



Integrated Cloud Applications & Platform Services



Oracle Database 12c: Performance Management and Tuning

Student Guide – Volume II
D79236GC10
Edition 1.0 | December 2016 | D86568

Learn more from Oracle University at education.oracle.com

Authors

Donna Keesling
James Spiller

Technical Contributors and Reviewers

Harald van Breederode
Yio Liong Liew
Sailaja Pasupuleti
Naoki Kato
Joel Goodman
Joe Fong
Ira Singer
Herbert Bradbury
Gerlinde Frenzen
Christopher D Andrews
Branislav Valny
Anthony Woodell
Andy Fortunak

Editor

Aju Kumar

Graphic Designer

Divya Thallap

Publishers

Pavithran Adka
Nita Brozowski
Jobi Varghese

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Disclaimer

This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

Restricted Rights Notice

If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Contents

1 Introduction

Course Objectives 1-2

Organization 1-3

Agenda 1-4

Topics Not Included in This Course 1-6

What Is Performance Management? 1-7

Who Tunes? 1-9

What Does the DBA Tune? 1-10

Types of Tuning 1-11

Tuning Methodology 1-12

Effective Tuning Goals 1-14

General Tuning Session 1-16

Quiz 1-18

Summary 1-19

2 Basic Tuning Diagnostics

Objectives 2-2

Performance Tuning: Diagnostics 2-3

Performance Tuning: Features and Tools 2-4

Tuning Objectives 2-5

Top Timed Events 2-6

DB Time 2-7

CPU and Wait Time Tuning Dimensions 2-8

Time Model: Overview 2-9

Time Model Statistics Hierarchy 2-10

Time Model: Example 2-12

Quiz 2-13

Dynamic Performance Views 2-14

Dynamic Performance Views: Usage Examples 2-15

Dynamic Performance Views: Considerations 2-16

Statistic Levels 2-17

Instance Activity and Wait Event Statistics 2-19

System Statistic Classes 2-21

Displaying Statistics 2-22

Displaying SGA Statistics 2-24

Wait Events 2-25

Using the V\$EVENT_NAME View 2-26

Wait Classes 2-27

Displaying Wait Event Statistics	2-28
Commonly Observed Wait Events	2-30
Using the V\$SESSION_WAIT View	2-31
Precision of System Statistics	2-33
Quiz	2-34
Enterprise Manager: Overview	2-35
Oracle Enterprise Manager Database Express Architecture	2-36
Configuring Enterprise Manager Database Express	2-37
Accessing the Enterprise Manager Database Express Database Home Page	2-38
Viewing Performance Information on the Database Home Page	2-39
Viewing the Performance Hub Page	2-41
Oracle Enterprise Manager Cloud Control Components	2-43
Using Features of the Oracle Management Packs	2-44
Viewing the Alert Log	2-46
Using Alert Log Information as an Aid in Tuning	2-48
Administering the DDL Log File	2-49
Understanding the Debug Log File	2-50
User Trace Files	2-51
Background Processes Trace Files	2-52
Quiz	2-53
Summary	2-54
Practice 2 Overview: Using Basic Tools	2-55

3 Using Automatic Workload Repository

Objectives	3-2
Automatic Workload Repository: Overview	3-3
Automatic Workload Repository Data	3-4
Workload Repository	3-5
AWR Administration	3-6
AWR Snapshot Purging Policy	3-7
Managing Snapshots with PL/SQL	3-8
AWR Snapshot Settings	3-9
Manual AWR Snapshots	3-10
Generating AWR Reports	3-11
Generating AWR Reports by Using SQL*Plus	3-12
Reading the AWR Report	3-13
Statspack and AWR Reports	3-14
Reading a Statspack or an AWR Report	3-15
Compare Periods: Benefits	3-16
Snapshots and Periods Comparisons	3-17
Compare Periods: Results	3-18

Compare Periods: Report 3-19
Quiz 3-20
Summary 3-21
Practice 3 Overview: Using AWR-Based Tools 3-22

4 Defining the Scope of Performance Issues

Objectives 4-2
Defining the Problem 4-3
Limit the Scope 4-4
Determining Tuning Priorities 4-5
Common Tuning Problems 4-6
Tuning Life Cycle Phases 4-8
Tuning During the Life Cycle 4-9
Application Design and Development 4-10
Testing: Database Configuration 4-11
Deployment 4-12
Production 4-13
Migration, Upgrade, and Environment Changes 4-14
ADDM Tuning Session 4-15
Performance Versus Business Requirements 4-16
Performance Tuning Resources 4-17
Filing a Performance Service Request 4-18
Monitoring and Tuning Tools: Overview 4-19
Quiz 4-21
Summary 4-22
Practice 4 Overview: Identifying the Problem 4-23

5 Using Metrics and Alerts

Objectives 5-2
Metrics and Alerts 5-3
Limitation of Base Statistics 5-4
Typical Delta Tools 5-5
Oracle Database Metrics 5-6
Benefits of Metrics 5-7
Viewing Metric History Information 5-8
Viewing Metric Details 5-9
Statistic Histograms 5-10
Histogram Views 5-11
Server-Generated Alerts 5-12
Alert Usage Model 5-13
Metric and Alert Views 5-14

Quiz 5-15
Summary 5-16
Practice Overview 5: Working with Metrics 5-17

6 Using Baselines

Objectives 6-2
Comparative Performance Analysis with AWR Baselines 6-3
Automatic Workload Repository Baselines 6-4
AWR Baselines 6-5
Types of Baselines 6-6
Moving Window Baseline 6-7
Baselines in Performance Page Settings 6-8
Baseline Templates 6-9
Creating AWR Baselines 6-10
Creating a Single AWR Baseline 6-11
Creating a Repeating Baseline and Template 6-12
Managing Baselines by Using the DBMS_WORKLOAD_REPOSITORY
 Package 6-13
Generating a Baseline Template for a Single Time Period 6-14
Creating a Repeating Baseline Template 6-15
Baseline Views 6-16
Performance Monitoring and Baselines 6-17
Defining Alert Thresholds Using a Static Baseline 6-19
Configuring a Basic Set of Thresholds 6-20
Quiz 6-21
Summary 6-22
Practice 6: Overview Using AWR Baselines 6-23

7 Using AWR-Based Tools

Objectives 7-2
Automated Maintenance Tasks 7-3
Maintenance Windows 7-4
Default Maintenance Plan 7-5
Automated Maintenance Task Priorities 7-6
Configuring Automated Maintenance Tasks 7-7
ADDM Performance Monitoring 7-8
ADDM and Database Time 7-9
DB Time-Graph and ADDM Methodology 7-10
Top Performance Issues Detected 7-12
Observing ADDM Findings 7-13
ADDM Analysis Results 7-14

ADDM Recommendations	7-15
Creating a Manual ADDM Task	7-16
Changing ADDM Attributes	7-17
Retrieving ADDM Reports by Using SQL	7-18
Quiz	7-19
AWR Compare Periods Report: Review	7-20
Compare Periods ADDM: Analysis	7-21
Workload Compatibility	7-22
Comparison Modes	7-23
Report: Configuration	7-24
Report: Finding	7-25
Report: Resource CPU and I/O	7-26
Report: Resource Memory	7-27
Using the DBMS_ADDM Package	7-28
Quiz	7-30
Active Session History: Overview	7-31
Active Session History: Mechanics	7-32
ASH Sampling: Example	7-33
Accessing ASH Data	7-34
Analyzing the ASH Data	7-35
Generating ASH Reports	7-36
Executing the ASH Report Script	7-37
ASH Report: General Section	7-38
ASH Report Structure	7-39
ASH Report: Activity Over Time	7-40
Additional Automatic Workload Repository Views	7-41
Quiz	7-42
Emergency Monitoring: Challenges	7-43
Emergency Monitoring: Goals	7-44
Real-Time ADDM: Challenges	7-46
Real-Time ADDM: Goals	7-47
Real-Time ADDM in the Database	7-49
Using Real-Time ADDM	7-51
Viewing Real-Time ADDM Results	7-52
Quiz	7-53
Summary	7-54
Practice 7 Overview: Using AWR-Based Tools	7-55

8 Real-Time Database Operation Monitoring

Objectives 8-2

Overview 8-3

Use Cases	8-4
Defining a DB Operation	8-5
Scope of a Composite DB Operation	8-6
Database Operation Concepts	8-7
Identifying a Database Operation	8-8
Enabling Monitoring of Database Operations	8-9
Identifying, Starting, and Completing a Database Operation	8-10
Monitoring the Progress of a Database Operation	8-11
Monitoring Load Database Operations	8-12
Monitoring Load Database Operation Details	8-13
Database Operation View: V\$SQL_MONITOR	8-14
Database Operation Views	8-15
Reporting Database Operations by Using Functions	8-16
Database Operation Tuning	8-17
Quiz	8-18
Summary	8-20
Practice 8 Overview: Real-Time Database Operation Monitoring	8-21

9 Monitoring Applications

Objectives	9-2
What Is a Service?	9-3
Service Attributes	9-4
Service Types	9-5
Creating Services	9-6
Managing Services in a Single-Instance Environment	9-7
Where Are Services Used?	9-8
Using Services with Client Applications	9-9
Using Services with the Resource Manager	9-10
Using Enterprise Manager to Manage Consumer Group Mappings	9-11
Services and the Resource Manager: Example	9-12
Using Enterprise Manager to Create a Job Class	9-13
Using Enterprise Manager to Create a Job	9-14
Services and the Scheduler: Example	9-15
Using Services with Metric Thresholds	9-16
Using Enterprise Manager to Change Service Thresholds	9-17
Services and Metric Thresholds: Example	9-18
Service Aggregation and Tracing	9-19
Top Services Performance Page	9-20
Service Aggregation Configuration	9-21
Service Aggregation: Example	9-22
Client Identifier Aggregation and Tracing	9-23

trcse ss Utility 9-24
Service Performance Views 9-25
Quiz 9-27
Summary 9-28
Practice 9 Overview: Using Services 9-29

10 Identifying Problem SQL Statements

Objectives 10-2
SQL Statement Processing Phases 10-3
Understanding Parsing 10-4
SQL Cursor Storage 10-5
Cursor Usage and Parsing 10-6
SQL Statement Processing Phases: Bind 10-8
SQL Statement Processing Phases: Execute and Fetch 10-9
Processing a DML Statement 10-10
Commit Processing 10-12
Role of the Oracle Optimizer 10-13
Quiz 10-15
Identifying Bad SQL 10-16
TOP SQL Reports 10-17
SQL Monitoring 10-18
Monitored SQL Execution Details 10-19
Quiz 10-20
What Is an Execution Plan? 10-21
Methods for Viewing Execution Plans 10-22
Uses of Execution Plans 10-24
DBMS_XPLAN Package: Overview 10-25
EXPLAIN PLAN Command 10-27
EXPLAIN PLAN Command: Example 10-28
EXPLAIN PLAN Command: Output 10-29
Reading an Execution Plan 10-30
Using the V\$SQL_PLAN View 10-31
V\$SQL_PLAN Columns 10-32
Querying V\$SQL_PLAN 10-33
V\$SQL_PLAN_STATISTICS View 10-34
Querying the AWR 10-35
SQL*Plus AUTOTRACE 10-37
Using SQL*Plus AUTOTRACE 10-38
SQL*Plus AUTOTRACE: Statistics 10-39
SQL Trace Facility 10-40
How to Use the SQL Trace Facility 10-42

Initialization Parameters 10-43
Enabling SQL Trace 10-45
Disabling SQL Trace 10-46
Formatting Your Trace Files 10-47
TKPROF Command Options 10-48
Output of the TKPROF Command 10-50
TKPROF Output with No Index: Example 10-55
TKPROF Output with Index: Example 10-56
Generate an Optimizer Trace 10-57
Quiz 10-58
Summary 10-59
Practice Overview 10: Using Execution Plan Utilities 10-60

11 Influencing the Optimizer

Objectives 11-2
Functions of the Query Optimizer 11-3
Selectivity 11-5
Cardinality and Cost 11-6
Changing Optimizer Behavior 11-7
Optimizer Statistics 11-9
Extended Statistics 11-10
Optimizer Parameters 11-11
Controlling the Behavior of the Optimizer with Parameters 11-12
Enabling Query Optimizer Features 11-14
Adaptive Execution Plans 11-15
Dynamic Plans 11-16
Dynamic Plan: Adaptive Process 11-17
Dynamic Plans: Example 11-18
Reoptimization: Cardinality Feedback 11-19
Cardinality Feedback: Monitoring Query Executions 11-20
Cardinality Feedback: Reparsing Statements 11-21
Automatic Re-optimization 11-22
Quiz 11-24
Influencing the Optimizer Approach 11-25
Optimizing SQL Statements 11-26
Quiz 11-27
Access Paths 11-28
Choosing an Access Path 11-29
Full Table Scans 11-30
Row ID Scans 11-32
Index Operations 11-33

B*Tree Index Operations 11-34
Bitmap Indexes 11-35
Bitmap Index Access 11-36
Combining Bitmaps 11-37
Bitmap Operations 11-38
Join Operations 11-39
Join Methods 11-40
Nested Loop Joins 11-41
Hash Joins 11-43
Sort-Merge Joins 11-44
Join Performance 11-46
How the Query Optimizer Chooses Execution Plans for Joins 11-47
Sort Operations 11-49
Tuning Sort Performance 11-50
Quiz 11-51
Summary 11-52
Practice 11 Overview: Influencing the Optimizer 11-53

12 Reducing the Cost of SQL Operations

Objectives 12-2
Reducing the Cost of SQL Operations 12-3
Index Maintenance 12-4
Dropping Indexes 12-6
Creating Indexes 12-7
Other Index Options 12-8
SQL Access Advisor 12-9
Quiz 12-10
Table Maintenance for Performance 12-11
Table Reorganization Methods 12-12
Space Management 12-14
Extent Management 12-15
Locally Managed Extents 12-16
Large Extents: Considerations 12-17
How Table Data Is Stored 12-19
Anatomy of a Database Block 12-20
Minimize Block Visits 12-21
Block Allocation 12-22
Free Lists 12-23
Block Space Management 12-24
Block Space Management with Free Lists 12-25
Automatic Segment Space Management 12-27

