20          HEADING "End Time"
COL undotsn         FORMAT 9999999999  HEADING "Undo|TSP #"
COL undoblkcnt      FORMAT 9999999     HEADING "Undo|Blocks"
COL txncount        FORMAT 9999999     HEADING "Trxn|Count"
COL maxquerylen     FORMAT 9999999     HEADING "Max|Query|Length"
TTITLE "Temporary UNDO Segment Statistics|(from V$TEMPUNDOSTAT)"
SELECT
    TO_CHAR(begin_time, 'mm-dd-yyyy hh24:mi:ss') begin_time
    ,TO_CHAR(end_time, 'mm-dd-yyyy hh24:mi:ss') end_time
    ,undotsn
    ,undoblkcnt
    ,txncount
    ,maxquerylen
FROM v$tempundostat
;
TTITLE OFF

```

Thu Jan 29

page 1

Temporary UNDO Segment Statistics
(from V\$TEMPUNDOSTAT)

| Begin Time | End Time | TSP # | Undo | Undo | Trxn | Max |
|---------------------|---------------------|------------|-------|--------|-------|--------------|
| | | | Count | Blocks | Count | Query Length |
| 01-29-2015 19:57:00 | 01-29-2015 20:06:07 | 3 | 0 | 0 | 0 | 0 |
| 01-29-2015 19:47:00 | 01-29-2015 19:57:00 | 3 | 1 | 1 | 1 | 0 |
| 01-29-2015 19:37:00 | 01-29-2015 19:47:00 | 3 | 1 | 1 | 1 | 0 |
| 01-27-2015 20:57:00 | 01-29-2015 19:37:00 | 2147483647 | 0 | 0 | 0 | 0 |

```
-----  
-- Gather statistics specific to each GTT:  
-----  
BEGIN  
    DBMS_STATS.SET_GLOBAL_PREFS (  
        pname => 'GLOBAL_TEMP_TABLE_STATS'  
        ,pvalue => 'SESSION'  
    );  
END;  
/  
PL/SQL procedure completed successfully.  
  
-----  
-- Gather statistics and share them for all GTT iterations:  
-----  
BEGIN  
    DBMS_STATS.SET_GLOBAL_PREFS (  
        pname => 'GLOBAL_TEMP_TABLE_STATS'  
        ,pvalue => 'SHARED'  
    );  
END;  
/  
PL/SQL procedure completed successfully.
```

```
SQL> ALTER SYSTEM SET pga_aggregate_target = 10M;
System altered.
```

```
SELECT /*+ MONITOR ReallyBadSorting */ DISTINCT
  cust_id
 ,prod_id
 ,SUM(qty)
 ,SUM(amt)
FROM (SELECT
  cust_id
 ,prod_id
 ,SUM(quantity_sold) qty
 ,SUM(amount_sold) amt
 FROM sh.sales
 WHERE cust_id BETWEEN 1 and 15000
 GROUP BY cust_id, prod_id
 UNION
 SELECT
  cust_id
 ,prod_id
 ,SUM(quantity_sold) qty
 ,SUM(amount_sold) amt
 FROM sh.sales
 WHERE cust_id BETWEEN 90000 and 105000
 GROUP BY cust_id, prod_id
 UNION
 SELECT
  cust_id
 ,prod_id
 ,SUM(quantity_sold) qty
 ,SUM(amount_sold) amt
 FROM sh.sales
 WHERE cust_id BETWEEN 20000 and 35000
 GROUP BY cust_id, prod_id
 UNION
 SELECT
```

```
        cust_id
        ,prod_id
        ,SUM(quantity_sold) qty
        ,SUM(amount_sold) amt
    FROM sh.sales
    WHERE cust_id BETWEEN 80000 and 95000
    GROUP BY cust_id, prod_id
UNION
SELECT
        cust_id
        ,prod_id
        ,SUM(quantity_sold) qty
        ,SUM(amount_sold) amt
    FROM sh.sales
    WHERE cust_id BETWEEN 30000 and 45000
    GROUP BY cust_id, prod_id
UNION
SELECT
        cust_id
        ,prod_id
        ,SUM(quantity_sold) qty
        ,SUM(amount_sold) amt
    FROM sh.sales
    WHERE cust_id BETWEEN 70000 and 85000
    GROUP BY cust_id, prod_id
UNION
SELECT
        cust_id
        ,prod_id
        ,SUM(quantity_sold) qty
        ,SUM(amount_sold) amt
    FROM sh.sales
    WHERE cust_id BETWEEN 40000 and 55000
    GROUP BY cust_id, prod_id
UNION
SELECT
```

```

        cust_id
        ,prod_id
        ,SUM(quantity_sold) qty
        ,SUM(amount_sold) amt
    FROM sh.sales
    WHERE cust_id > 100000
    GROUP BY cust_id, prod_id
UNION
SELECT
        cust_id
        ,prod_id
        ,SUM(quantity_sold) qty
        ,SUM(amount_sold) amt
    FROM sh.sales
    WHERE cust_id BETWEEN 50000 and 65000
    GROUP BY cust_id, prod_id
UNION
SELECT
        cust_id
        ,prod_id
        ,SUM(quantity_sold) qty
        ,SUM(amount_sold) amt
    FROM sh.sales
    WHERE cust_id BETWEEN 10000 and 25000
    GROUP BY cust_id, prod_id
UNION
SELECT
        cust_id
        ,prod_id
        ,SUM(quantity_sold) qty
        ,SUM(amount_sold) amt
    FROM sh.sales
    WHERE cust_id BETWEEN 60000 and 75000
    GROUP BY cust_id, prod_id
)
GROUP BY ROLLUP(cust_id, prod_id)
ORDER BY
    1 DESC
    ,2 ASC
    ,3 DESC
    ,4 ASC;

```

<< output from AUTOTRACE: >>

Statistics

| | |
|---------|--|
| 41 | recursive calls |
| 35 | db block gets |
| 352 | consistent gets |
| 4858 | physical reads |
| 0 | redo size |
| 8014678 | bytes sent via SQL*Net to client |
| 211026 | bytes received via SQL*Net from client |
| 19136 | SQL*Net roundtrips to/from client |
| 0 | sorts (memory) |
| 3 | sorts (disk) |
| 287014 | rows processed |

```

SET LINESIZE 130
SET PAGESIZE 20000
COL tablespace FORMAT A12      HEADING "Tablespace"
COL file#   FORMAT 9999       HEADING "TEMP|File|#"
COL phyrds  FORMAT 9999999    HEADING "Phys|Reads"
COL phywrts FORMAT 9999999    HEADING "Phys|Writes"
COL phyblkrd FORMAT 9999999   HEADING "Phys|Blocks|Read"
COL phyblkwrt FORMAT 9999999  HEADING "Phys|Blocks|Written"
COL sbrs    FORMAT 9999999    HEADING "Single|Block|Reads"
COL readtim FORMAT 9999999    HEADING "Read|Time"
COL writetim FORMAT 9999999   HEADING "Write|Time"
COL sbtmp   FORMAT 99999      HEADING "Single|Block|Read|Time"
COL avgiotim FORMAT 99999     HEADING "Avg|I/O|Time"
COL lstatotim FORMAT 99999    HEADING "Last|I/O|Time"
COL miniotim FORMAT 99999    HEADING "Min|I/O|Time"
COL maxiortm FORMAT 99999    HEADING "Max|I/O|Read|Time"
COL maxiowtm FORMAT 99999    HEADING "Max|I/O|Write|Time"
TTITLE 'TEMPFILE I/O Statistics'
SELECT
  TSP.name tablespace
 ,TS.file#
 ,phyrds
 ,phywrts
 ,phyblkrd
 ,phyblkwrt
 ,singleblkrd sbrs
 ,readtim
 ,writetim
 ,singleblkrdtim sbtmp
 ,avgiotim
 ,lstatotim
 ,miniotim
 ,maxiortm
 ,maxiowtm
FROM v$tempstat TS, v$tempfile TF, v$tablespace TSP
WHERE TS.file# = TF.file#
 AND TF.ts# = TSP.ts#
ORDER BY TS.file#
;
TTITLE OFF

```

TEMPFILE I/O Statistics

| Tablespace | TEMP
File
| Single | | | | | | | | | | Max
I/O
Time | Max
I/O
Time | |
|------------|-------------------|---------------|----------------|------------------------|---------------------------|--------------------------|-------------------------|--------------|---------------|--------------|------------|--------------------|--------------------|--------------|
| | | Phys
Reads | Phys
Writes | Phys
Blocks
Read | Phys
Blocks
Written | Single
Block
Reads | Single
Block
Time | Read
Time | Write
Time | Read
Time | Avg
I/O | Last
I/O | Min
I/O | Read
Time |
| TEMP | 1 | 839 | 852 | 6340 | 6692 | 57 | 628 | 3171 | 38 | 3 | 1 | 0 | 20 | 289 |
| GTT_LMTT | 2 | 2 | 6 | 2 | 6 | 2 | 3 | 4 | 3 | 2 | 0 | 0 | 2 | 2 |
| BIGLMTT1 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 7 | 0 | 0 | 0 |
| BIGLMTT2 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 7 | 0 | 0 | 0 |

```
SET LINESIZE 60
SET PAGESIZE 20000
COL name    FORMAT A45          HEADING "Statistic"
COL value   FORMAT 99999999     HEADING "Value"
TTITLE "Temporary Tablespace - Current Session Instance Statistics"
SELECT SN.name, SS.value
  FROM .
      v$mystat SS
 ,v$statname SN
WHERE SS.statistic# = SN.statistic#
  AND (SN.name LIKE '%physical reads direct temporary%'
    OR SN.name LIKE '%physical writes direct temporary%')
;
TTITLE OFF
```

Temporary Tablespace - Current Session Instance Statistics

| Statistic | Value |
|---|-------|
| physical reads direct temporary tablespace | 5060 |
| physical writes direct temporary tablespace | 5648 |

```

SET LINESIZE 80
SET PAGESIZE 20000
COL event           FORMAT A40  HEADING "System Wait Event"
COL total_waits     HEADING "Total|Waits"
COL wait_secs       HEADING "Total|Wait|Time|(s)"
COL avg_wait_secs   HEADING "Avg|Wait|Time|(s)"
TTITLE "Temporary Tablespace - Current Session Wait Events"
SELECT
  DISTINCT event
 ,total_waits
 ,(time_waited / 100) wait_secs
 ,(average_wait / 100) avg_wait_secs
FROM
  v$session_event E
 ,v$mystat S
WHERE event LIKE '%temp%'
 AND E.sid = S.sid
;
TTITLE OFF

```

| System Wait Event | Total | Avg |
|------------------------|-------|------------|
| | Wait | Wait |
| | Total | Time |
| | Waits | (s) |
| direct path write temp | 1 | 0 .0031 |
| direct path read temp | 278 | 1.79 .0064 |

```

SQL> WITH system_event AS
  2      (SELECT CASE WHEN (event LIKE '%latch%' or event
  3                          LIKE '%mutex%' or event like 'cursor:%')
  4                      THEN event ELSE wait_class
  5                  END wait_type, e.*
  6              FROM v$system_event e)
  7  SELECT wait_type, SUM(total_waits) total_waits,
  8         round(SUM(time_waited_micro)/1000000,2) time_waited_seconds,
  9         ROUND(  SUM(time_waited_micro)
10                 * 100
11                 / SUM(SUM(time_waited_micro)) OVER (), 2) pct
12  FROM (SELECT wait_type, event, total_waits, time_waited_micro
13          FROM system_event e
14        UNION
15        SELECT 'CPU', stat_name, NULL, VALUE
16          FROM v$sys_time_model
17         WHERE stat_name IN ('background cpu time', 'DB CPU')) l
18 WHERE wait_type <> 'Idle'
19 GROUP BY wait_type
20 ORDER BY 4 DESC
21 /

```

| 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 |

```

SQL> WITH latch AS (
  2   SELECT name,
  3     ROUND(gets * 100 / SUM(gets) OVER (), 2) pct_of_gets,
  4     ROUND(misses * 100 / SUM(misses) OVER (), 2) pct_of_misses,
  5     ROUND(sleeps * 100 / SUM(sleeps) OVER (), 2) pct_of_sleeps,
  6     ROUND(wait_time * 100 / SUM(wait_time) OVER (), 2)
  7       pct_of_wait_time
  8   FROM v$latch)
  9 SELECT *
10 FROM latch
11 WHERE pct_of_wait_time > .1 OR pct_of_sleeps > .1
12 ORDER BY pct_of_wait_time DESC;

```

| NAME | Pct of Gets | Pct of Misses | Pct of Sleeps | Pct of Wait Time |
|--------------------------------|----------------------|---------------|---------------|------------------|
| | cache buffers chains | 99.59 | 99.91 | 70.59 |
| 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 |

```

SQL> 1
 1 WITH ash_query AS (
 2     SELECT event, program,
 3             h.module, h.action,    object_name,
 4             SUM(time_waited)/1000 reltime, COUNT( * ) waits,
 5             username, sql_text,
 6             RANK() OVER (ORDER BY COUNT(*) DESC) AS wait_rank
 7     FROM  v$active_session_history h
 8     JOIN dba_users u  USING (user_id)
 9     LEFT OUTER JOIN dba_objects o
10         ON (o.object_id = h.current_obj#)
11     LEFT OUTER JOIN v$sql s USING (sql_id)
12     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 |
|---|----------|----------|-------------|-------|
| <hr/> SQL_TEXT <hr/> | | | | |
| library cache: mutex X | SQL*Plus | OPSG | | 13 |
| latch: shared pool | SQL*Plus | OPSG | | 8 |
| latch: shared pool
begin | 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
begin | 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 WHERE id BETWEEN 124410 AND 360759 | | | | |
| latch: shared pool
SELECT quantity_sold , amount_sold FROM lt_sales t539571 WHERE id BETWEEN 512313 AND 825315 | SQL*Plus | OPSG | | 1 |
| library cache: mutex X
SELECT quantity_sold , amount_sold FROM lt_sales t539563 WHERE id BETWEEN 698302 AND 392634 | SQL*Plus | OPSG | | 1 |
| latch: shared pool
SELECT quantity_sold , amount_sold FROM lt_sales t539555 WHERE id BETWEEN 387009 AND 268338 | SQL*Plus | OPSG | LT_SALES_PK | 1 |

```

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 /
  8             SUM(concurrency_wait_time) OVER (), 2) pct_of_con_time,
  9       RANK() OVER (ORDER BY concurrency_wait_time DESC) ranking
 10      FROM v$sql
 11     WHERE elapsed_time > 0)
 12    SELECT sql_text, con_time_ms, con_time_pct,
 13          pct_of_con_time
 14   FROM sql_conc_waits
 15  WHERE ranking <= 10
 16 ORDER BY ranking ;