Automatic Segment Space Management at Work	12-28
Block Space Management with ASSM	12-30
Creating an Automatic Segment Space Management Segment	12-31
Quiz 12-32	
Migration and Chaining	12-33
Guidelines for PCTFREE and PCTUSED	12-35
Detecting Migration and Chaining	12-36
Selecting Migrated Rows	12-37
Eliminating Migrated Rows	12-38
Shrinking Segments: Overview	12-40
Shrinking Segments: Considerations	12-41
Shrinking Segments by Using SQL	12-42
Segment Shrink: Basic Execution	12-43
Segment Shrink: Execution Considerations	12-44
Using Enterprise Manager to Shrink Segments	12-45
Data Compression	12-46
Advanced Row Compression: Overview	12-48
Advanced Row Compression: Concepts	12-49
Using Advanced Row Compression	12-50
Oracle Hybrid Columnar Compression	12-51
How Does Columnar Compression Work?	12-52
Using the Compression Advisor	12-53
Viewing Table Compression Information	12-54
Quiz 12-55	
Summary	12-56
Practice 12 Overview: Reducing the Cost of SQL Operations	12-57

13 Using the SQL Performance Analyzer

Objectives	13-2
Real Application Testing: Overview	13-3
Real Application Testing: Use Cases	13-5
SQL Performance Analyzer: Process	13-6
Capturing the SQL Workload	13-8
Creating a SQL Performance Analyzer Task	13-9
SQL Performance Analyzer: Tasks	13-10
SQL Performance Analyzer Task Page	13-11
Comparison Report	13-12
Tuning Regressing Statements	13-13
SQL Tuning Recommendations	13-15
Preventing Regressions	13-16
SQL Performance Analyzer: PL/SQL Example	13-17

- Tuning Regressed SQL Statements 13-19
- SQL Performance Analyzer: Data Dictionary Views 13-20
- Quiz 13-21
- Summary 13-22
- Practice 13: Using SQL Performance Analyzer 13-23

14 SQL Performance Management

- Objectives 14-2
- Maintaining SQL Performance 14-3
- Maintaining Optimizer Statistics 14-4
- Automated Maintenance Tasks 14-5
- Statistic Gathering Options 14-6
- Setting Statistic Preferences 14-7
- Restoring Statistics 14-9
- Deferred Statistics Publishing: Overview 14-10
- Deferred Statistics Publishing: Example 14-12
- Automatic SQL Tuning: Overview 14-13
- SQL Statement Profiling 14-14
- Plan Tuning Flow and SQL Profile Creation 14-15
- SQL Tuning Loop 14-16
- Using SQL Profiles 14-17
- SQL Tuning Advisor: Overview 14-18
- Using the SQL Tuning Advisor 14-19
- SQL Tuning Advisor Options 14-20
- SQL Tuning Advisor Recommendations 14-21
- Alternative Execution Plans 14-22
- Quiz 14-24
- Using the SQL Access Advisor 14-25
- View Recommendations 14-27
- View Recommendation Details 14-28
- SQL Plan Management: Overview 14-29
- SQL Plan Baseline: Architecture 14-31
- Loading SQL Plan Baselines 14-33
- Evolving SQL Plan Baselines 14-34
- Adaptive SQL Plan Management 14-35
- Automatically Evolving SQL Plan Baseline 14-36
- Important Baseline SQL Plan Attributes 14-37
- SQL Plan Selection 14-39
- Possible SQL Plan Manageability Scenarios 14-41
- SQL Performance Analyzer and SQL Plan Baseline Scenario 14-42
- Loading a SQL Plan Baseline Automatically 14-43

Purging SQL Management Base Policy 14-44
Enterprise Manager and SQL Plan Baselines 14-45
Quiz 14-46
Summary 14-47
Practice 14: SQL Performance Management 14-48

15 Using Database Replay

Objectives 15-2
Using Database Replay 15-3
The Big Picture 15-4
System Architecture: Capture 15-5
System Architecture: Processing the Workload 15-7
System Architecture: Replay 15-8
Database Replay Workflow in Enterprise Manager 15-9
Capture Considerations 15-10
Capturing Workload with Enterprise Manager 15-12
Viewing Capture Progress 15-13
Capture Reports Tab 15-14
Replay Considerations: Preparation 15-15
Create Replay Task with Enterprise Manager 15-16
Create a Replay 15-17
Replay Tasks 15-18
Replay Considerations 15-19
Preprocess Workload 15-20
Running a Replay 15-21
Replay Customized Options 15-22
Replay: Customize Options Replay Parameters 15-23
Viewing Workload Replay Progress 15-24
Replay Analysis 15-25
Viewing Workload Replay Reports 15-27
Quiz 15-28
Database Replay Packages 15-29
Data Dictionary Views: Database Replay 15-30
Database Replay: PL/SQL Example 15-31
Calibrating Replay Clients 15-33
Quiz 15-34
Summary 15-35
Practice 15: Using Database Replay 15-36

16 Tuning the Shared Pool

- Objectives 16-2
- Shared Pool Architecture 16-3
- Shared Pool Operation 16-4
- Library Cache 16-5
- Latch and Mutex 16-7
- Latch and Mutex: Views and Statistics 16-9
- Diagnostic Tools for Tuning the Shared Pool 16-11
- AWR/Statspack Indicators 16-13
- Top Timed Events 16-14
- Time Model 16-15
- Load Profile 16-17
- Instance Efficiencies 16-18
- Library Cache Activity 16-19
- Avoid Hard Parses 16-20
- Are Cursors Being Shared? 16-21
- Candidate Cursors for Sharing 16-22
- Sharing Cursors 16-23
- Adaptive Cursor Sharing: Example 16-25
- Adaptive Cursor Sharing Views 16-27
- Interacting with Adaptive Cursor Sharing 16-28
- Reduce the Cost of Soft Parses 16-29
- Quiz 16-30
- Sizing the Shared Pool 16-31
- Shared Pool Advisory 16-32
- Shared Pool Advisory in an AWR Report 16-34
- Shared Pool Advisor 16-35
- Avoiding Fragmentation 16-36
- Large Memory Requirements 16-37
- Tuning the Shared Pool Reserved Pool 16-39
- Keeping Large Objects 16-41
- Data Dictionary Cache 16-43
- Dictionary Cache Misses 16-44
- SQL Query Result Cache: Overview 16-45
- Managing the SQL Query Result Cache 16-46
- Using the RESULT_CACHE Hint 16-48
- Using Table Annotation to Control Result Caching 16-49
- Using the DBMS_RESULT_CACHE Package 16-50
- Viewing SQL Result Cache Dictionary Information 16-51
- SQL Query Result Cache: Considerations 16-52
- Quiz 16-53

Summary 16-54
Practice 16: Tuning the Shared Pool 16-55

17 Tuning the Buffer Cache

Objectives 17-2
Oracle Database Architecture 17-3
Buffer Cache: Highlights 17-4
Database Buffers 17-5
Buffer Hash Table for Lookups 17-6
Working Sets 17-7
Tuning Goals and Techniques 17-9
Symptoms of a Buffer Cache Issue 17-11
Cache Buffer Chains Latch Contention 17-12
Finding Hot Segments 17-13
Buffer Busy Waits 17-14
Buffer Cache Hit Ratio 17-15
Buffer Cache Hit Ratio Is Not Everything 17-16
Interpreting Buffer Cache Hit Ratio 17-17
Read Waits 17-19
Free Buffer Waits 17-21
Solutions for Buffer Cache Issues 17-22
Sizing the Buffer Cache 17-23
Buffer Cache Size Parameters 17-24
Quiz 17-25
Dynamic Buffer Cache Advisory Parameter 17-26
Buffer Cache Advisory View 17-27
Using the V\$DB_CACHE_ADVICE View 17-28
Using the Buffer Cache Advisor 17-29
Caching Tables 17-30
Multiple Buffer Pools 17-31
Enabling Multiple Buffer Pools 17-33
Calculating the Hit Ratio for Multiple Pools 17-34
Multiple Block Sizes 17-36
Multiple Database Writers 17-37
Multiple I/O Slaves 17-38
Using Multiple Writers and I/O Slaves 17-39
Private Pool for I/O-Intensive Operations 17-40
Automatically Tuned Multiblock Reads 17-41
Database Smart Flash Cache Overview 17-42
Using Database Smart Flash Cache 17-43
Database Smart Flash Cache Architecture Overview 17-44

- Configuring Database Smart Flash Cache 17-45
- Sizing Database Smart Flash Cache 17-46
- Enabling and Disabling Flash Devices 17-47
- Specifying Database Smart Flash Cache for a Table 17-48
- Flushing the Buffer Cache (for Testing Only) 17-49
- Quiz 17-50
- Summary 17-51
- Practice 17: Tuning the Buffer Cache 17-52

18 Tuning PGA and Temporary Space

- Objectives 18-2
- SQL Memory Usage 18-3
- Performance Impact 18-4
- Automatic PGA Memory 18-5
- SQL Memory Manager 18-6
- Configuring Automatic PGA Memory 18-8
- Setting PGA_AGGREGATE_TARGET Initially 18-9
- Limiting the Size of the Program Global Area 18-10
- Monitoring SQL Memory Usage 18-11
- Monitoring SQL Memory Usage: Examples 18-13
- Tuning SQL Memory Usage 18-14
- PGA Target Advice Statistics 18-15
- PGA Target Advice Histograms 18-16
- Automatic PGA and Enterprise Manager 18-17
- Automatic PGA and AWR Reports 18-18
- Temporary Tablespace Management: Overview 18-19
- Temporary Tablespace: Locally Managed 18-20
- Configuring Temporary Tablespace 18-21
- Temporary Tablespace Group: Overview 18-23
- Temporary Tablespace Group: Benefits 18-24
- Creating Temporary Tablespace Groups 18-25
- Maintaining Temporary Tablespace Groups 18-26
- Viewing Tablespace Groups 18-27
- Monitoring Temporary Tablespace 18-28
- Shrinking a Temporary Tablespace 18-29
- Using the Tablespace Option when Creating a Temporary Table 18-30
- Quiz 18-31
- Summary 18-32
- Practice 18: Tuning PGA Memory 18-33

19 Using Automatic Memory Management

- Objectives 19-2
- Oracle Database Architecture 19-3
- Dynamic SGA 19-4
- Granules 19-5
- Memory Advisories 19-6
- Manually Adding Granules to Components 19-7
- Increasing the Size of an SGA Component 19-8
- Automatic Shared Memory Management: Overview 19-9
- SGA Sizing Parameters: Overview 19-10
- Dynamic SGA Transfer Modes 19-11
- Memory Broker Architecture 19-12
- Manually Resizing Dynamic SGA Parameters 19-13
- Behavior of Auto-Tuned SGA Parameters 19-14
- Behavior of Manually Tuned SGA Parameters 19-15
- Using the V\$PARAMETER View 19-16
- Resizing SGA_TARGET 19-17
- Disabling Automatic Shared Memory Management 19-18
- Enabling ASMM 19-19
- Using the SGA Advisor 19-20
- Monitoring ASMM 19-21
- Automatic Memory Management: Overview 19-22
- Oracle Database Memory Parameters 19-24
- Enabling Automatic Memory Management 19-25
- Monitoring Automatic Memory Management 19-26
- DBCA and Automatic Memory Management 19-28
- Quiz 19-29
- Summary 19-30
- Practice 19: Automatic Memory Tuning 19-31

20 Performance Tuning Summary

- Objectives 20-2
- Methodology 20-3
- Top 10 Mistakes Found in Customer Systems 20-4
- Common Symptoms 20-6
- Errors 20-7
- Diagnosing Errors 20-8
- Common Errors and Actions 20-9
- Initialization Parameters with Performance Impact 20-11
- Wait Events 20-14
- Diagnosing CPU waits 20-15

Tuning CPU Waits	20-16
Redo Path Wait Events	20-18
Redo Generation	20-21
Redo Write	20-22
Oracle Database 12c Redo Write	20-23
Automatic Checkpoint Tuning	20-24
Sizing the Redo Log Buffer	20-26
Sizing Redo Log Files	20-27
Increasing the Performance of Archiving	20-28
Buffer Cache Waits	20-30
Buffer Busy Waits	20-32
Shared Pool Waits	20-34
UGA and Oracle Shared Server	20-35
Large Pool	20-36
Tuning the Large Pool	20-37
Space Management Waits	20-38
General Tablespace: Best Practices	20-39
Undo Tablespace: Best Practices	20-40
SQL Execution Related Waits	20-41
I/O Waits for SQL	20-43
Internal Fragmentation Considerations	20-45
I/O Modes	20-46
Direct I/O	20-47
Direct Path Read and Write Waits	20-48
Implementing SQL Tuning Recommendations	20-50
Automatic Statistics Gathering	20-51
Automatic Statistics Collection: Considerations	20-52
Quiz	20-53
Summary	20-54

21 Oracle Database Cloud Service: Overview

Objectives	21-2
Database as a Service Architecture	21-3
Database as a Service: Features and Tooling	21-4
Software Editions: Included Database Options and Management Packs	21-5
Accessing the Oracle Database Cloud Service Console	21-6
Automated Database Provisioning	21-7
Accessing Tools from the Service Console	21-8
Managing the Compute Node Associated With a Database Deployment	21-9
Managing Network Access to Database as a Service	21-10
Enabling Access to a Compute Node Port	21-11

Scaling a Database Deployment	21-12
Performance Management in the Database Cloud Environment	21-13
Performance Monitoring and Tuning	21-14
What Can be Tuned in a DBCS Environment?	21-15
Summary	21-17

A Using Statspack

Objectives	A-2
Introduction to Statspack	A-3
Statspack Scripts	A-4
Installing Statspack	A-6
Capturing Statspack Snapshots	A-7
Configuring Snapshot Data Capture	A-8
Statspack Snapshot Levels	A-9
Statspack Baselines and Purging	A-11
Reporting with Statspack	A-13
Statspack Considerations	A-14
Statspack and AWR Reports	A-16
Reading an Statspack or AWR Report	A-17
Statspack/AWR Report Drilldown Sections	A-18
Report Drilldown Examples	A-20
Load Profile Section	A-21
Time Model Section	A-22
Statspack and AWR	A-23
Summary	A-24
Practice Overview: Use Statspack	A-25

Unauthorized reproduction or distribution prohibited. Copyright© 2018, Oracle and/or its affiliates.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

12

Reducing the Cost of SQL Operations

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Adjust data structures to influence the optimizer
- Tune segment space management
- Use the Segment Advisor
- Convert to Automatic Segment Space Management
- Tune block space management
- Diagnose and correct row migration
- Diagnose table fragmentation
- Use table compression

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Reducing the Cost of SQL Operations

Reduce the cost of SQL operations by managing the data structures.

- Maintaining indexes:
 - Rebuilding
 - Coalescing
 - Creating and dropping
- Maintaining tables:
 - Controlling space management (migrated rows)
 - Controlling empty blocks below the high-water mark
 - Reorganizing tables



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The DBA has a definite role in reducing the cost and increasing the performance of SQL statements. The DBA is primarily responsible for maintaining indexes and tables to keep SQL commands performing at optimum levels.

The DBA has tools, such as SQL Access Advisor, SQL Tuning Advisor, Automatic Maintenance Tasks, and various manual tools, that simplify these tasks.