```

| SQL Text | SQL Conc | % Tot |
|--|---------------|----------------|
| | Conc Time(ms) | Time% ConcTime |
| DECLARE job BINARY_INTEGER := :job; next
_date DATE := :mydate; broken BOOLEAN : | 899 | 18.41 44.21 |
| select max(data) from log_data where id<
:id | 472 | .01 23.18 |
| begin query_loops (run_seconds=>120 ,
hi_val =>1000 , use_ | 464 | .01 22.80 |
| update sys.aud\$ set action#=:2, returnco
de=:3, logoff\$time=cast(SYS_EXTRACT_UTC
{ | 143 | 75.46 7.02 |

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

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

```
SQL> WITH force_matches AS
  2      (SELECT force_matching_signature,
  3       COUNT( * ) matches,
  4       MAX(sql_id || child_number) max_sql_child,
  5       DENSE_RANK() OVER (ORDER BY COUNT( * ) DESC)
  6          ranking
  7     FROM v$sql
  8    WHERE force_matching_signature <> 0
  9        AND parsing_schema_name <> 'SYS'
10    GROUP BY force_matching_signature
11    HAVING COUNT( * ) > 5)
12   SELECT sql_id,      matches, parsing_schema_name schema, sql_text
13   FROM      v$sql JOIN force_matches
14     ON (sql_id || child_number = max_sql_child)
15 WHERE ranking <= 10
16 ORDER BY matches DESC;
```

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

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

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

```
SQL> SELECT COUNT(DISTINCT l.addr) cbc_latches,
  2      SUM(COUNT( * )) buffers,
  3      MIN(COUNT( * )) min_buffer_per_latch,
  4      MAX(COUNT( * )) max_buffer_per_latch,
  5      ROUND(AVG(COUNT( * ))) avg_buffer_per_latch
  6  FROM      v$Latch_children l
  7  JOIN
  8      x$bh b
  9  ON (l.addr = b.hladdr)
10 WHERE name = 'cache buffers chains'
11 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 |

```
SQL> WITH cbc_latches AS
  2      (SELECT addr, name, sleeps,
  3       rank() over(order by sleeps desc) ranking
  4      FROM v$latch_children
  5     WHERE name = 'cache buffers chains')
  6  SELECT owner, object_name,object_type,
  7        COUNT(distinct l.addr) latches,
  8        SUM(tch) touches
  9      FROM cbc_latches l JOIN x$bh b
 10        ON (l.addr = b.hladdr)
 11      JOIN dba_objects o
 12        ON (b.obj = o.object_id)
 13     WHERE l.ranking <=100
 14   GROUP BY owner, object_name,object_type
 15 ORDER BY sum(tch) DESC;
```

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

```
SQL> ALTER SYSTEM SET db_flash_cache_size=1024M SCOPE=SPFILE;
System altered.
```

```
SQL> ALTER SYSTEM SET
  db_flash_cache_file='/ssdfs/ora/dbfc/g11g_dbfc1.dbf'
  SCOPE=SPFILE;
System altered.
```

```
SQL> shutdown immediate
SQL> startup
ORACLE instance started.
SQL> show parameter flash_cache
NAME          TYPE        VALUE
-----  -----
db_flash_cache_file  string      /ssdfs/ora/dbfc/g11g_dbfc1.dbf
db_flash_cache_size  big integer 1G
```

```
SQL> ALTER TABLE sales_fc_none STORAGE (FLASH_CACHE NONE) ;
Table altered.

SQL> ALTER INDEX sales_fc_pk STORAGE (FLASH_CACHE NONE) ;
Index altered.

SQL> ALTER TABLE sales_fc_default STORAGE (FLASH_CACHE DEFAULT) ;
Table altered.

SQL> ALTER INDEX sales_fc_default_pk STORAGE (FLASH_CACHE DEFAULT) ;
Index altered.

SQL> ALTER TABLE sales_fc_keep STORAGE (FLASH_CACHE KEEP) ;
Table altered.

SQL> ALTER INDEX sales_fc_keep_pk STORAGE (FLASH_CACHE KEEP) ;
Index altered.
```

```

SQL> WITH stats AS (SELECT /*+ materialize */
  2          name, VALUE
  3      FROM  v$sysstat
  4     WHERE  name LIKE 'flash cache%')
 5  SELECT name, VALUE,
 6        ROUND (VALUE * 100 / tot_inserts, 2) pct_of_inserts
 7    FROM (SELECT SUM (VALUE) tot_inserts
 8          FROM stats where name = 'flash cache inserts')
 9  CROSS JOIN stats
10 ORDER BY value DESC
11 /

```

| NAME | VALUE | Pct of
Inserts |
|--|-----------|-------------------|
| flash cache insert skip: exists | 3,224,749 | 551.26 |
| flash cache insert skip: not useful | 1,856,276 | 317.33 |
| flash cache inserts | 584,974 | 100.00 |
| flash cache eviction: aged out | 454,144 | 77.63 |
| flash cache insert skip: DBWR overloaded | 22,202 | 3.80 |
| flash cache insert skip: modification | 1,600 | .27 |
| flash cache eviction: invalidated | 22 | .00 |
| flash cache insert skip: not current | 1 | .00 |
| flash cache eviction: buffer pinned | 0 | .00 |
| flash cache insert skip: corrupt | 0 | .00 |

```
SQL> SELECT owner || '.' || object_name object,
  2      SUM (CASE WHEN b.status LIKE 'flash%' THEN 1 END) flash_blocks,
  3      SUM (CASE WHEN b.status LIKE 'flash%' THEN 0 else 1 END) cache_blocks,
  4      count(*) total_blocks
  5  FROM      v$bh b
  6      JOIN
  7          dba_objects
  8      ON (objd = object_id)
  9 GROUP BY   owner, object_name
10 order by 4 desc ;
```

| Object Name | DBFC
Blocks | Buf
Blocks | Cache
Blocks | Tot
Blocks | Cached |
|--------------------------|----------------|---------------|-----------------|---------------|--------|
| OPSG.SALES_FC_KEEP | 70,450 | 1,325 | 71,775 | | |
| OPSG.SALES_FC_KEEP_PK | 34,155 | 2,325 | 36,480 | | |
| OPSG.SALES_FC_DEFAULT | 13,721 | 969 | 14,690 | | |
| OPSG.SALES_FC_DEFAULT_PK | 12,032 | 1,939 | 13,971 | | |
| OPSG.SALES_FC_PK | | 3,394 | 3,394 | | |
| OPSG.SALES_FC_NONE | | 2,305 | 2,305 | | |

```

SQL> WITH wait_table
  2      AS (SELECT    SUM (total_waits) total_waits,
  3                  SUM (time_waited_micro) total_time,
  4                  MAX(CASE event
  5                      WHEN 'db file sequential read'
  6                          THEN time_waited_micro / total_waits
  7                      END) avg_db_file_micro,
  8                  MAX(CASE event
  9                      WHEN 'db flash cache single block physical read'
 10                          THEN time_waited_micro / total_waits
 11                      END) avg_db_flash_micro,
 12                  SUM(CASE event
 13                      WHEN 'db flash cache single block physical read'
 14                          THEN total_waits
 15                      END) flash_waits
 16              FROM v$system_event
 17              WHERE event IN ('db file sequential read',
 18                               'db flash cache single block physical read'))
 19      SELECT    total_waits,flash_waits,
 20                  ROUND (avg_db_file_micro) avg_disk_time,
 21                  ROUND (avg_db_flash_micro) avg_flash_time,
 22                  total_time,
 23                  ROUND(flash_waits *
 24                      (avg_db_file_micro - avg_db_flash_micro)) time_saved,
 25                  ROUND(flash_waits *
 26                      (avg_db_file_micro - avg_db_flash_micro) * 100
 27                      / (total_waits* avg_db_file_micro),2) pct_io_time_saved
 28      FROM    wait_table
 29  /

```

| Total Waits | Flash Cache Waits | Avg Disk time us | Avg DBFC time us | Total Time us |
|---------------|-------------------|------------------|------------------|---------------|
| 2,817,955 | 1,555,408 | 1,785 | 90 | 2,393,741,738 |
| 2,636,254,812 | | 52.41 | | |

```
CREATE TABLE ssd_partition_demo
(
    id          NUMBER PRIMARY KEY,
    category    VARCHAR2 (1) NOT NULL,
    rating      VARCHAR2 (3) NOT NULL,
    insert_date DATE NOT NULL
)
PARTITION BY RANGE (insert_date)
  INTERVAL ( NUMTOYMINTERVAL (1, 'month') )
  STORE IN (ssd_ts)
(PARTITION cold_data VALUES LESS THAN
  (TO_DATE ('2013-07-01', 'SYYYY-MM-DD'))
  TABLESPACE sas_ts);
```

```

SQL> 1
 1  SELECT partition_name, high_value, tablespace_name
 2    FROM user_tab_partitions
 3*   WHERE table_name = 'SSD_PARTITION_DEMO'
SQL> /

```

| PARTITION | HIGH_VALUE | TABLESPACE |
|-----------|--|------------|
| COLD_DATA | TO_DATE(' 2013-07-01 00:00:00', 'SYDDD-M HH24:MI:SS', 'NLS_CALENDAR=GREGORIA | SAS_TS |
| SYS_P68 | TO_DATE(' 2013-11-01 00:00:00', 'SYDDD-M HH24:MI:SS', 'NLS_CALENDAR=GREGORIA | SSD_TS |
| SYS_P69 | TO_DATE(' 2013-12-01 00:00:00', 'SYDDD-M HH24:MI:SS', 'NLS_CALENDAR=GREGORIA | SSD_TS |
| SYS_P70 | TO_DATE(' 2013-10-01 00:00:00', 'SYDDD-M HH24:MI:SS', 'NLS_CALENDAR=GREGORIA | SSD_TS |
| SYS_P71 | TO_DATE(' 2013-09-01 00:00:00', 'SYDDD-M HH24:MI:SS', 'NLS_CALENDAR=GREGORIA | SSD_TS |
| SYS_P72 | TO_DATE(' 2013-08-01 00:00:00', 'SYDDD-M HH24:MI:SS', 'NLS_CALENDAR=GREGORIA | SSD_TS |
| SYS_P73 | TO_DATE(' 2014-01-01 00:00:00', 'SYDDD-M HH24:MI:SS', 'NLS_CALENDAR=GREGORIA | SSD_TS |
| SYS_P74 | TO_DATE(' 2014-02-01 00:00:00', 'SYDDD-M HH24:MI:SS', 'NLS_CALENDAR=GREGORIA | SSD_TS |

```
DECLARE
    num_not_date      EXCEPTION;
    PRAGMA EXCEPTION_INIT (NUM_NOT_DATE, -932);
    invalid_identifier EXCEPTION;
    PRAGMA EXCEPTION_INIT (invalid_identifier, -904);

    l_highdate        DATE;
BEGIN
    FOR r IN (SELECT table_name,
                      partition_name,
                      high_value
                 FROM user_tab_partitions
                WHERE tablespace_name <> 'SSD_TS')
LOOP
    BEGIN
        -- pull the highvalue out as a date
        EXECUTE IMMEDIATE 'SELECT ' || r.high_value || ' from dual'
        INTO l_highdate;

        IF l_highdate < SYSDATE - 90
        THEN
            EXECUTE IMMEDIATE
                'alter table '
                || r.table_name
                || ' move partition '''
                || r.partition_name
                || '" tablespace sas_ts';
        END IF;
    EXCEPTION
        WHEN num_not_date OR invalid_identifier -- max_value not a date
        THEN
            NULL;
    END;
END LOOP;
END;
```

```
-- Enable/ Check that table is eligible for redefinition
BEGIN
  DBMS_REDEFINITION.CAN_REDEF_TABLE(
    uname      => USER,
    tname      => 'SSD_PARTITION_DEMO',
    options_flag => DBMS_REDEFINITION.CONS_USE_ROWID,
    part_name   => 'SYS_P86');
END;
/
-- Create interim table in the tablespace where we want to move to
CREATE TABLE interim_partition_storage TABLESPACE sas_ts
AS SELECT * FROM ssd_partition_demo PARTITION (sys_p86) WHERE ROWNUM <1;

-- Begin redefinition
BEGIN
  DBMS_REDEFINITION.START_REDEF_TABLE(
    uname      => USER,
    orig_table => 'SSD_PARTITION_DEMO',
    int_table  => 'INTERIM_PARTITION_STORAGE',
    col_mapping => NULL,
    options_flag => DBMS_REDEFINITION.CONS_USE_ROWID,
    part_name   => 'SYS_P86');
END;
/
-- If there are any local indexes create them here
-- Synchronize
BEGIN
  DBMS_REDEFINITION.SYNC_INTERIM_TABLE(
    uname      => USER,
    orig_table => 'SSD_PARTITION_DEMO',
    int_table  => 'INTERIM_PARTITION_STORAGE',
    part_name   => 'SYS_P86');
END;
/
-- Finalize the redefinition (exchange partitions)
BEGIN
  DBMS_REDEFINITION.FINISH_REDEF_TABLE(
    uname      => USER,
    orig_table => 'SSD_PARTITION_DEMO',
    int_table  => 'INTERIM_PARTITION_STORAGE',
    part_name   => 'SYS_P86');
END;
/
```

```

DROP TABLE ap.randomized_sorted PURGE;
CREATE TABLE ap.randomized_sorted (
    key_id      NUMBER(8)
    ,key_date   DATE
    ,key_desc   VARCHAR2(32)
    ,key_sts    NUMBER(2) NOT NULL
)
TABLESPACE ado_cold_data
NOLOGGING;

TRUNCATE TABLE ap.randomized_sorted;