Index Maintenance

DML activity can cause B*Tree indexes to develop sparsely populated blocks. This can:

- Reduce the number of entries per block
- Increase the number of I/Os required to do range scans

Maintain the indexes by:

- Rebuilding the index
- Coalescing the index



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When performing an index unique access, the server process reads the index root block, then reads a branch block at each level down the index tree to the leaf block, and then does a table access. The number of reads from root to leaf is equal to the number of levels in the index. Fewer levels mean fewer reads. Each branch and root block can hold only so many pointers to the next level down, depending on the size of the key. Generally this is a large number and the number of levels in an index seldom exceeds three.

In OLTP environments, where there is DML activity, some indexes can develop areas that have few entries per block. This is not a common condition. Random inserts tend to use the space that is released by deletes, and group deletes tend to make empty index blocks that are removed from the set of leaf blocks.

An *index rebuild* operation creates a new index from the existing index or the table. This operation can reduce the number of blocks needed for the index, and possibly the number of levels in the index. Reducing the size of the index means that there are more entries per block and fewer blocks, so more of the index can exist in the buffer cache at any given time. This can reduce the number of logical and physical I/Os. The rebuild operation can be performed online or offline. Rebuilding the index can be faster than coalescing when there is a random distribution of sparsely populated blocks. Rebuilding is not recommended if the pattern of inserts is likely to fill the sparse blocks in the near future.

An *index coalesce* operation combines adjacent leaf blocks when two or more adjacent blocks can fit in one block. This operation has many of the same benefits as index rebuild, with lower overhead. A coalesce operation will not reduce the number of levels in an index. The coalesce operation is always performed online. The coalesce operation works well when the sparsely populated blocks are relatively close together, as in a delete of a range of data that leaves some rows. If the sparse blocks are very sparse, the coalesce operation may move the same row several times (combining rows from several blocks before the block is full), increasing the overhead.

B*Tree indexes are used extensively in OLTP environments. If the number of deleted leaf rows exceeds 20% of the size of the leaf rows, a rebuild or coalesce of the index may improve performance. The following query retrieves the deleted leaf row ratio for a single index.

```
ANALYZE INDEX index_name VALIDATE STRUCTURE;  
SELECT name, del_lf_rows/lf_rows*100 reclaimable_space_pct FROM  
index_stats;
```

Note: INDEX_STATS holds only one row, the statistics from the last index analyzed.

Bitmap Indexes

Bitmap indexes are not recommended for OLTP environments. There are concurrency issues when bitmap indexes are built on tables incurring DML. Bitmap indexes are recommended for read-only applications, and read mostly application such as OLAP.

Dropping Indexes

Monitor index usage:

```
ALTER INDEX <index_name> MONITORING USAGE;  
SELECT index_name, used, monitoring  
FROM V$OBJECT_USAGE  
WHERE index_name = '<index_name>'
```

Test the impact with an invisible index:

```
ALTER INDEX <index_name> INVISIBLE
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The optimizer avoids using nonselective indexes within query execution, but all indexes defined against a table must be maintained. Index maintenance can present a significant CPU and I/O resource demand in any write-intensive application. In other words, do not build indexes unless necessary.

For best performance, drop indexes that an application is not using. You can find indexes that are not being used by using the ALTER INDEX MONITORING USAGE functionality over a period of time that is representative of your workload. This records whether or not an index has been used in the V\$OBJECT_USAGE view. If you find that an index has not been used, then drop it. Make sure that you are monitoring a representative workload to avoid dropping an index that is used by a workload that you did not sample.

Also, indexes within an application sometimes have uses that are not immediately apparent from a survey of statement execution plans. An example of this is a foreign key index on a parent table, which prevents share locks from being taken out on a child table.

You can test the impact of dropping an index by setting it to INVISIBLE. An invisible index still exists and is maintained, but is not used by the optimizer. If the index is needed, use the ALTER INDEX ... VISIBLE command.

Be sure to check the use of indexes for primary key and foreign key constraint enforcement before dropping them.

Creating Indexes

Use indexes to improve SQL performance.

- Use B*Tree indexes:
 - For join columns
 - For columns frequently appearing in WHERE clauses
- Use composite indexes:
 - For columns frequently used together in WHERE clauses
 - Consider using compressed key indexes for large indexes.
- Use bitmap indexes:
 - For columns with few distinct values and large number of rows
 - For read-only or read-mostly tables
 - For WHERE clause with multiple predicates



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Creating additional indexes can improve performance. To improve the performance of individual SQL statements consider using the following:

- **B*Tree indexes** on the columns most used in the WHERE clause. Indexing a column frequently used in a join column can improve join performance. Do not use B*Tree indexes on columns with only a few distinct values, bitmap indexes are a better choice for these.
- **Composite B*Tree indexes** for columns that frequently appear together in the same WHERE clause. Composite indexes of several low selectivity columns may have a higher selectivity. Place the most frequently used column first in the index order. For large composite indexes, consider using a compressed index. Index key compression works well on indexes with leading columns that have a few distinct values—the leading, repeating values are eliminated. This reduces the size of the index and number of I/Os.
- **Bitmap indexes** can help performance where the tables have a large number of rows, the columns have a few distinct values, and the WHERE clause contains multiple predicates using these columns. Do not use bitmap indexes on tables or columns that are frequently updated.

Other Index Options

Less frequently used are:

- Reverse key indexes
 - Reduces block contention for concurrent DML
 - Cannot be used for Index range scans for queries
- Compressed key indexes
 - Can reduce storage when the key or keys have repeating values
 - Does not hurt performance on DML or query

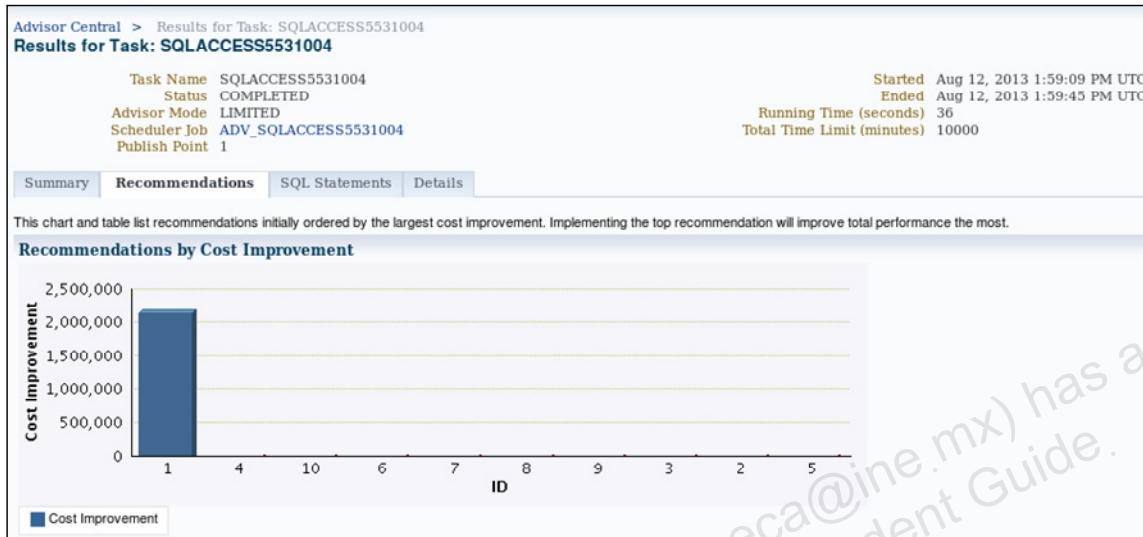


Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Less frequently used options:

- **Reverse key indexes:** Reverse the key value, for example, 1709 becomes 9071 in the index. This works very well to reduce buffer contention, as in buffer busy waits and enqueue index waits in situations where a number of processes are attempting to insert into the same block. This can happen in RAC databases and busy OLTP databases when the key is a sequence number or some other monotonically increasing value. In a normal index, these new key values will all be inserted in the same (last) leaf block of the index. With a reverse key index, the values will be scattered throughout the leaf blocks. Reverse key indexes cannot be used for an index range scan, because the normally sequential values are scattered throughout the index.
- **Compressed key indexes:** Replace the key value with a pointer to the actual key value in the block. The pointer takes very little space. So when the key value is repeated frequently in the index block there is a space savings. There is no performance penalty because all the information is in the index block, so there are no additional I/Os required. Compressed key indexes work well for large indexes that have repeating key values. The more often the key value repeats the more space is saved. Compressed indexes offer no benefit to unique key indexes.

SQL Access Advisor



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The problem with adding indexes is knowing the overall impact. A new index may improve the performance of one or a few SQL statements, but hurt the performance of a large number of the DML statements that must update the index for INSERT, DELETE, and UPDATE of the column.

The SQL Access Advisor makes a set of recommendations that can include adding or dropping indexes. SQL Access Advisor can be invoked from Advisor Central in Oracle Enterprise Manager Cloud Control or through the DBMS_ADVISOR package APIs. The Access Advisor uses a workload you specify, or it generates a hypothetical workload for a specified schema. Various workload sources are available, such as the current contents of the SQL cache, a user defined set of SQL statements, or a SQL tuning set. Given a workload, the Access Advisor generates a set of recommendations from which you can select the indexes that are to be implemented. An implementation script is provided that can be executed manually or automatically through Oracle Enterprise Manager Cloud Control.

Quiz

The major cost in retrieving data from any database is disk I/O. A disk access takes 1,000 to 100,000 times more time than a memory access. An access path that uses an index is always faster than a full table scan.

- a. True
- b. False

 ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: b

A full table scan (FTS) generally requires physical reads, but those reads can be multiblock reads. A multiblock read costs more than a single block read, but much less than reading the same number of blocks with single block reads. But if the table is already in memory, there may be few or no physical reads generated.

An index access always requires a minimum of two accesses. First to get the index block, and second to access the table row. If the table is small or cached, a full table scan could be faster. With larger indexes and tables, there are more gets required. If the query requires reading a significant portion of the table, an FTS could easily be faster than index lookups.

Table Maintenance for Performance

- Table space usage can impact performance.
 - Empty/partial blocks below the high-water mark affect full table scans.
 - Migrated rows affect index lookups.
- Table organization can affect performance.
 - Index organized
 - Hash cluster
 - Partitioned



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Table space usage can effect performance. A full table scan reads all the formatted blocks below the high-water mark empty blocks as well. The Segment Advisor (the Automatic Segment Advisor task) can help by identifying segments that have empty space below the high-water mark. The segments identified are candidates for reorganization, or shrinking.

A row in a table can be migrated when it is updated and there is not enough space in the current block to hold the entire row. When a row is migrated, the row is placed in another block and a pointer to the new location is placed in the original block. The index row ID is not updated. Index lookups use the row ID in the index to read the block that holds the row. When the block is retrieved, the pointer is found and the new block is read. When 10–15% of the rows in a table are migrated and the table primarily is used by index access, you can see a performance degradation due to extra I/Os.

The default heap organization of a table is a good fit for most applications. Other data structures are available: Index-organized tables hold the table rows in a B*Tree structure. Partitioned tables break the table into separate segments organized by a partition key value. Clustered tables are not used much in applications, but the hash cluster is good for small tables where the key value is frequently queried, and seldom or never changed. The hash cluster allows a direct lookup without an index. For more information about using these and other data structures for performance, see the *Oracle Database Performance Tuning Guide*.

Table Reorganization Methods

- Online reorganization (`DBMS_REDEFINITION`)
 - Online; data is available
 - Could use considerable resources
- Shrink segments
 - Online; data is available
 - In place, no additional space required
- Export, re-create, and import
 - Offline; data is not available
 - Time consuming for large tables
- CTAS, `ALTER TABLE ... MOVE`
 - Mostly offline; data is not available
 - Out of place, requires additional space



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You have decided to reorganize a table (whether to take advantage of a different data structure or to remove migrated rows). Now you must choose the method.

Online reorganization can reorganize a table, change its structure, add and remove columns, or change the table space, while the data is available for use. For large table, this operation could take a long time and it will require at least two times the space currently being used for the table and its indexes. For information about using the `DBMS_REDEFINITION` package, see the *Oracle Database Administrator's Guide*.

The `ALTER TABLE ... SHRINK` command operates on the table and cascades to indexes and other dependent objects when you specify `CASCADE`. The shrink command moves rows inside an existing heap-organized table to create empty blocks near the high-water mark. The command will move the high-water mark and by default returns those empty blocks to the tablespace for use by other segments. If `COMPACT` is specified, the table rows are moved but the high-water mark is not moved and the space is not released. For more information and restrictions, see *Oracle Database SQL Language Reference*.

Using export and import to do the reorganization is a traditional method that requires that the table be dropped; this means that the data will be unavailable for the duration of the operation. The time could be large for large tables. The export and import utilities are described in *Oracle Database Utilities*.

CREATE TABLE ... AS SELECT (CTAS) and ALTER TABLE ... MOVE commands allow reorganization to different levels. CTAS does an insert into a new table structure. The MOVE command copies the blocks to a new location. The MOVE command can be executed ONLINE for index-organized tables and index-organized nested tables.

Space Management

Space is managed at three levels:

- Files (OS, ASM) assign disk space to tablespaces.
- Extents are used to allocate file space to segments in a tablespace.
- Blocks are used to organize the space inside data objects.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Oracle Database server organizes physical space at three levels. Files are used to assign physical disk space to tablespaces with the rule that a file can be assigned to one and only one tablespace. A tablespace may be assigned many files.

Each file is divided into extents. Extents are allocated to segments as needed. Segment types include tables, indexes, LOBs, partitions, and undo segments. As segments grow, more extents are allocated until there are no more free extents in the files. Extents are created in contiguous sets of blocks. An extent may be as small as two blocks or as large as the file (under 2 GB). An extent is a subdivision of a file and cannot span file boundaries.

After an extent is allocated to a segment, it is not returned to the tablespace for reuse until the segment is dropped, except for certain operations. Segments place rows, chunks, or undo records into blocks, following certain rules set by parameters on the segment. Blocks can be unformatted, empty, or have some measure of data in them.

Space can be returned to the tablespace from a segment with the following commands:

```
ALTER TABLE ... SHRINK SPACE;  
TRUNCATE TABLE ... [DROP STORAGE];  
ALTER TABLE ... DEALLOCATE UNUSED;
```

Extent Management

Extents are allocated in two ways:

- Dictionary managed
 - Only supported for backward compatibility
 - DBMS_SPACE_ADMIN.
TABLESPACE_MIGRATE_TO_LOCAL is provided.
- Locally managed
 - Extents managed in the file header bitmap
 - No undo created on extent operations
 - Possible contention on file header blocks
 - No recursive SQL for space management



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Oracle Database server manages extents in two ways. Dictionary managed tablespaces are supported only for backward compatibility. The DBMS_SPACE_ADMIN package includes the TABLESPACE_MIGRATE_TO_LOCAL procedure to convert dictionary management to local management. Dictionary managed tablespaces cannot be created unless the SYSTEM tablespace is dictionary managed. The SYSTEM tablespace can be created as dictionary managed only through the CREATE DATABASE command; it is not supported in DBCA.

Local management of extents is accomplished through bitmaps located in a space management block in the file. Extent operations only switch bits and each bit covers one or more blocks in the file. This method has a much lower overhead than dictionary-managed extents. Local management of extents is the recommended method. Even local extent management can see contention in some cases where there is a high rate of extent operations. For example, in a scenario where there is a high rate of DML operations with many separate transactions, undo segments are constantly adding and releasing extents. Because the extent bitmap is in the file header, you may see buffer busy waits on file header blocks in the undo tablespace. The solution is to create more files, increasing the number of file header bitmaps that are being used.

Locally Managed Extents

- Create a locally managed tablespace:

```
SQL> CREATE TABLESPACE user_data_1
  2  DATAFILE
  3  '/u01/app/oracle/oradata/orcl/lm_1.dbf'
  4  SIZE 100M
  5  EXTENT MANAGEMENT LOCAL
  6  UNIFORM SIZE 2M;
```

- Default extent management is local.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A locally managed tablespace manages its own extents and maintains a bitmap in each data file to keep track of the free or used status of blocks in that data file. Each bit in the bitmap corresponds to a block or a group of blocks. When an extent is allocated or freed for reuse, the bitmap values change to show the new status of the blocks. These changes do not generate undo information because they do not update tables in the data dictionary.

The Database Creation Assistant (DBCA) creates the system tablespace as locally managed by default. The `CREATE DATABASE` command defaults to a dictionary-managed system tablespace.

When the system tablespace is locally managed, a dictionary-managed tablespace cannot be created in the database, but can be added as a read-only tablespace using the Transportable Tablespace feature. In such a case, the dictionary managed tablespace can never be made read-write.

Extent management, either dictionary or local, is set when the tablespace is created.

Large Extents: Considerations

- Advantages:
 - Are less likely to extend dynamically
 - Deliver a small performance benefit
 - Enable the server process to read the entire extent map with a single I/O operation
- Disadvantages:
 - Free space may not be available
 - May contain unused space



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To ease space management, you should create objects with appropriately sized extents. As a general rule, larger extents are preferred over smaller extents.

Advantages of Large Extents

- Large extents avoid dynamic extent allocation, because segments with larger extents are less likely to be extended. This benefit is reduced with locally managed extents.
- Larger extents sometimes provides a small performance benefit for full table scans (including index fast full scans) when the server process can read one large extent from disk with fewer reads than would be required to read many small extents. This situation can be avoided by matching extent sizes to the I/O and space allocation sizes. Then the performance cost of having many extents in a segment is minimized. As the extent size increases, the benefit is reduced. However, for a table that never has a full table scan operation, it makes no difference in terms of query performance whether the table has few extents or many extents. The performance of searches using an index is not affected by the size of the extents in the index.

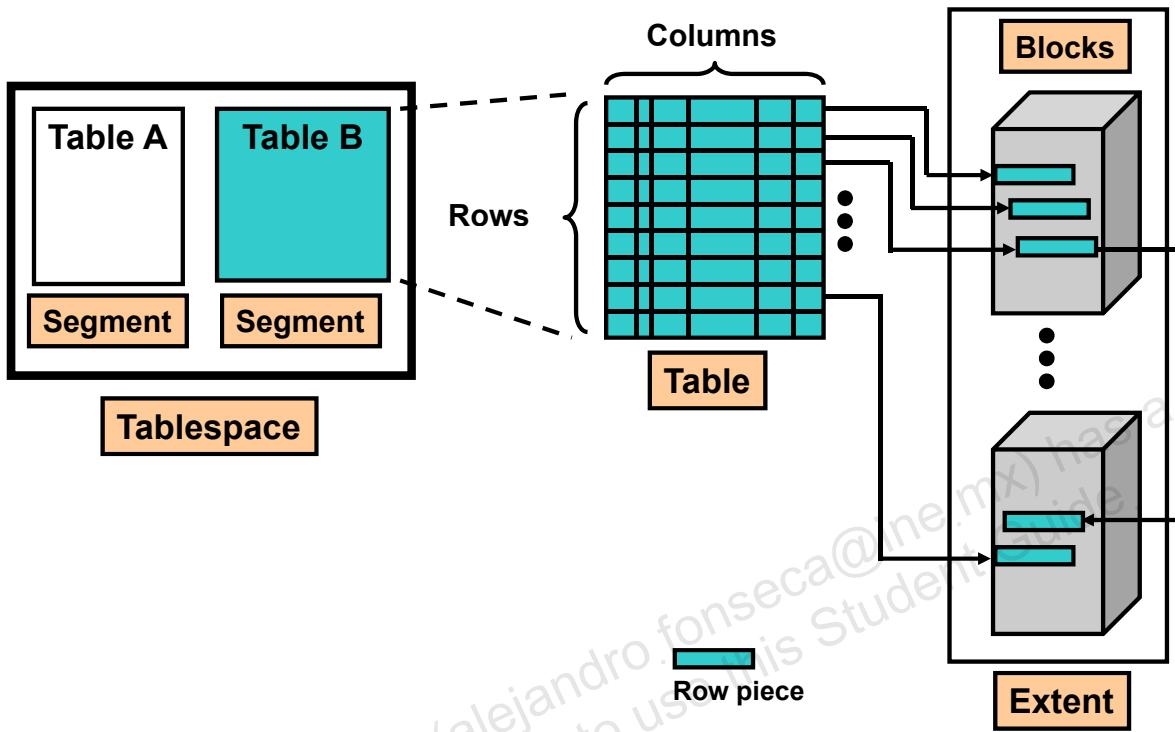
- Extent maps list all the extents for a certain segment. When MAXEXTENTS is set to UNLIMITED, additional extent map blocks are created as needed. For best performance, the extent map is in one block and is read with a single I/O. Performance degrades a small amount if multiple I/Os are necessary to get the extent map, as in the case of a full table scan. The number of additional I/Os required for the additional extent map blocks are negligible by the time the table reaches the size that requires the additional maps. With dictionary extent management, a large number of extents can degrade data dictionary performance, because each extent uses a row in either the UET\$ or FET\$ table in the dictionary.

Disadvantages of Large Extents

- Large extents require more contiguous blocks; therefore, the server process may have difficulty finding enough contiguous space to store them. The segment space advisor may be used to shrink tables that do not use all the space that has been allocated to them. The ALTER TABLESPACE COALESCE command is used with dictionary-managed extents to combine adjacent free extents. Locally managed extents avoid this issue by automatically combining adjacent free extents.
- The DBA sizes the segment to allow for growth, so some of the space allocated to the segment is not used initially, but is reserved for future use by that segment.

To determine whether to allocate a few large extents or many small extents, consider how the benefits and drawbacks of each would affect your plans for the growth and use of your tables.

How Table Data Is Stored



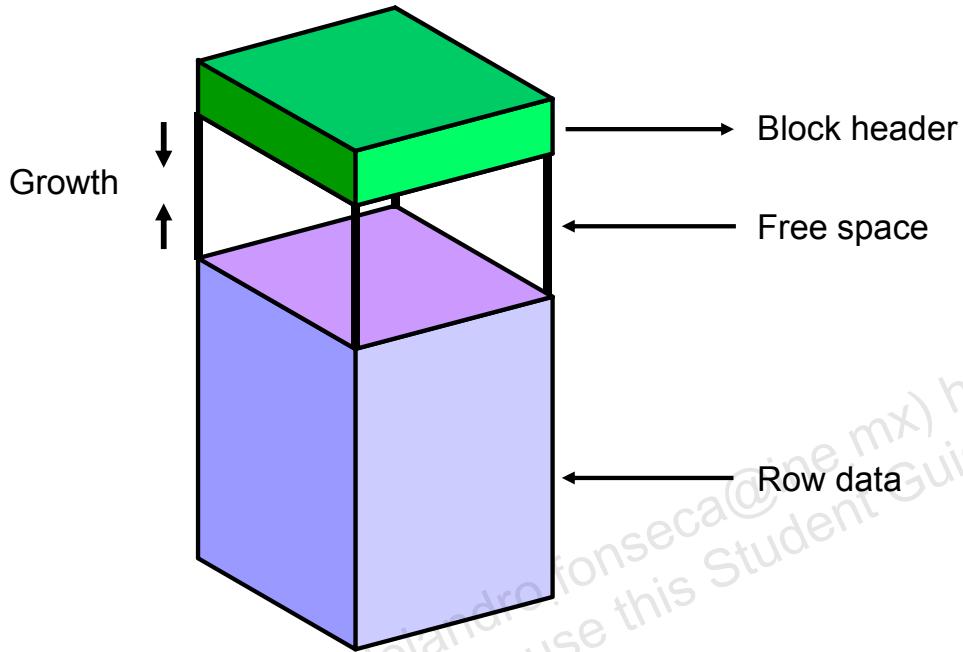
ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When a table is created, a segment is created to hold its data. A tablespace contains a collection of segments. Logically, a table contains rows of column values. A row is ultimately stored in a database block in the form of a row piece. It is called a “row piece” because under some circumstances, the entire row may not be stored in one place. This happens when an inserted row is too large to fit into a single block or when an update causes an existing row to outgrow its current space. Rows that are too large to fit into any block are called “chained rows,” because they are chained across multiple blocks. A row that is moved to a new block, because it grew too large for the original block, is called a “migrated row.”

When a row has more than 255 columns, it will be broken into row pieces. Ideally all the pieces will be in the same block, if not, there is a performance impact to fetch the other blocks. An index-organized table (IOT) can have some columns specified as overflow, or have the row divided into pieces due to the PCTTHRESHOLD storage parameter. In either case, the row is counted as chained. For more information about IOT storage, see *Oracle Database SQL Language Reference*. When a table has an associated LOB segment holding out-of-line LOBs, the LOB locator is included in the row and the row is not considered chained. But to access the LOB, at least one more I/O is required to access the LOB segment. For more details about performance and using LOBs, see *Oracle Database SecureFiles and Large Objects Developer's Guide*.

Anatomy of a Database Block



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

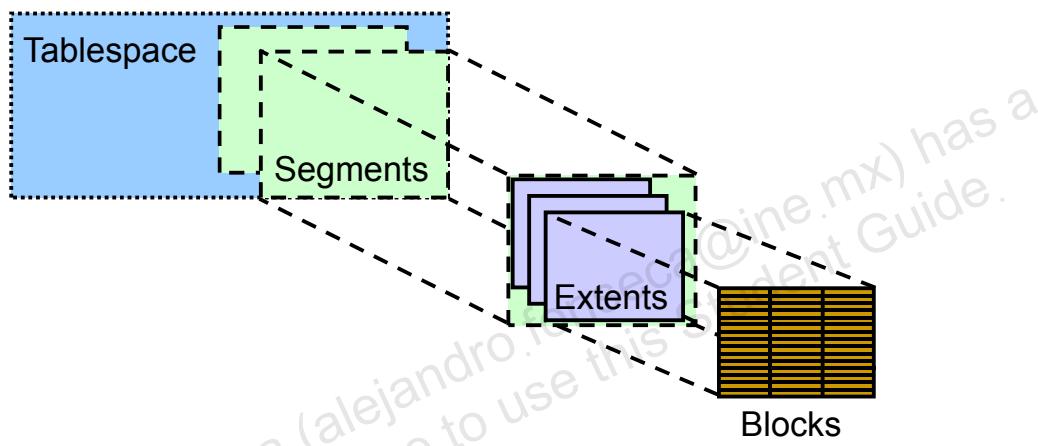
Database data blocks contain the following:

- **Block header:** The block header contains the segment type (such as table or index), data block address, table directory, row directory, and transaction slots that are used when modifications are made to rows in the block. The block header grows downward from the top.
- **Row data:** This is the actual data for the rows in the block. Row data space grows upward from the bottom.
- **Free space:** Free space is in the middle of the block. This enables the header and the row data space to grow when necessary. Row data takes up free space as new rows are inserted or columns of existing rows are updated with larger values. The examples of events that cause header growth are when the row directory needs more row entries or more transaction slots are required than initially configured. Initially, the free space in a block is contiguous. However, deletions and updates may fragment the free space in the block. The free space in the block is coalesced by the server processes when necessary.

Minimize Block Visits

Minimize block visits by:

- Using a larger block size
- Packing rows tightly
- Preventing row migration



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

One of the database tuning goals is to minimize the number of blocks visited. Each block visit does produce a logical read, and can result in a physical read. Estimated block visits is one of the major factors in the estimated cost generated by the optimizer. The developer contributes to this goal by tuning the application and SQL statements. The DBA reduces block visits by:

- Using a larger block size
- Packing rows as closely as possible into blocks
- Preventing row migration

Row migration occurs when rows are updated. If the updated row grows, and can no longer fit in the block, the row is moved to another block. A pointer is left in the original row location, and it points to the new location.

Unfortunately for the DBA, the last two goals conflict: As more data is packed into a block, the likelihood of migration increases.

Block Allocation

- When an INSERT or UPDATE operation requires more space, a block must be found with adequate space.
- Two methods for space allocation:
 - Manual segment space management
 - Uses freelists
 - Automatic Segment Space Management (ASSM)
 - Uses bitmap blocks



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When an insert or update operation requires more space, a block must be found with adequate space. There are two methods for assigning an operation to a block: free lists and Automatic Segment Space Management (ASSM). Both methods are available with locally managed tablespaces.

Free Lists

Free list–managed space characteristics:

- Segment header blocks hold all free lists.
- Blocks are added to and removed from the free lists.
- Free lists are searched for available blocks.
- Segment headers are pinned for the search and update of free lists.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Manual Segment Space Management uses a free list to control block allocation. A free list is a linked list of blocks that are available in the segment for inserts. A segment can be created with multiple free lists or free list groups. There is one free list created by default for a segment . When the free space in a block drops below the PCT_FREE threshold, the block is removed from the free list. If rows are deleted and the used space decreases past the PCTUSED threshold, the block is returned to the head of a free list. The header block containing the free list is pinned while any block in that free list is being changed. A process that requires a block for INSERT pins the header block while it searches the free list. When multiple processes are inserting into the same segment concurrently, there can be contention for the free list.

Free-list contention is shown by buffer busy waits on data header blocks. Free-list contention is common in high-insert environments, when multiple processes attempt to search the free list at the same time.

One solution to free list contention is to create multiple free lists, or free-list groups. Each free list group can hold multiple free lists and is created in an additional segment header block. Free list groups have been used primarily to control extent allocation in a RAC environment.

Block Space Management

Each segment has parameters that control the space usage inside a block. For a table:

- PCTFREE: Amount of space reserved for updates
- PCTUSED: A minimum level of free space in a block before a block is placed on the free list

For an index:

- PCTFREE: Amount of space reserved for new index entries at creation time
- PCTUSED: Always 0 for indexes



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Each segment has parameters that control the space usage inside a block. You use two space management parameters, PCTFREE and PCTUSED, to control the use of free space within all the data blocks of a table or an index. You specify these parameters when creating or altering a table. These parameters may be specified when you create or rebuild an index.

For Tables

PCTFREE sets the minimum percentage of a data block to be reserved as free space for possible updates to rows that already exist in that block. When the amount of free space in a block drops below PCTFREE, the block is no longer available for inserts.

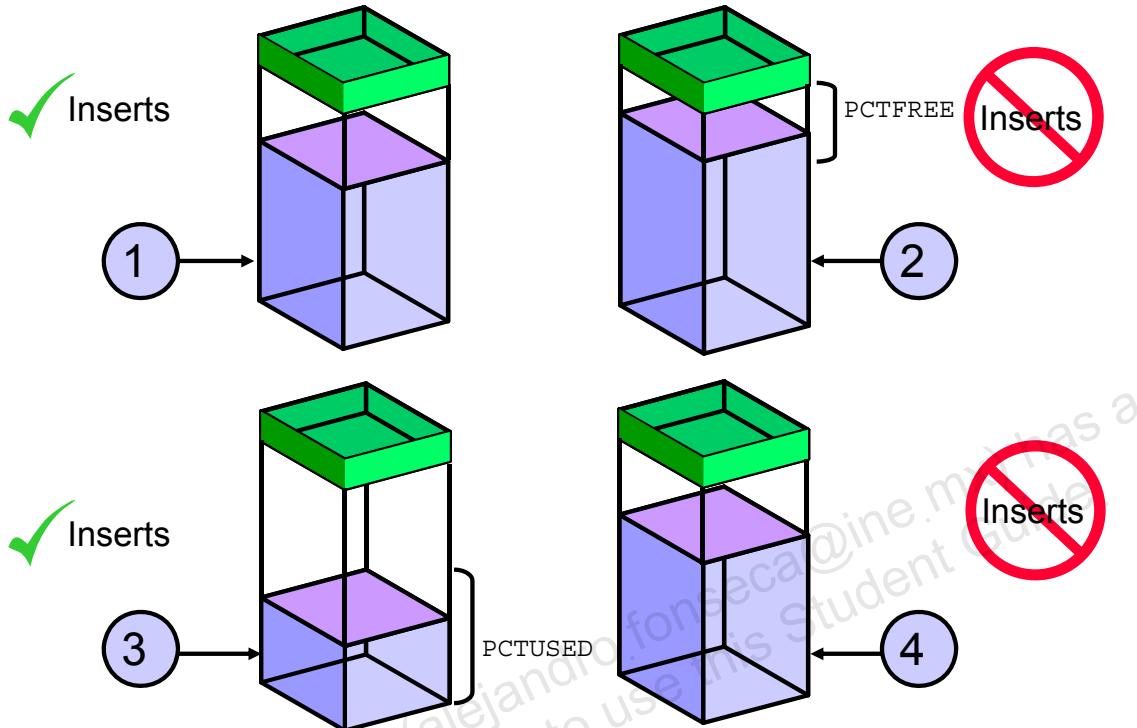
PCTUSED sets a lower limit on the percentage of a block that is used for row data plus overhead. When the percentage of space used drops below this limit, the block is available for inserts.

For Indexes

PCTFREE sets the amount of space reserved for new index leaf entries. It applies only during index creation if the table is not empty.

PCTUSED is always 0 for indexes.

Block Space Management with Free Lists



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

PCTFREE and PCTUSED

As an example, if you execute a `CREATE TABLE` statement with `PCTFREE` set to 20, 20% of each data block in the data segment of this table is reserved for updates to the existing rows in each block. The used space in the block can grow (1) until the row data and overhead total 80% of the total block size. Then the block is removed from the free list (2).

With free list-managed space, after you run a `DELETE` or an `UPDATE` statement, the server process checks whether the space being used in the block is now less than `PCTUSED`. If it is, the block is added to the beginning of the free list. When the transaction is committed, free space in the block becomes available for other transactions (3).

After a data block is filled to the `PCTFREE` limit again (4), the server process again considers the block unavailable for the insertion of new rows until the percentage of that block falls below the `PCTUSED` parameter.

DML Statements, PCTFREE, and PCTUSED

Two types of statements can increase the free space of one or more data blocks:

- `DELETE` statements
- `UPDATE` statements, which update existing values to values that use less space

How PCTFREE and PCTUSED Work Together

Released space in a block may not be contiguous; for example, when a row in the middle of the block is deleted. The server process coalesces the free space of a data block only when both of the following are true:

- An `INSERT` or an `UPDATE` statement attempts to use a block that contains enough free space to contain a new row piece
- The free space is fragmented so that the row piece cannot be inserted in a contiguous section of the block

The server process performs this compaction only when required, because otherwise the performance of a database system would decrease due to the continuous compaction of the free space in data blocks.

Note: If you change `PCTFREE` and `PCTUSED` for existing tables, there is no immediate impact on blocks. However, future DML activity uses the new rules for tables.

Setting PCTFREE for an Index

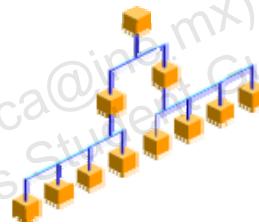
You can also specify the `PCTFREE` storage parameter when creating or altering an index.

Setting `PCTFREE` for an index specifies how much of a block to fill when the index is created. It does not keep space available for updates as is done with data blocks.

Automatic Segment Space Management

Automatic Segment Space Management (ASSM) characteristics:

- Space is managed with bitmap blocks (BMB).
- Multiple processes search different BMBs.
- Availability of a block is shown with a full bit.
- The fullness is shown by a percentage-full bit for each of 25, 50, 75, and 100 percent used.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

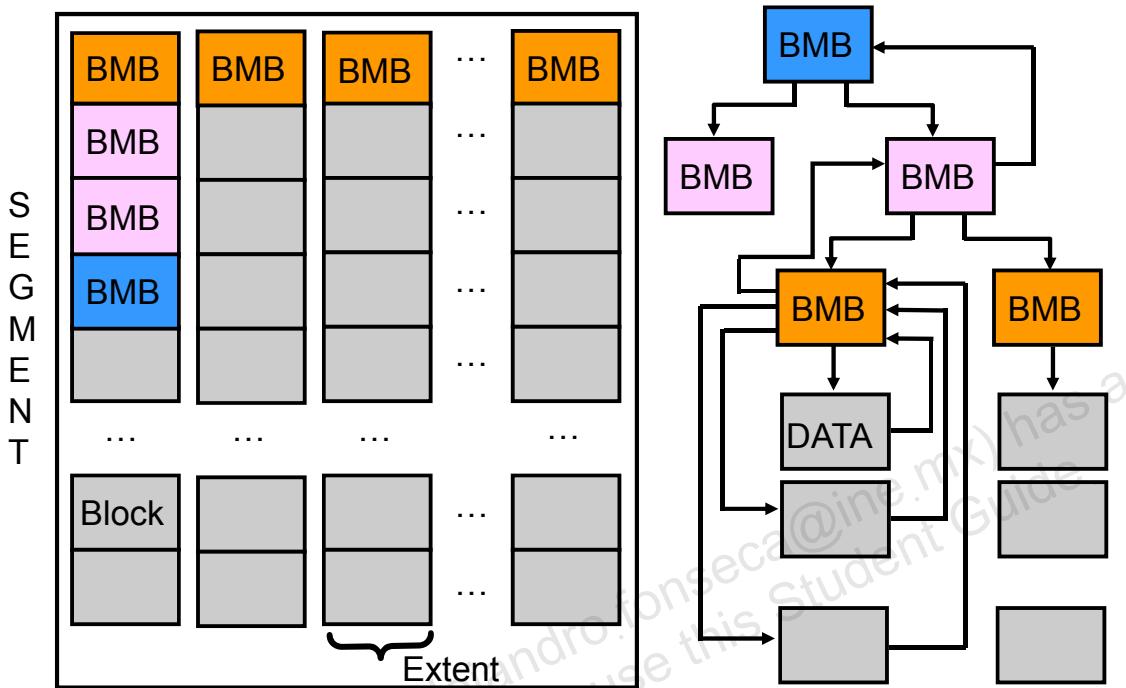
Automatic Segment Space Management uses a set of bitmap blocks (BMB) that are scattered throughout the segment to hold block fill-level information. A few bits for each block indicates the fill level and whether the block is available for inserts.

The BMBs are organized in a tree structure. When multiple processes are attempting to find a block for inserts, they access different BMBs based on a hash of the instance ID and process ID. If the selected block does not have available space, the process can traverse the tree.

With ASSM, the free list is eliminated and the contention for finding blocks with available space is greatly reduced or eliminated.

The ASSM structure is recommended. It is especially useful for RAC databases to avoid free-list contention and free-list group maintenance. In the RAC environment, multiple free list groups solve the internode contention but not the intersession contention. In the single instance environment, multiple free lists reduce, but does not completely solve the intersession contention. ASSM solves both problems.

Automatic Segment Space Management at Work



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Segments using Automatic Segment Space Management have a set of bitmap blocks (BMBs) describing the space utilization of the data blocks in that segment. For each data block, there is a set of bits per block indicating the space available in that block.

BMBs are organized in a tree hierarchy similar to a B*Tree index with the levels doubly linked. The maximum number of levels inside this hierarchy is three. The leaves of this hierarchy represent the space information for a set of contiguous data blocks that belong to the segment. The BMB leaves are the units at which space has affinity to an instance in a multi-instance environment.

When segments are growing with larger amounts of data being stored in the database, it becomes prohibitively expensive to search all the leaves of that hierarchy during an insert. Thus, an intermediate level of BMBs is created to contain metadata about the leaves.

During an `INSERT` operation, the server process starts from the root of the hierarchy. It goes down to the leaves to find a useful bitmap block pointing to data blocks containing free space. The BMBs are acquired in shared mode so that multiple processes can search in the same BMB.

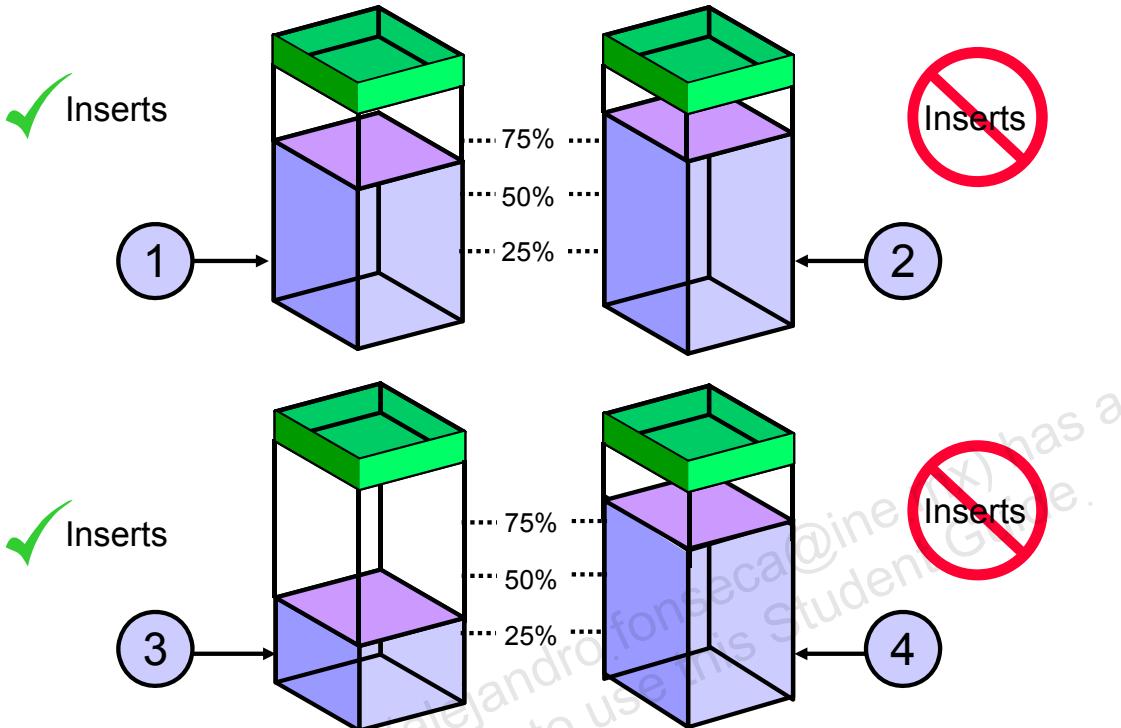
Within an intermediate BMB level, concurrent processes get distributed by hashing on the instance number and process ID. This operation obtains BMB leaves. The server process hashes the process ID to provide a starting point again inside this BMB leaf.

In this type of BMB hierarchy, the leaves point to a range of data blocks that are potential candidates for the `INSERT` operation.

Note: The exact number of BMBs at each BMB index level depends on the individual extent sizes as well as on the overall segment size.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

Block Space Management with ASSM



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

If you execute a `CREATE TABLE` statement with `PCTFREE` set to 20 in an ASSM tablespace, 20% of each data block in the data segment of this table is reserved for updates to the existing rows in each block. The used space in the block can grow (1) until the row data and overhead total 80% of the total block size. Then the block is marked as full, and the full bit is set (2). After the full bit is set, the block is not available for inserts. There are four other bits that indicate 100%, 75%, 50%, and 25% full. When the block reaches 80%, all the bits except the 100% bit have been set and then the full bit is set.

With ASSM-managed space, after the `DELETE` or `UPDATE` operation, the server process checks whether the space being used in the block is now less than a preset threshold. The threshold levels are 25, 50, 75, and 100 percent. If the used space is less than a threshold that is lower than `PCTFREE`, the block is marked as not full and available for inserts. In this example, the used space must fall below 75 percent to be marked as available for inserts (3). After a data block is filled to the `PCTFREE` limit again (4), the server process again considers the block unavailable for the insertion of new rows until the percentage of that block falls below the ASSM threshold.

Creating an Automatic Segment Space Management Segment

- SEGMENT SPACE MANAGEMENT is the attribute used for tablespace creation, which cannot be subsequently altered.
- Segment space management is declared at the tablespace level.
- Tablespace must be permanent and locally managed.
- Automatic space management segments are specified through the default AUTO keyword.
- For free-list segments, use the value MANUAL.
- PCTUSED, FREELIST, and FREELIST GROUPS are ignored at table creation for ASSM.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The SEGMENT SPACE MANAGEMENT specification applies only to a tablespace. You cannot specify the SEGMENT SPACE MANAGEMENT clause at the table level.

All segments residing in a tablespace with SEGMENT SPACE MANAGEMENT set to AUTO are automatic space management segments. Segments residing in a tablespace with SEGMENT SPACE MANAGEMENT set to MANUAL are managed by the PCTUSED, FREELIST, and FREELIST GROUPS attributes.

Note: Segments that can be defined as automatic space management segments are heap tables, indexes, index-organized tables (IOTs), and LOBs.

There is no Oracle-supplied procedure to migrate from manual segment space management to Automatic Segment Space Management. To alter a segment to use ASSM, the segment must be moved to a tablespace using ASSM.

Quiz

Reorganizing tables can serve several purposes, such as reduce storage space, decrease I/O, and improve performance. A large heap-organized table needs to be moved to a different tablespace to help manage backups, and reorganized to remove empty space; however, it must be available during the move due to 24x7 requirements. What steps or procedure could be used for the move?

- a. Use the ALTER TABLE ... MOVE command.
- b. Use online reorganization (using the DBMS_REDEFINITION package).
- c. Use CREATE TABLE ... AS SELECT, and then rename.
- d. Use export, drop table, and import.
- e. Use ALTER TABLE ... SHRINK.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: b

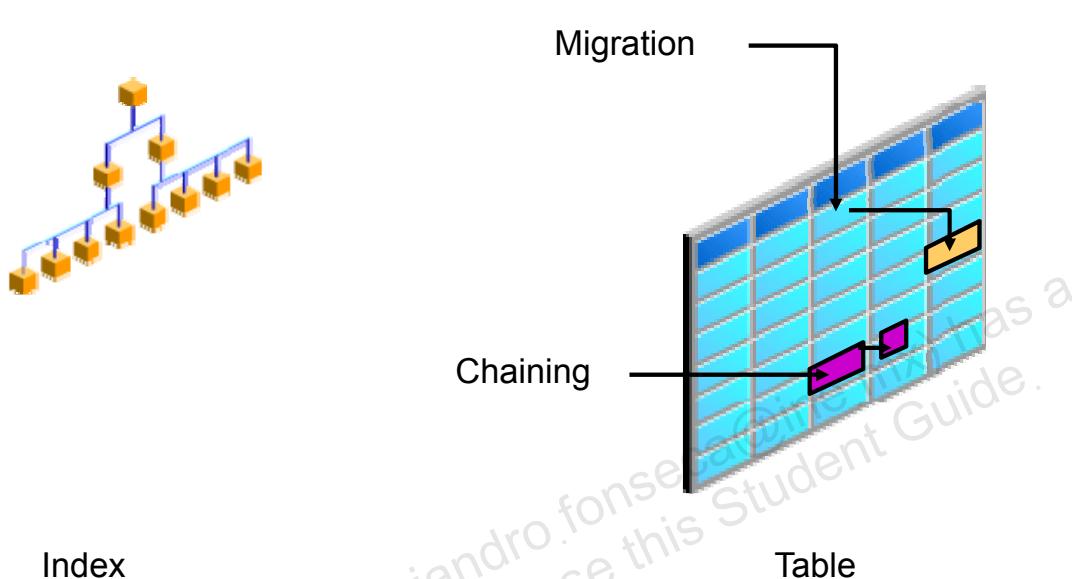
Because of the requirement to keep the data available, online redefinition is the only option.

MOVE TABLE is an online option only for index-organized tables.

Shrinking a table is an online, in-place operation. It does not move the table to a new location.

The other options are offline operations.

Migration and Chaining



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In two circumstances, accessing a row requires more than one block to be read. This is caused by:

- **Migration:** An UPDATE statement increases the amount of data in a row so that the row no longer fits in its data block. The server process tries to find another block with enough free space to hold the entire row. If such a block is available, the server process moves the entire row to the new block. The server process keeps the original row piece of a migrated row to point to the new block containing the actual row; the ROWID of a migrated row does not change. Indexes are not updated; they continue to point to the original row location. During an index access of the row, the original row piece is found, and then another read to retrieve the row is required.
- **Chaining:** The row is too large to fit into an empty data block. The server process stores the data for the row in a chain of two or more data blocks. Chaining can occur when the row is inserted or updated. Row chaining usually occurs with large rows, such as rows that contain a large object (LOB), LONG, or LONG RAW data types. Row chaining in these cases is unavoidable, unless a larger block size is used.

Migration and chaining have a negative effect on performance:

- `INSERT` and `UPDATE` statements that cause migration and chaining perform poorly because they perform additional processing.
- Queries that use an index to select migrated or chained rows must perform additional I/Os.

Migration is caused by `PCTFREE` being set too low; there is not enough room in the block kept for updates. To avoid migration, all tables that are updated should have their `PCTFREE` set so that there is enough space within the block for updates.

In a free list–managed segment, changing the `PCTFREE` value does not affect blocks that are already filled. It applies only to blocks that are added to the free list in subsequent operations.

If you change the `PCTFREE` value in an ASSM-managed segment, you must run the `DBMS_REPAIR.SEGMENT_FIX_STATUS` procedure to implement the new setting on blocks already allocated to the segment.

Changing the `PCTFREE` value changes the amount of free space reserved for updates in blocks that are not yet full. This helps prevent row migration in future updates, but does not change already migrated rows or blocks that are already full.

Guidelines for PCTFREE and PCTUSED

- PCTFREE
 - Default: 10
 - Zero if no UPDATE activity
 - $PCTFREE = 100 \times UPD / (\text{average row length})$
- PCTUSED
 - Only with free lists
 - Default: 40
 - Set if rows are deleted
 - $PCTUSED = 100 - PCTFREE - (100 \times \text{rows} \times \text{average row length} / \text{block size})$



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Guidelines for Setting the Values of PCTFREE and PCTUSED

$$\text{PCTFREE} = 100 \times \text{UPD} / (\text{average row length})$$

$$\text{PCTUSED} = 100 - \text{PCTFREE} - (100 \times \text{rows} \times \text{average row length} / \text{block size})$$

where:

- UPD = Average amount added by updates, in bytes. This is determined by subtracting the average row length after the insert from the current average row length.
- Average row length: After gathering statistics on the table, get this value from the AVG_ROW_LEN column of DBA_TABLES.
- Rows = Number of rows to be deleted before free-list maintenance occurs
- The space between PCTFREE and PCTUSED should provide enough space in the block for row updates and for at least one more row.

PCTUSED is relevant only in tables that undergo deletes. In many tables, you may be able to pack rows into blocks more tightly by setting PCTUSED to be higher than the default. Choose a value for PCTUSED that balances data density and free-list contention. For tables with many DELETE and INSERT statements, increasing PCTUSED can improve block storage performance. Fewer blocks are required, tables are smaller, and full table scan operations are faster. However, blocks that are densely populated with a high PCTUSED can cause free-list contention by being placed back on the free list after a very few deletes.

Detecting Migration and Chaining

Use the ANALYZE command to detect migration and chaining:

```
SQL> ANALYZE TABLE oe.orders COMPUTE STATISTICS;
Table Analyzed.
SQL> SELECT num_rows, avg_row_len, chain_cnt
  2   FROM DBA_TABLES
  3  WHERE table_name='ORDERS';
  NUM_ROWS AVG_ROW_LEN  CHAIN_CNT
-----  -----
    1171          67        83
```

Detect migration and chaining by using Statspack or AWR:

Statistic	Total	Per transaction...
table fetch continued row	495	.02 ...



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The ANALYZE Command

You can identify the existence of migrated and chained rows in a table or cluster by using the ANALYZE command. This command counts the number of migrated and chained rows and places this information into the CHAIN_CNT column of DBA_TABLES.

The NUM_ROWS column provides the number of rows stored in the analyzed table or cluster. Compute the ratio of chained and migrated rows to the number of rows to decide whether migrated rows need to be eliminated.

The AVG_ROW_LEN column gives an indication of whether the chain count is due to migrated rows or rows that are too long for any block.

The Table Fetch Continued Row Statistic

You can also detect migrated or chained rows by checking the Table Fetch Continued Row statistic in V\$SYSSTAT or in the Statspack report under "Instance Activity Stats for DB."

Guidelines

Increase PCTFREE to avoid migrated rows. If you leave more free space available in the block for updates, the row has room to grow. You can also reorganize (re-create) tables and indexes with a high deletion rate.

Selecting Migrated Rows

```
SQL> ANALYZE TABLE oe.orders LIST CHAINED ROWS;
Table analyzed.

SQL> SELECT owner_name, table_name, head_rowid
  2   FROM chained_rows
  3  WHERE table_name = 'ORDERS';
OWNER_NAME    TABLE_NAME    HEAD_ROWID
-----        -----        -----
SALES         ORDER_HIST   AAAAluAAHAAAAA1AAA
SALES         ORDER_HIST   AAAAluAAHAAAAA1AAB
...
...
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can identify migrated and chained rows in a table or cluster by using the ANALYZE command with the LIST CHAINED ROWS option. This command collects information about each migrated or chained row and places this information into a specified output table. To create the table that holds the chained rows, execute the `utlchain.sql` script to create the table named CHAINED_ROWS as follows:

```
create table CHAINED_ROWS (
  owner_name      varchar2(128),
  table_name      varchar2(128),
  cluster_name    varchar2(128),
  partition_name  varchar2(128),
  subpartition_name  varchar2(128),
  head_rowid      rowid,
  analyze_timestamp date)
```

If you create this table manually, it must have the same column names, data types, and sizes as the CHAINED_ROWS table.

Note: Using the command ANALYZE TABLE ... LIST CHAINED ROWS has the advantage of *not* overwriting the current statistics.

Eliminating Migrated Rows

- Export/import:
 - Export the table.
 - Drop or truncate the table.
 - Import the table.
- The MOVE table command:
 - ALTER TABLE EMPLOYEES MOVE
- Online table redefinition
- Copy migrated rows:
 - Find migrated rows by using ANALYZE.
 - Copy migrated rows to a new table.
 - Delete migrated rows from the original table.
 - Copy rows from the new table to the original table.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Any procedure that inserts rows into a table removes migrated rows.

Export and import eliminate migrated rows by inserting all the rows into a re-created table, but the table is unavailable for the duration of the operation.

The MOVE table command is faster than export and import. It requires two times the space during the operation, and the table is unavailable. All indexes will be unusable and must be rebuilt after the move operation.

Online redefinition allows the table to be available for DML and SELECT commands for the duration of the operation, and the indexes are maintained. This method requires the use of the DBMS_REDEFINITION package. This method requires the most space, because there are two copies of the table, a journal table, and multiple materialized views.

The copy method locks only the rows being moved and uses additional space only for the chained rows. You can eliminate migrated rows by using this SQL*Plus script:

```
/* Get the name of the table with migrated rows */  
ACCEPT table_name PROMPT 'Enter the name of the table with migrated  
rows: '
```

```
/* Clean up from last execution */
SET ECHO OFF
DROP TABLE migrated_rows;
DROP TABLE chained_rows;

/* Create the CHAINED_ROWS table */
@?/rdbms/admin/utlchain
SET ECHO ON
SPOOL fix_mig
/* List the chained & migrated rows */
ANALYZE TABLE &table_name LIST CHAINED ROWS;

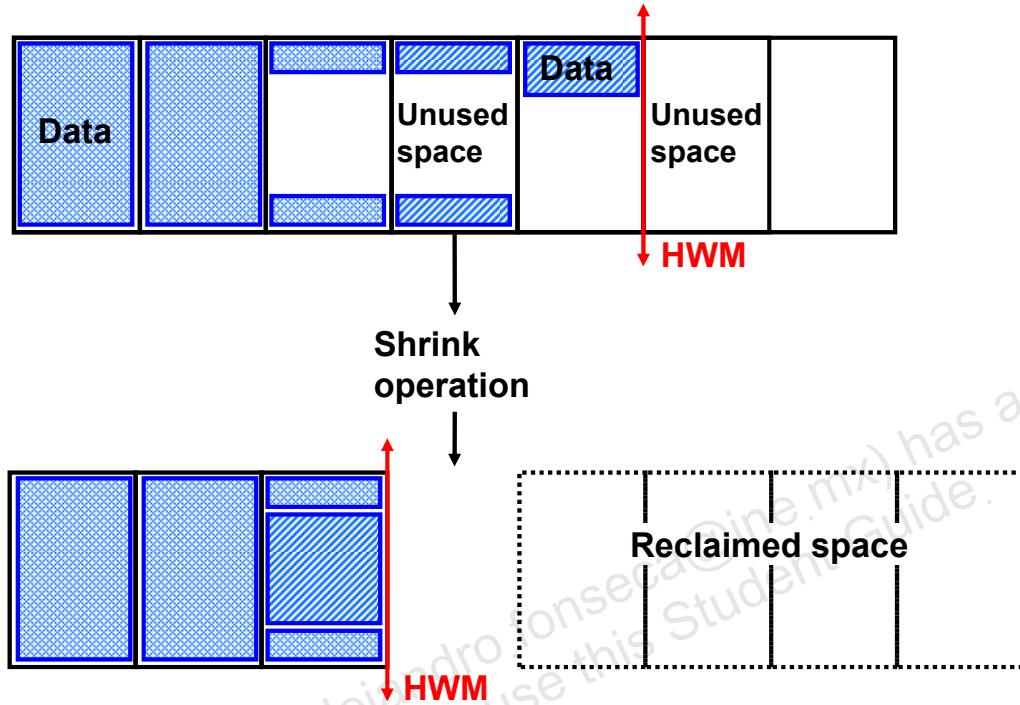
/* Copy the chained/migrated rows to another table */
CREATE TABLE migrated_rows AS
  SELECT orig.*
    FROM &table_name orig, chained_rows cr
   WHERE orig.rowid = cr.head_rowid
     AND cr.table_name = upper('&table_name');

/* Delete the chained/migrated rows from the original table */
DELETE FROM &table_name
WHERE rowid IN (
  SELECT head_rowid
    FROM chained_rows);

/* Copy the chained/migrated rows back into the original table */
INSERT INTO &table_name
  SELECT *
    FROM migrated_rows;
SPOOL OFF
```

When using this script, you must disable any foreign key constraints that would be violated when the rows are deleted. Additional considerations are the implications of insert and delete triggers, row-level security, and auditing.

Shrinking Segments: Overview



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The upper part of the diagram in the slide shows a sparsely populated segment. There is some unused space both above and below the segment's high-water mark (HWM). Shrinking a sparsely populated segment improves the performance of scan and DML operations on that segment. This is because there are fewer blocks to look at after the segment has been shrunk. This is especially true for:

- Full table scans (fewer and denser blocks)
- Better index access (fewer I/Os on range ROWID scans due to a more compact tree)

You can make more free space available in tablespaces by shrinking sparsely populated segments. When a segment is shrunk, its data is compacted, its HWM is pushed down, and unused space is released back to the tablespace containing the segment.