SET TIMING ON
DECLARE
    TYPE tnb_KeyID IS
        TABLE OF ap.randomized_sorted.key_id%TYPE
        INDEX BY PLS_INTEGER;
    TYPE tdt_KeyDate IS
        TABLE OF ap.randomized_sorted.key_date%TYPE
        INDEX BY PLS_INTEGER;
    TYPE tvc_KeyDesc IS
        TABLE OF ap.randomized_sorted.key_desc%TYPE
        INDEX BY PLS_INTEGER;
    TYPE tnb_KeySts IS
        TABLE OF ap.randomized_sorted.key_sts%TYPE
        INDEX BY PLS_INTEGER;
    tb_KeyId    tnb_KeyID;
    tb_KeyDate  tdt_KeyDate;
    tb_KeyDesc  tvc_KeyDesc;
    tb_KeySts   tnb_KeySts;
    ctr         PLS_INTEGER;
    nMaxIterations CONSTANT PLS_INTEGER := 10000000;

```

BEGIN

```

-- Populate collections
FOR ctr IN 1..nMaxIterations
LOOP
    tb_KeyID(ctr) := ctr;
    tb_KeyDate(ctr) :=
        (TO_DATE('12/31/2014','mm/dd/yyyy') - DBMS_RANDOM.VALUE(1,3650));
    tb_KeyDesc(ctr) := LPAD(' ',DBMS_RANDOM.VALUE(1,32),
SUBSTR('abcdefghijklmnopqrstuvwxyz',DBMS_RANDOM.VALUE(1,26), 1));
    CASE MOD(ROUND(DBMS_RANDOM.VALUE(1,nMaxIterations),0),50)
        WHEN  0 THEN tb_Keysts(ctr) := 30;
        WHEN  1 THEN tb_Keysts(ctr) :=40;
        WHEN  2 THEN tb_Keysts(ctr) :=40;
        WHEN  3 THEN tb_Keysts(ctr) :=40;
        WHEN  4 THEN tb_Keysts(ctr) :=40;
        WHEN  5 THEN tb_Keysts(ctr) :=20;
        WHEN  6 THEN tb_Keysts(ctr) :=40;
        WHEN  7 THEN tb_Keysts(ctr) :=40;
        WHEN  8 THEN tb_Keysts(ctr) :=40;
        WHEN  9 THEN tb_Keysts(ctr) :=30;
        WHEN 10 THEN tb_Keysts(ctr) :=40;
        WHEN 11 THEN tb_Keysts(ctr) :=20;
        WHEN 12 THEN tb_Keysts(ctr) :=10;
        WHEN 13 THEN tb_Keysts(ctr) :=15;
        WHEN 14 THEN tb_Keysts(ctr) :=30;
        ELSE tb_Keysts(ctr) :=50;
    END CASE;
END LOOP;

-- Load table from collection items
FORALL ctr IN 1..nMaxIterations
    INSERT INTO ap.randomized_sorted(key_id, key_date, key_desc, key_sts)
    VALUES (tb_KeyID(ctr), tb_KeyDate(ctr), tb_KeyDesc(ctr), tb_Keysts(ctr));

COMMIT;

EXCEPTION
    WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('Fatal error detected!');
END;
/

```

```
ALTER TABLE ap.randomized_sorted
  ADD CONSTRAINT randomized_sorted_pk
  PRIMARY KEY (key_id)
  USING INDEX (
    CREATE UNIQUE INDEX ap.randomized_sorted_pk
      ON ap.randomized_sorted(key_id)
      TABLESPACE ado_cold_idx
      NOLOGGING
      PARALLEL 4);

ALTER INDEX ap.randomized_sorted_pk LOGGING;
```

```
CREATE BITMAP INDEX ap.randomized_sorted_sts_bix
  ON ap.randomized_sorted (key_sts)
  TABLESPACE ado_cold_idx
  NOLOGGING PARALLEL(4);
```

```
ALTER INDEX ap.randomized_sorted_sts_bix LOGGING;
```

```

DROP TABLE ap.randomized_parted PURGE;
CREATE TABLE ap.randomized_parted (
    key_id      NUMBER(8)
    ,key_date   DATE
    ,key_desc   VARCHAR2(32)
    ,key_sts    NUMBER(2) NOT NULL
)
PARTITION BY RANGE(key_date) (
    PARTITION p1_frigid
        VALUES LESS THAN (TO_DATE('2010-01-01', 'YYYY-mm-dd'))
        TABLESPACE ado_cold_data
    ,PARTITION p2_cool
        VALUES LESS THAN (TO_DATE('2013-01-01', 'YYYY-mm-dd'))
        TABLESPACE ado_cool_data
    ,PARTITION p3_warm
        VALUES LESS THAN (TO_DATE('2014-01-01', 'YYYY-mm-dd'))
        TABLESPACE ado_warm_data
    ,PARTITION p4_hot
        VALUES LESS THAN (TO_DATE('2014-07-01', 'YYYY-mm-dd'))
        TABLESPACE ado_hot_data
    ,PARTITION p5_radiant
        VALUES LESS THAN (MAXVALUE)
        TABLESPACE ap_data
)
NOLOGGING
PARALLEL 4
;

INSERT /*+ APPEND NOLOGGING PARALLEL(4) */ INTO ap.randomized_parted
SELECT * FROM ap.randomized_sorted;

COMMIT;

ALTER TABLE ap.randomized_parted
    DROP CONSTRAINT randomized_parted_pk;

DROP INDEX ap.randomized_parted_pk;

ALTER TABLE ap.randomized_parted
    ADD CONSTRAINT randomized_parted_pk
    PRIMARY KEY (key_id)
    USING INDEX (
        CREATE UNIQUE INDEX ap.randomized_parted_pk
            ON ap.randomized_parted(key_id)
            TABLESPACE ap_idx
            NOLOGGING
            PARALLEL 4
    );

```

Table created.

```
-----  
-- Create local partitioned indexes on KEY_STS  
-----  
DROP INDEX ap.randomized_parted_loc_sts;  
CREATE INDEX ap.randomized_parted_loc_sts  
    ON ap.randomized_parted (key_sts)  
    STORAGE (INITIAL 10M)  
    LOCAL (  
        PARTITION lx1_frigid      TABLESPACE ado_cold_idx  
        ,PARTITION lx2_cool       TABLESPACE ado_cool_idx  
        ,PARTITION lx3_warm       TABLESPACE ado_warm_idx  
        ,PARTITION lx4_hot        TABLESPACE ado_hot_idx  
        ,PARTITION lx5_radiant    TABLESPACE ap_idx  
    );  
  
-----  
-- Create a global partitioned index on KEY_DESC  
-----  
CREATE INDEX ap.randomized_parted_glb_desc  
    ON ap.randomized_parted (key_desc)  
    STORAGE (INITIAL 10M)  
    GLOBAL;
```