Row IDs are not preserved. When a segment is shrunk, the rows move inside the segment to different blocks, causing the row ID to change.

Note: The number of migrated rows in a segment may be reduced as a result of a segment shrink operation. Because a shrink operation does not touch all the blocks in the segment, you cannot depend on reducing the number of migrated rows after a segment has been shrunk.

Shrinking Segments: Considerations

- A shrink operation is an online and in-place operation.
- It is applicable only to segments residing in ASSM tablespaces.
- Candidate segment types:
 - Heap-organized tables and index-organized tables
 - Indexes
 - Partitions and subpartitions
 - Materialized views and materialized view logs
 - LOB segments
- Indexes are maintained.
- Triggers are not fired.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A shrink operation is an online and in-place operation.

You cannot execute a shrink operation on segments managed by free lists. Segments in automatic segment space-managed tablespaces can be shrunk. However, the following objects cannot be shrunk:

- Tables in clusters
- Tables with LONG columns
- Tables with on-commit materialized views
- IOT mapping tables, even with the cascade clause
- IOT overflow segments
- Compressed tables

Other objects have restrictions:

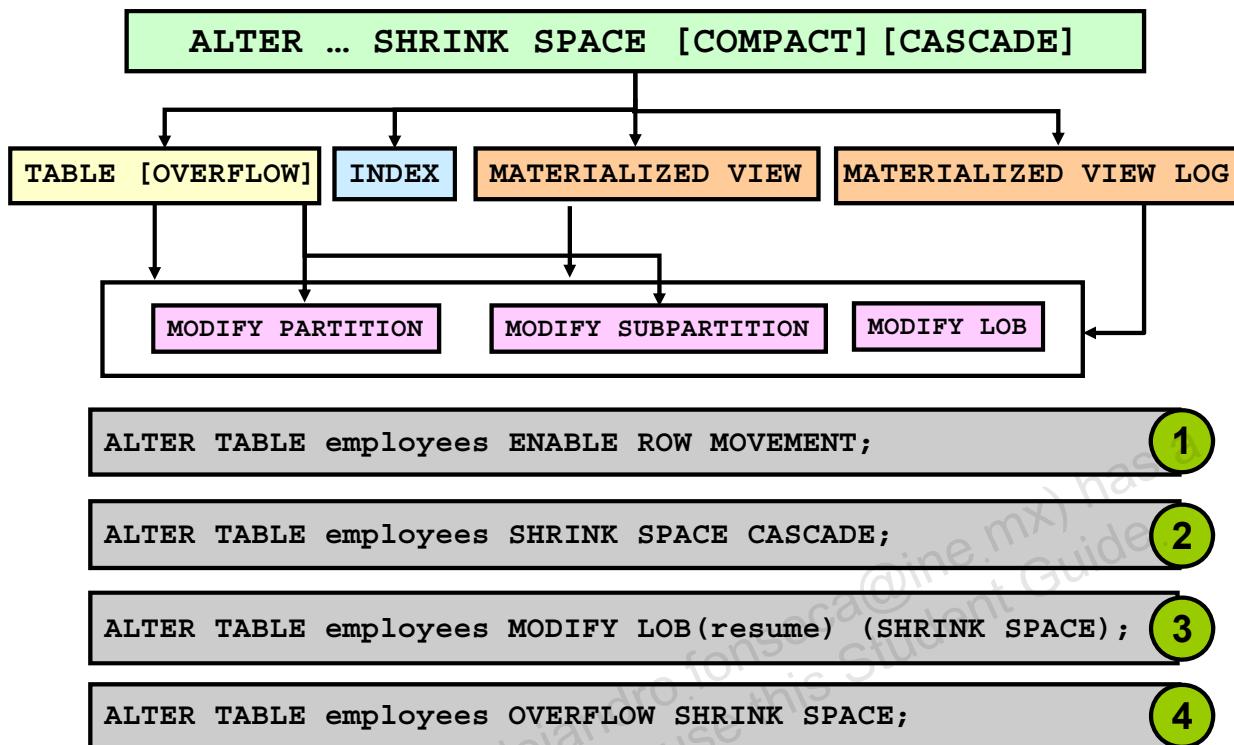
- ROWID-based materialized views must be rebuilt after a shrink operation on base tables.
- Tables with function-based indexes are not supported
- Tables with bitmap-join indexes are not supported

ROW MOVEMENT must be enabled for heap-organized segments.

Indexes are in a usable state after shrinking the corresponding table.

The actual shrink operation is handled internally as an INSERT/DELETE operation. However, DML triggers are not fired because the data itself is not changed.

Shrinking Segments by Using SQL



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Because a shrink operation may cause ROWIDs to change in heap-organized segments, you must enable row movement on the corresponding segment before executing a shrink operation on that segment. Row movement by default is disabled at the segment level. To enable row movement, the `ENABLE ROW MOVEMENT` clause of the `CREATE TABLE` or `ALTER TABLE` command is used. This is illustrated in the first example in the slide.

Use the `ALTER` command to invoke segment shrink on an object. The object's type can be one of the following: table (heap- or index-organized), partition, subpartition, LOB (data and index segment), index, materialized view, or materialized view log. Use the `SHRINK SPACE` clause to shrink space in a segment. If `CASCADE` is specified, the shrink behavior is cascaded to all the dependent segments that support a shrink operation, except materialized views, LOB indexes, and IOT (index-organized tables) mapping tables. The `SHRINK SPACE` clause is illustrated in the second example.

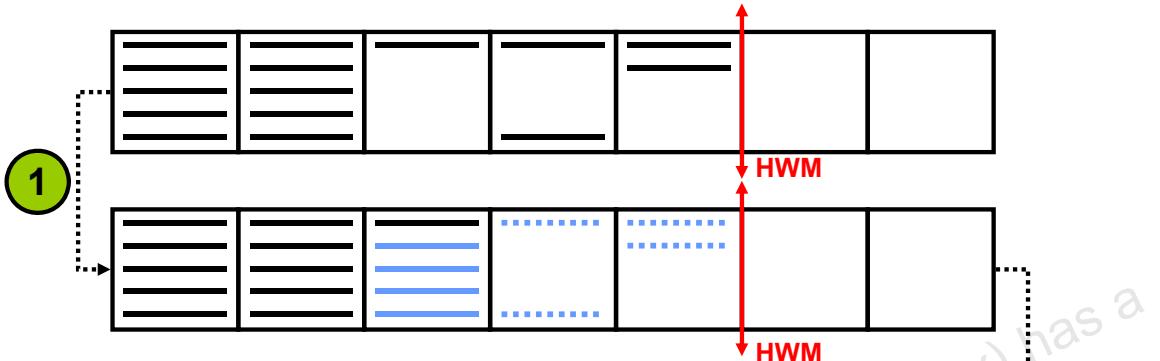
In an index segment, the shrink operation coalesces the index before compacting the data.

Example 3 shows a command that shrinks a LOB segment, given that the `RESUME` column is a `CLOB`. Example 4 shows a command that shrinks an IOT overflow segment belonging to the `EMPLOYEES` table.

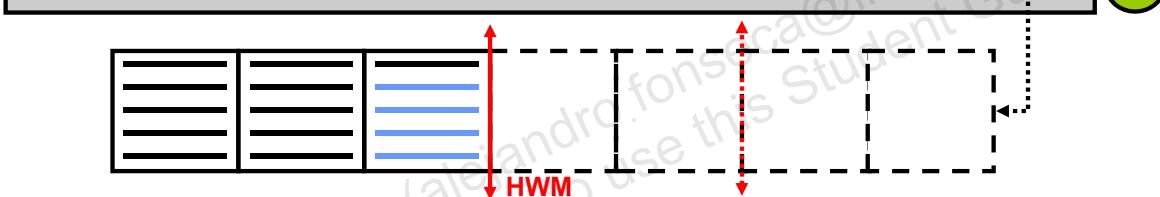
Note: For more information, refer to the *Oracle Database SQL Reference* guide.

Segment Shrink: Basic Execution

```
ALTER TABLE employees SHRINK SPACE COMPACT;
```



```
ALTER TABLE employees SHRINK SPACE;
```



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The diagram in the slide describes the two phases of a table shrink operation. Compaction is performed in the first phase. During this phase, rows are moved to the left part of the segment as much as possible. After the rows have been moved, the second phase of the shrink operation is started. During this phase, the HWM is adjusted, and the unused space is released.

During a shrink operation, you can execute only the compaction phase by specifying the SHRINK SPACE COMPACT clause. This is illustrated by the first example in the slide.

As shown by the second example in the slide, if COMPACT is not specified, the segment space is compacted, and at the end of the compaction phase, the HWM is adjusted and the unused space is released.

Segment Shrink: Execution Considerations

- Use compaction only:
 - To avoid unnecessary cursor invalidation
 - During peak hours
- DML operations and queries can be issued during compaction.
- DML operations are blocked when HWM is adjusted.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The compaction phase of a segment shrink operation is performed online. Conventional DML operations as well as queries can coexist with a segment shrink operation.

During the compaction phase, locks are held on individual rows containing the data. Concurrent DML operations such as updates and deletes serialize on these locks. However, only packets of rows are locked at one time to avoid the locking of the entire segment.

Similarly, conventional DML operations can block the progress of the compaction phase until they commit.

The `COMPACT` clause is useful if you have long-running queries that might span the shrink operation and attempt to read from blocks that have been reclaimed. When you specify the `SHRINK SPACE COMPACT` clause, the progress of the shrink operation is saved in the bitmap blocks of the corresponding segment. This means that the next time a shrink operation is executed on the same segment, the Oracle database remembers what has been done already. You can then reissue the `SHRINK SPACE` clause without the `COMPACT` clause during off-peak hours to complete the second phase.

During the second phase of the segment shrink operation, when the HWM is adjusted, the object is locked in exclusive mode. This occurs for a very short duration and does not affect the availability of the segment significantly. Dependent cursors are invalidated.

Using Enterprise Manager to Shrink Segments

The shrink operation compacts fragmented space and, optionally, frees the space. The shrink operation will take some time and will be scheduled as a job.

Shrink Options

- Compact Segments and Release Space
This will first compact the segments and then release the recovered space to the tablespace. During the short space release phase, any cursors referencing this segment may be invalidated.
- Compact Segments
Compacting will compact segment data without releasing the recovered space. After compacting the data, the recovered space can be quickly released by running Compact Segments and Release Space.

Segment Selection

- Shrink HR.EMPLOYEES Only
- Shrink HR.EMPLOYEES and All Dependent Segments

Schema	Segment Name	Type	Tablespace
HR	EMPLOYEES	TABLE	EXAMPLE
HR	EMP_MANAGER_IX	INDEX	EXAMPLE
HR	EMP_NAME_IX	INDEX	EXAMPLE
HR	EMP_DEPARTMENT_IX	INDEX	EXAMPLE
HR	EMP_JOB_IX	INDEX	EXAMPLE
HR	EMP_EMAIL_UK	INDEX	EXAMPLE
HR	EMP_EMP_ID_PK	INDEX	EXAMPLE

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Using Enterprise Manager Cloud Control, you can shrink individual segments as follows:

1. On the database home page, expand the Schema menu, expand the Database Objects menu, and click the Tables link.
2. On the Tables page, select your table, and then select Shrink Segment in the Actions drop-down list. Click Go.
3. On the Shrink Segment page, choose the dependent segments to shrink. You can either compact only, or compact and release the space. You can also choose the CASCADE option in the Segment Selection section by selecting “Shrink HR.EMPLOYEES and All Dependent Segments.”
4. Click the Continue button. This allows you to submit the shrink statements as a scheduled job.

Note: Before shrinking a heap-organized table, you must enable row movement on that table. You can do that by accessing the Options tab of the Edit Table page in Enterprise Manager Cloud Control.

Data Compression

Row Compression

- Basic
- Advanced

Oracle Hybrid Columnar Compression

- Query
- Archive

Compression Type

- ROW STORE COMPRESS [BASIC]
- ROW STORE COMPRESS ADVANCED
- COLUMN STORE COMPRESS FOR QUERY LOW / HIGH
- COLUMN STORE COMPRESS FOR ARCHIVE LOW / HIGH

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Oracle offers multiple types and levels of compression. Basic compression is included in the Enterprise Edition of the database; Advanced compression requires licensing the Advance Compression Option (ACO); and Oracle Hybrid Columnar Compression (HCC) is available with Exadata Storage Cells, Solaris XFS, and Pillar Axiom storage. Compression for SecureFile LOBS is specified in the LOB storage clause. For details of SecureFile LOB compression, see *Oracle Database SecureFiles and Large Objects Developer's Guide 12c Release 1*. The compression types shown in the slide may be applied to heap organized tables.

Basic compression works only on bulk loads. The database does not compress data modified using conventional DML. You must use direct-path INSERT operations, ALTER TABLE . . . MOVE operations, or online table redefinition to achieve basic table compression. This level is designed for use with DSS and DW systems and seldom-updated rows.

Advanced compression works with DML operations, and is designed for OLTP systems.

Both the Basic and Advanced compression types perform compression at the block level, and so do not require any uncompression.

The parameter COLUMN STORE causes Oracle Hybrid Columnar Compression to be applied. The parameter QUERY HIGH provides a higher compression ratio, but less CPU is used when the QUERY LOW parameter is specified. ARCHIVE HIGH achieves the highest ratios, but with an increase in CPU overhead, ARCHIVE LOW reduces the CPU usage and compression ratio.

Using the CREATE or ALTER TABLE command, you can specify table compression:

- For an entire table (in the PHYSICAL_PROPERTIES clause of RELATIONAL_TABLE or OBJECT_TABLE)
- For partitioned tables (Each partition can have a different type or level of compression.)
- For the storage of a nested table (in the NESTED_TABLE_COL_PROPERTIES clause)

Table compression has the following restrictions:

- Data segments of BasicFile LOBs are not compressed. For information about compression of SecureFile LOBs, see *LOB_compression_clause* in *Oracle Database SQL Language Reference*.
- You cannot specify table compression for an index-organized table, any overflow segment or partition of an overflow segment, or any mapping table segment of an index-organized table.
- You cannot specify table compression for external tables or for tables that are part of a cluster.
- You cannot drop a column from a table that is compressed for direct-load operations, although you can set such a column as unused. All the operations of *ALTER TABLE ... drop_column_clause* are valid for tables that are compressed for all operations.

Even though HCC compression can incur a CPU penalty for uncompression, the reduced I/O can more than compensate for that cost through fewer data block reads, reduced network load, reduced backup time, and space requirements.