```
CREATE TABLE ap.randomized_parted (
    key_id      NUMBER(8)
    ,key_date   DATE
    ,key_desc   VARCHAR2(32)
    ,key_sts    NUMBER(2) NOT NULL
)
INDEXING OFF
PARTITION BY RANGE(key_date) (
    PARTITION p1_frigid
        VALUES LESS THAN (TO_DATE('2010-01-01', 'YYYY-mm-dd'))
        TABLESPACE ado_cold_data
    ,PARTITION p2_cool
        VALUES LESS THAN (TO_DATE('2013-01-01', 'YYYY-mm-dd'))
        TABLESPACE ado_cool_data
    ,PARTITION p3_warm
        VALUES LESS THAN (TO_DATE('2014-01-01', 'YYYY-mm-dd'))
        TABLESPACE ado_warm_data
        INDEXING ON
    ,PARTITION p4_hot
        VALUES LESS THAN (TO_DATE('2014-07-01', 'YYYY-mm-dd'))
        TABLESPACE ado_hot_data
        INDEXING ON
    ,PARTITION p5_radiant
        VALUES LESS THAN (MAXVALUE)
        TABLESPACE ap_data
        INDEXING ON
)
NOLOGGING
PARALLEL 4;
```

```
ALTER TABLE oe.order_items DROP CONSTRAINT order_items_pk;
ALTER TABLE oe.order_items
  ADD CONSTRAINT order_items_pk
    PRIMARY KEY (order_id, line_item_id)
  USING INDEX (
    CREATE UNIQUE INDEX oe.order_items_pk_idx
      ON oe.order_items (order_id, line_item_id)
      TABLESPACE example
      COMPRESS);
```

```
CREATE BITMAP INDEX oe.order_item_qty_bjx
ON oe.order_items (quantity)
  FROM oe.order_items OI, oe.product_information P
 WHERE OI.product_id = P.product_id
TABLESPACE example;
```

```

SET LINESIZE 170
SET PAGESIZE 20000
COL owner      FORMAT A08      HEADING "Owner"
COL index_owner FORMAT A08      HEADING "Owner"
COL table_name  FORMAT A24      HEADING "Table Name"
COL index_name  FORMAT A30      HEADING "Index Name"
COL partition_name FORMAT A12    HEADING "Partition Name"
COL tsp_name    FORMAT A12      HEADING "Tablespace|Name"
COL status      FORMAT A08      HEADING "Status"
COL num_rows    FORMAT 999,999,999 HEADING "Row|Count"
COL distinct_keys FORMAT 999,999,999 HEADING "Distinct|Keys"
COL clusfctr   FORMAT 99,999,999 HEADING "Clust|Factor"
COL blevel      FORMAT 99999     HEADING "Index|Depth"
COL leaf_blocks FORMAT 9,999,999 HEADING "Leaf|Blocks"
COL avg_lbpk    FORMAT 999,999     HEADING "Avg Leaf|Blks/Key"
COL avg_dbpk    FORMAT 999,999     HEADING "Avg Data|Blks/Key"
TTITLE "Index Performance Metadata| (from DBA_INDEXES)"
SELECT
  owner
 ,table_name
 ,index_name
 ,tablespace_name tsp_name
 ,status
 ,num_rows
 ,distinct_keys
 ,clustering_factor clusfctr
 ,blevel
 ,leaf_blocks
 ,avg_leaf_blocks_per_key avg_lbpk
 ,avg_data_blocks_per_key avg_dbpk
FROM dba_indexes
WHERE owner = 'AP'
ORDER BY 1,2,3
;
TTITLE OFF
Index Performance Metadata

```

(from DBA\_INDEXES)

| Owner | Table Name | Index Name | Tablespace | | | Row Count | Distinct Keys | Clust Factor | Index Depth | Leaf Avg Leaf Avg Data | | |
|-------|---------------------|-------------------------|--------------|--------|------------|------------|---------------|--------------|-------------|------------------------|----------|----------|
| | | | Name | Status | Blks/Key | | | | | Blks/Key | Blks/Key | Blks/Key |
| AP | EST_SALES | EST_SALES_CITYST_IDX | ADO_COLD_IDX | VALID | 680,352 | 29,768 | 526,867 | 2 | 1,124 | 1 | 17 | |
| AP | EST_SALES | EST_SALES_CITY_IDX | ADO_COLD_IDX | VALID | 680,352 | 18,792 | 526,845 | 2 | 1,948 | 1 | 28 | |
| AP | EST_SALES | EST_SALES_PK_IDX | ADO_COLD_IDX | VALID | 680,352 | 680,352 | 74,988 | 2 | 1,519 | 1 | 1 | |
| AP | EST_SALES | EST_SALES_STATE_IDX | ADO_COLD_IDX | VALID | 680,352 | 62 | 89,808 | 2 | 1,047 | 16 | 1,448 | |
| AP | EST_SALES | EST_SALES_TIMEZONE_IDX | ADO_COLD_IDX | VALID | 649,680 | 26 | 82,138 | 2 | 1,001 | 38 | 3,159 | |
| AP | EST_SALES | EST_SALES_ZIPCITYST_IDX | ADO_COLD_IDX | VALID | 680,352 | 42,522 | 680,352 | 2 | 1,193 | 1 | 16 | |
| AP | EST_SALES | EST_SALES_ZIPCODE_IDX | ADO_COLD_IDX | VALID | 680,352 | 42,264 | 680,352 | 2 | 1,118 | 1 | 16 | |
| AP | INVOICES | INVOICES_CUST_IDX | AP_IDX | VALID | 500 | 1 | 3 | 0 | 1 | 1 | 3 | |
| AP | INVOICES | INVOICES_PK_IDX | AP_IDX | VALID | 500 | 500 | 3 | 0 | 1 | 1 | 1 | |
| AP | INVOICE_ITEMS | INVOICE_ITEMS_PK_IDX | AP_IDX | VALID | 47,500 | 47,500 | 191 | 1 | 118 | 1 | 1 | |
| AP | INVOICE_ITEMS | INVOICE_ITEMS_PROD_IDX | AP_IDX | VALID | 47,500 | 5 | 955 | 1 | 140 | 28 | 191 | |
| AP | RANDOMIZED_PARTED | RANDOMIZED_PARTED_PK | ADO_COLD_IDX | VALID | 10,365,139 | 10,365,139 | 6,722,752 | 2 | 28,733 | 1 | 1 | |
| AP | RANDOMIZED_SORTED | RANDOMIZED_SORTED_PK | ADO_COLD_IDX | VALID | 10,194,228 | 10,194,228 | 76,739 | 2 | 22,556 | 1 | 1 | |
| AP | RANDOMIZED_UNSORTED | RANDOMIZED_UNSORTED_PK | ADO_COLD_IDX | VALID | 10,395,465 | 10,395,465 | 10,379,934 | 2 | 23,004 | 1 | 1 | |
| AP | VENDORS | VENDORS_PK_IDX | AP_IDX | VALID | 164 | 164 | 3 | 0 | 1 | 1 | 1 | |

```
DROP TABLE ap.randomized_unsorted PURGE;
CREATE TABLE ap.randomized_unsorted (
    key_id      NUMBER(8)
    ,key_date   DATE
    ,key_desc   VARCHAR2(32)
    ,key_sts    NUMBER(2) NOT NULL
)
TABLESPACE ado_cold_data
NOLOGGING
PCTFREE 0;

TRUNCATE TABLE ap.randomized_unsorted;

INSERT /*+ APPEND NOLOGGING PARALLEL(4) */ INTO ap.randomized_unsorted
SELECT /*+ PARALLEL(4) */ * FROM ap.randomized_sorted
ORDER BY key_desc DESC;

COMMIT;

ALTER TABLE ap.randomized_unsorted
    DROP CONSTRAINT randomized_unsorted_pk;

DROP INDEX ap.randomized_unsorted_pk;

ALTER TABLE ap.randomized_unsorted
    ADD CONSTRAINT randomized_unsorted_pk
    PRIMARY KEY (key_id)
    USING INDEX (
        CREATE UNIQUE INDEX ap.randomized_unsorted_pk
        ON ap.randomized_unsorted(key_id)
        TABLESPACE ado_cold_idx
        NOLOGGING
        PARALLEL 4
    );
ALTER INDEX ap.randomized_unsorted_pk LOGGING;
```

```
ALTER INDEX sh.zm_customers_marital_bix INVISIBLE;

DROP INDEX sh.zm_customers_marital_bix;
CREATE BITMAP INDEX sh.zm_customers_marital_bix
  ON sh.zm_customers (cust_marital_status)
  TABLESPACE example
  INVISIBLE;
```

```

SELECT /*+ INDEX(sh.zm_customers_marital_bix) */
       cust_state_province, COUNT(*)
  FROM sh.zm_customers
 WHERE cust_marital_status = 'widow'
 GROUP BY cust_state_province
 ORDER BY cust_state_province;

```

Plan hash value: 2056234527

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|-----|-------------------|--------------|------|-------|-------------|----------|
| 0 | SELECT STATEMENT | | 145 | 2465 | 424 (1) | 00:00:01 |
| 1 | SORT GROUP BY | | 145 | 2465 | 424 (1) | 00:00:01 |
| * 2 | TABLE ACCESS FULL | ZM_CUSTOMERS | 3461 | 58837 | 423 (1) | 00:00:01 |

Predicate Information (identified by operation id):
 2 - filter("CUST\_MARITAL\_STATUS"='widow')

ALTER SESSION SET optimizer\_use\_invisible\_index = TRUE;

```

SELECT /* +INDEX(sh.zm_customers_marital_idx) */
       cust_city, COUNT(*)
  FROM sh.zm_customers
 WHERE marital_status = 'S';

```

Plan hash value: 2120557835

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|-----|-------------------------------------|--------------------------|------|-------|-------------|----------|
| 0 | SELECT STATEMENT | | 58 | 986 | 19 (6) | 00:00:01 |
| 1 | SORT GROUP BY | | 58 | 986 | 19 (6) | 00:00:01 |
| 2 | TABLE ACCESS BY INDEX ROWID BATCHED | ZM_CUSTOMERS | 75 | 1275 | 18 (0) | 00:00:01 |
| * 4 | BITMAP CONVERSION TO ROWIDS | | | | | |
| * 4 | BITMAP INDEX SINGLE VALUE | ZM_CUSTOMERS_MARITAL_BIX | | | | |

Predicate Information (identified by operation id):
 4 - access("CUST\_MARITAL\_STATUS"='widow')

```
$> cd sqlt/install
$> sqlplus / as sysdba
SQL> START sqcreate.sql

zip warning: name not matched: *_sq*.log

zip error: Nothing to do! (SQLT_installation_logs_archive.zip)
zip warning: name not matched: *_ta*.log

zip error: Nothing to do! (SQLT_installation_logs_archive.zip)

PL/SQL procedure successfully completed.

.
.
.
(many lines of output suppressed for sake of brevity)
.
.
.
SQUTLTEST completed.
adding: 150523163243_10_squtltest.log (deflated 59%)

SQLT users must be granted SQLT_USER_ROLE before using this tool.

SQCREATE completed. Installation completed successfully.
```

```
$> cd sqlt/install  
$> sqlplus / as sysdba  
SQL> START sqcsilent2.sql '' sqlxplain USERS TEMP '' T
```

```
# cd sqlt
# sqlplus apps
SQL> START [path]sqltxecute.sql [path]scriptname [sqltxplain_password]
SQL> START run/sqltxecute.sql input/sample/script1.sql sqltxplain_password
```

```

-- execute sqlt xecute as sh passing script name
-- cd sqlt
-- #sqlplus sh
-- SQL> start run/sqltxecute.sql input/sample/script1.sql

REM Optional ALTER SESSION commands
REM ~~~~~

--ALTER SESSION SET statistics_level = ALL;

REM Optional Binds
REM ~~~~~

VAR b1 NUMBER;
EXEC :b1 := 10;

REM SQL statement to be executed
REM ~~~~~

SELECT /*+ GATHER_PLAN_STATISTICS MONITOR BIND_AWARE */
       /* ^^^unique_id */
       s1.channel_id,
       SUM(p.prod_list_price) price
  FROM products p,
       sales s1,
       sales s2
 WHERE s1.cust_id = :b1
   AND s1.prod_id = p.prod_id
   AND s1.time_id = s2.time_id
 GROUP BY
       s1.channel_id;
/
/
REM Notes:
REM 1. SQL must contain token: /* ^^^unique_id */
REM 2. Do not replace ^^^unique_id with your own tag.
REM 3. SQL may contain CBO Hints, like:
REM      /*+ gather_plan_statistics monitor bind_aware */

```

```
# cd sqlt
# sqlplus apps
SQL> START [path]sqltxecute.sql [path]scriptname [sqltxplain_password]
16:43:56 APPS@BWEBSPR5 > START run/sqltxecute.sql input/input_pp.sql f2cb2w08

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.01

Parameter 1:
SCRIPT name which contains SQL and its binds (required)
. . .
```

```
SQL> SELECT * FROM SQLTXADMIN.sqlt$log_v;  
  
TIME      LINE  
-----  
16:49:11 sqlt$a: tool version: 12.1.11  
16:49:11 sqlt$a: script version: 12.1.11  
16:49:11 sqlt$a: -> common_initialization  
16:49:11 sqlt$a: ALTER SESSION SET NLS_NUMERIC_CHARACTERS = ".,"  
16:49:11 sqlt$a: ALTER SESSION SET NLS_SORT = BINARY  
.  
16:49:37 sqlt$d: task name = "sqlt_s93766_mem"  
16:49:37 sqlt$d: <- create_tuning_task_text  
16:49:37 sqlt$d: -> collect_tuning_sets_mem  
16:49:37 sqlt$d: sqlt$gv$sql_plan: 1518346075
```

181 rows selected.

SQL>

```
select sql_id  
  from DBA_HIST_SQLTEXT  
 where sql_text like '%where DATE_VALUE > sysdate -180% ';
```

```
$> START run/sqltxecute.sql input/input_pp.sql f2cb2w08  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:00.01
```

```
-- This creates the script below that must be run to create the profile:  
START coe_xfr_sql_profile.sql gqmv2hr133bjv 163726467  
  
--This script finally creates the profile:  
@coe_xfr_sql_profile_gqmv2hr133bjv_4271579545.sql
```

```
SQL> SELECT COUNT(*) FROM dba_2pc_pending;
```

```
SQL> SET HEADING OFF
SQL> SELECT 'ROLLBACK FORCE '''||local_tran_id||''';' FROM dba_2pc_pending;
```

```
SQL> SELECT 'EXECUTE DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY(''':''||local_tran_id||'');COMMIT;'  
  FROM dba_2pc_pending;
```

ORA-01591: lock held by in-doubt distributed transaction 1.21.17

```
SQL> SELECT local_tran_id, state FROM dba_2pc_pending;
```

| LOCAL_TRAN_ID | STATE |
|---------------|----------|
| 1704.34.52393 | prepared |

```
SQL> SELECT KTUXEBUSN, KTUXESLT, KTUXESQN, /* Transaction ID */  
KTUXESTA Status,  
KTUXECFL Flags  
FROM x$ktuxe  
WHERE ktuxesta != 'INACTIVE'  
AND ktuxeusn = 1704; <== this is the rollback segment#
```

```
SQL> rollback force '1704.34.52393';
ORA-02058: no prepared transaction found with ID 1704.34.52393
```

```
SQL> SET TRANSACTION USE ROLLBACK SEGMENT SYSTEM;
```

```
SQL> DELETE FROM sys.pending_trans$ WHERE local_tran_id = '1704.34.52393';
SQL> DELETE FROM sys.pending_sessions$ WHERE local_tran_id = '1704.34.52393';
SQL> DELETE FROM sys.pending_sub_sessions$ WHERE local_tran_id = '1704.34.52393';
SQL> COMMIT;
```

ORA-1591: lock held by in-doubt distributed transaction '1704.34.52393'

```
SQL> SELECT *
      FROM dba_2pc_pending
     WHERE local_tran_id='1704.34.52393';
```

No rows returned

```
SQL> SELECT
  KTUXEUSN,
  KTUXESLT,
  KTUXESQN,
  KTUXESTA Status,
  KTUXECFL Flags
  FROM x$ktuxe
 WHERE ktuxesta != 'INACTIVE'
   AND ktixeusn= 1704;
```

| KTUXEUSN | KTUXESLT | KTUXESQN | STATUS | FLAGS |
|----------|----------|----------|----------|------------------------|
| 1704 | 34 | 52393 | PREPARED | SCO COL REV DEAD |

```
SQL> COMMIT FORCE '1704.34.52393';
ORA-02058: no prepared transaction found with ID 1704.34.52393
```

```
-- Disable distributed recovery to prevent the RECO background process
-- from handling this dummy row
SQL> Alter System Disable Distributed Recovery;

SQL> INSERT INTO pending_trans$ (
local_tran_id,
global_tran_fmt,
global_oracle_id,
state,
status,
session_vector,
reco_vector,
type#,
fail_time,
reco_time)
VALUES(
'1704.34.52393', /* ONLY THIS ROW SHOULD BE MODIFIED WITH YOUR LOCAL TRAN ID*/
306206,
'XXXXXXXX.12345.1.2.3',
'prepared',
'P',
hextoraw( '00000001' ),
hextoraw( '00000000' ),
0,
sysdate,
sysdate );

SQL> INSERT INTO pending_sessions$
VALUES(
'1704.34.52393',
1,
hextoraw('05004F003A1500000104'),
'C',
0,
30258592,
 '',
146
);

SQL> COMMIT;

SQL> COMMIT FORCE '1704.34.52393';
```

```
SQL> DELETE FROM pending_trans$ WHERE local_tran_id='1704.34.52393';
SQL> DELETE FROM pending_sessions$ WHERE local_tran_id='1704.34.52393';
SQL> COMMIT;
SQL> ALTER SYSTEM ENABLE DISTRIBUTED RECOVERY;
```

```
SQL> EXEC DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('69.24.6020053');
```

```
SQL> SELECT local_tran_id, state FROM dba_2pc_pending;
```

| LOCAL_TRAN_ID | STATE |
|---------------|----------|
| 7.12.102 | prepared |

```
SQL> SELECT
  KTUXEUSN,
  KTUXESLT,
  KTUXESQN,
  KTUXESTA Status,
  KTUXECFL Flags
  FROM x$ktuxe
 WHERE ktuxesta != 'INACTIVE'
   AND ktixeusn = 7; --HERE WE MUST CHANGE TO THE UNDO BLOCK. IN THIS CASE IT'S NUMBER 7
```

| KTUXEUSN | KTUXESLT | KTUXESQN | STATUS | FLAGS |
|----------|----------|----------|----------|------------------|
| 7 | 12 | 102 | PREPARED | SCO COL REV DEAD |

SQL> rollback force '7.12.102';

· · ·
{ Session HANGs }

SQL> commit force '7.12.102';

· · ·
{ Session HANGs }

```
SQL> SELECT event
      FROM gv$session_wait
     WHERE event LIKE '%free%global%entry%' ;
```

```
SQL> SELECT a.sql_text, s.osuser, s.username,s.sid
      FROM v$transaction t, v$session s, v$sqlarea a
     WHERE s.taddr = t.addr
       AND a.address = s.prev_sql_addr
       AND t.xidusn = 7
       AND t.xidslot =12
       AND t.xidsqn = 102;
```

```
SQL> EXECUTE DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('7.12.102');COMMIT;
*
ERROR at line 1:
ORA-06510: PL/SQL: unhandled user-defined exception
ORA-06512: at "SYS.DBMS_TRANSACTION", line 94
ORA-06512: at line 1
```

```
SQL> DELETE FROM sys.pending_sub_sessions$ WHERE local_tran_id = '7.12.102';
SQL> DELETE FROM sys.pending_sessions$ WHERE local_tran_id = '7.12.102';
SQL> DELETE FROM sys.pending_trans$ WHERE local_tran_id = '7.12.102';
SQL> COMMIT;
```

```
--Disable distributed recovery to not let the RECO background process work on this dummy row
SQL> ALTER SYSTEM DISABLE DISTRIBUTED RECOVERY;

SQL> INSERT INTO pending_trans$ (
local_tran_id,
global_tran_fmt,
global_oracle_id,
state,
status,
session_vector,
reco_vector,
type#,
fail_time,
reco_time)
VALUES(
'7.12.102', /* ONLY THIS ROWS SHOULD BE MODIFIED WITH YOUR LOCAL TRAN ID*/
306206,
'XXXXXXXX.12345.1.2.3',
'prepared',
'P',
hextoraw( '00000001' ),
hextoraw( '00000000' ),
0,
sysdate,
sysdate );

SQL> INSERT INTO pending_sessions$
VALUES(
'7.12.102',
1,
hextoraw('05004F003A1500000104'),
'C',
0,
30258592,
',
146
);

SQL> COMMIT;
```

```
SQL> EXEC DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('7.12.102');
```

```
SELECT a.sql_text, s.osuser, s.username
  FROM v$transaction t, v$session s, v$sqlarea a
 WHERE s.taddr = t.addr
   AND a.address = s.prev_sql_addr
   AND t.xidusn = (select local_tran_id from dba_2pc_pending)
   AND t.xidslot = <second-part-of -transaction-ID>
   AND t.xidsqn = <third-part-of -transaction-ID>;
```

```
SET HEADING OFF
SELECT 'ROLLBACK FORCE '''||local_tran_id||''' ;
  FROM dba_2pc_pending;
```

```
SELECT
'EXECUTE DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('' || local_tran_id || '');COMMIT;
FROM dba_2pc_pending;
```

```
SELECT a.sql_text, s.osuser, s.username
  FROM v$transaction t, v$session s, v$sqlarea a
 WHERE s.taddr = t.addr
   AND a.address = s.prev_sql_addr
   AND t.xidusn = 3335
   AND t.xidslot =14
   AND t.xidsqn = 632719;
```