Advanced Row Compression: Overview

- Oracle Database includes compression for OLTP data.
 - It provides support for conventional DML operations (INSERT, UPDATE, DELETE).
- Algorithm significantly reduces write overhead.
 - Batched compression ensures no impact for most OLTP transactions.
- Table compression has no impact on reads.
 - Reads may actually see improved performance due to fewer I/Os and enhanced memory efficiency.



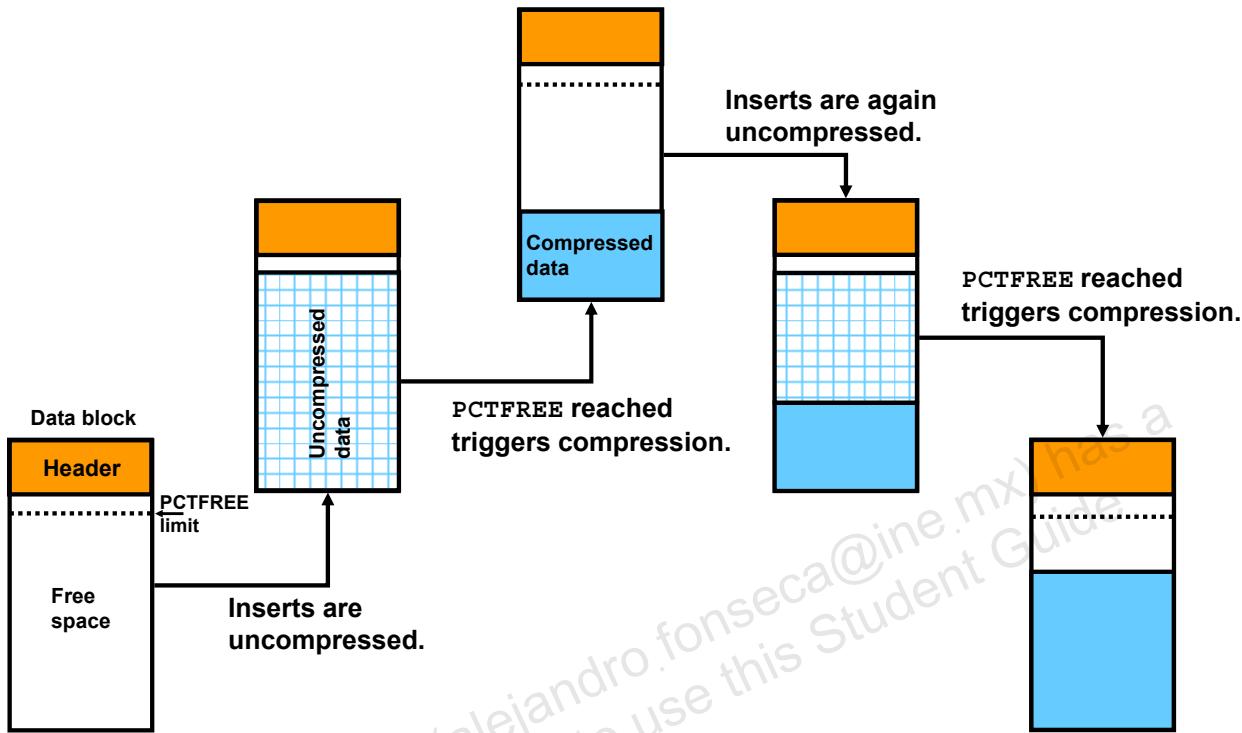
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Oracle Database includes compression technology to support regular data manipulation operations such as INSERT, UPDATE, and DELETE. Compression in Oracle Database can be used for all kinds of workloads—online transaction processing (OLTP) or data warehousing.

Compression technology in Oracle Database is very efficient and could reduce space consumption by 50–75%. So, your write performance does not degrade, and your read performance or queries improve. This is because unlike desktop-based compression techniques where you have to wait for data to be uncompressed, Oracle technology reads the compressed data (less fetches needed) directly and does not require any uncompress operation.

Note: Compression technology is completely application-transparent. This means that you can use this technology with any application such as SAP, Siebel, or EBS.

Advanced Row Compression: Concepts



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The slide shows you a data block evolution when that block is part of a table compressed with the Advanced Row Compression option. You should read it from left to right. At the start, the block is empty and available for inserts. When you start inserting into this block, data is stored in an uncompressed format (like for uncompressed tables). However, as soon as you reach PCTFREE of that block, the data is automatically compressed, potentially reducing the space it originally occupied. This allows new uncompressed inserts to take place in the same block, until PCTFREE is reached again. At that point, compression is triggered again to reduce space occupation in the block.

Note: Compression eliminates holes created due to deletions and maximizes contiguous free space in blocks.

Both Advanced Row Compression and Basic use a compression method where repeating field values are replaced with a pointer, and the value is stored once in the block. Basic compression can be used only with direct-path `INSERT` methods; no compression is applied for standard inserts or updates.

Using Advanced Row Compression

- Requires database compatibility level at 11.1 or greater
- Use the COMPRESS keyword:
 - ROW STORE COMPRESS BASIC is the default
 - ROW STORE COMPRESS ADVANCED for OLTP plus direct loads
- Enable compression for new tables:

```
CREATE TABLE t1 ROW STORE COMPRESS ADVANCED
```

- Enable compression on existing table:

```
ALTER TABLE t2 ROW STORE COMPRESS ADVANCED
```

- Does not trigger compression on existing rows



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To use the Advanced Row Compression algorithm, you must flag your table with the ROW STORE COMPRESS ADVANCED clause. You can do so at table creation, or after creation. This is illustrated in the examples given in the slide.

If you use the ROW STORE COMPRESS clause without specifying any option, or if you use the ROW STORE COMPRESS BASIC clause, you enable basic table compression.

You can also enable compression at the partition or tablespace level. For example, you can use the DEFAULT storage clause of the CREATE TABLESPACE command to optionally specify a COMPRESS FOR clause.

Note: You can view compression flags for your tables using the COMPRESSION and COMPRESS_FOR columns in views such as DBA_TABLES and DBA_TAB_PARTITIONS.

Oracle Hybrid Columnar Compression

- Combines Row and Column compression techniques for 5 to 50X compression ratios
- Only in Oracle Storage:
 - Solaris ZFS
 - Axiom Pillar
 - Exadata Storage Cell



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Oracle Hybrid Columnar Compression is available only with the enterprise edition of the database using Oracle Storage: Solaris ZFS, Axiom Pillar, or Exadata Storage Cell.

How Does Columnar Compression Work?

- Reorganize by column, in a compression unit
- Apply compression



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In general terms, the hybrid columnar compression works by reorganizing the rows major structure of the standard RDBMS to a columnar major arrangement. Then compression is applied.

The hybrid part of the name refers to the ability to continue to perform updates to the rows, though columnar compression has a higher overhead for DML operations. All levels of columnar compression are unsuited for frequently updated rows or segments having active DML.

The rows of a compression unit (multiple blocks) are reorganized by column. This places all the values of a column in close proximity; these values are of the same datatype, and are frequently repeating values. These characteristics allow higher compression ratios. Replacing the repeated values with a token becomes more efficient when there are more repeated values. The compression unit allows more rows, and by extension, more repeating values in the same unit; therefore, the amount of space taken by a symbol table is reduced. Another option is that inside the compression unit, a single value could be listed and followed by a list of row IDs where it is referenced. In addition, when the values are all of the same data type, the various compression algorithms used at the different levels produce better compression ratios.

Using the Compression Advisor

- A compression advisor helps to determine optimal compression ratios.
- The DBMS_COMPRESSION package includes the GET_COMPRESSION_RATIO procedure.

```
BEGIN
DBMS_COMPRESSION.GET_COMPRESSION_RATIO ('USERS','SH','SALES',
NULL,DBMS_COMPRESSION.COMP_FOR_OLTP, blkcnt_cmp, blkcnt_uncmp,
rowcnt_cmp, rowcnt_uncmp, comptype);
DBMS_OUTPUT.PUT_LINE('Blk count compressed = ' || blkcnt_cmp);
DBMS_OUTPUT.PUT_LINE('Blk count uncompressed = ' || blkcnt_uncmp);
DBMS_OUTPUT.PUT_LINE('Row count per block compressed = ' ||
rowcnt_cmp);
DBMS_OUTPUT.PUT_LINE('Row count per block uncompressed = ' ||
rowcnt_uncmp);
DBMS_OUTPUT.PUT_LINE('Compression type = ' || comptype);
DBMS_OUTPUT.PUT_LINE('Compression ratio =
' || blkcnt_uncmp/blkcnt_cmp|| ' to 1');

```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A compression advisor, provided by the DBMS_COMPRESSION package, helps you to determine the compression ratio that can be expected for a specified table. The advisor analyzes the objects in the database, discovers the possible compression ratios that could be achieved, and recommends the optimal compression levels. In addition to the DBMS_COMPRESSION package, the compression advisor can also be used within the existing advisor framework (with the DBMS_ADVISOR package).

To determine the compression ratio, the DBMS_COMPRESSION package has the following subprograms:

- The GET_COMPRESSION_RATIO procedure gives you the possible compression ratio for an uncompressed table, for specified compression type specified by a constant, such as DBMS_COMPRESSION.COMP_FOR_OLTP, shown in the slide.
- The GET_COMPRESSION_TYPE function returns the compression type for a given row.

For more details, refer to *Oracle Database PL/SQL Packages and Types Reference*.

Viewing Table Compression Information

* _TABLES views have the following columns:

- COMPRESSION: Indicates whether table compression is enabled (ENABLED) or not (DISABLED)
- COMPRESS_FOR: Type of compression



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The * _TABLES views have the COMPRESSION and COMPRESS_FOR columns, which enable you to view compression information.

Quiz

Advanced Row Compression reduces the space required to store data. The amount of compression you can achieve depends on:

- a. The number of columns in the table
- b. The number of repeated values in the columns
- c. The length of the row
- d. The compression algorithm you select



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: b

Advanced Row Compression stores repeated column values once per block and places pointers in the row. The compression is better when there are more repeated values in the columns.

Summary

In this lesson, you should have learned how to:

- Adjust data structures to influence the optimizer
- Tune segment space management
- Use the Segment Space Advisor
- Convert to Automatic Segment Space Management
- Tune block space management
- Diagnose and correct row migration
- Diagnose table fragmentation
- Use table compression

Practice 12 Overview: Reducing the Cost of SQL Operations

This practice covers the following topics:

- Managing space usage
 - Observe the changes to the explain plan when excess blocks are used in a table.
- Managing B*Tree indexes
 - Observe the changes to the actual cost when indexes are coalesced.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2018, Oracle and/or its affiliates.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

13

Using the SQL Performance Analyzer

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Identify the benefits of using SQL Performance Analyzer
- Describe the SQL Performance Analyzer workflow phases
- Use SQL Performance Analyzer to:
 - Ascertain performance gains following a database change
 - Test the impact of proposed changes

Real Application Testing: Overview

SQL Performance Analyzer and Database Replay are key components of Oracle Real Application Testing and offer complementary solutions.

	SQL Performance Analyzer	Database Replay
What is it used for?	To predict SQL performance deviations	To replay real database workload on test system
What is its purpose?	To assess impact of change on SQL response time	To assess impact of change on workload throughput
How does it work?	Single, isolated execution of SQL	Replay actual workload
When should it be used?	When testing application SQL to identify the set of SQL statements with changed performance	When comprehensively testing all subsystems of the database server using real production workload



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Oracle Real Application Testing option and test data management features of Oracle Database help you to assure the integrity of database changes and to manage test data. The Oracle Real Application Testing option enables you to perform real-world testing. By capturing production workloads and assessing the impact of system changes on these workloads before production deployment, Oracle Real Application Testing minimizes the risk of instabilities associated with system changes. SQL Performance Analyzer and Database Replay are key components of Oracle Real Application Testing. Depending on the nature and impact of the system change being tested, and on the type of system the test will be performed, you can use either or both the components to perform the test.

SQL Performance Analyzer (SPA)

SPA is the tool of choice when you are trying to identify SQL statements that will perform differently when a change is made at the database or OS level. SQL Performance Analyzer captures SQL statements into a SQL tuning set (STS) from various sources, including the cursor cache, Automatic Workload Repository (AWR), and existing SQL tuning sets. The STS is analyzed by executing each SQL statement in isolation. The order of execution depends on the order of the statement in the tuning set. The STS includes bind variable, execution plan, and execution context information. With SQL Performance Analyzer, you will execute the STS and capture performance statistics, make the change to the system and execute the STS again, and then compare. SQL Performance Analyzer does not consider the impact that SQL statements can have on each other.

Database Replay

When there are extensive changes such as OS, or database version, or when overall system performance must be tested, Database Replay is the preferred tool. Database Replay captures the actual workload over a period of time, along with performance statistics. This workload is processed for the target system and then replayed. The replay has options for speed of replay, think time, maintaining the transaction order (synchronization), and others. After the workload has been replayed, the performance statistics, data divergence, and errors can be compared to the original capture or another replay session. AWR and Replay reports are available.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

Real Application Testing: Use Cases

Real Application Testing is beneficial in the following use cases:

- Database upgrades
- Implementation of tuning recommendations
 - Example: Accepting SQL profiles
- Schema changes
 - Example: Adding indexes or materialized views
- Statistics gathering
- Database parameter changes
- OS and hardware changes



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Real Application Testing option enables you to test the impact of database and OS changes including, but not limited to any of the following:

- Database upgrades
- Implementation of tuning recommendations
- Schema changes
- Statistics gathering
- Database parameter changes
- OS and hardware changes

SQL Performance Analyzer can be used to predict and prevent potential performance problems for any database environment change that affects the structure of the SQL execution plans. DBAs can use SQL Performance Analyzer to foresee SQL performance changes that result from the preceding changes for even the most complex environments. As applications evolve through the development life cycle, database application developers can test (for example, changes to schemas, database objects, and rewritten applications) to mitigate any potential performance impact. SQL Performance Analyzer enables the comparison of SQL performance statistics.

Database Replay can be used for the same change scenarios, but tests the entire workload, honoring transaction dependencies. Database Replay executes all statements including DML. The entire workload is tested, including interactions and dependencies.

SQL Performance Analyzer: Process

1. Capture SQL workload on production.
2. Transport the SQL workload to a test system.
3. Build “before-change” performance data.
4. Make changes.
5. Build “after-change” performance data.
6. Compare results from steps 3 and 5.
7. Tune regressed SQL.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

1. **Gather SQL:** In this phase, you collect the set of SQL statements that represent your SQL workload on the production system. You can use SQL tuning sets or Automatic Workload Repository (AWR) to capture the information to transport. Because AWR essentially captures high-load SQLs, you should consider modifying the default AWR snapshot settings and captured Top SQL to ensure that AWR captures the maximum number of SQL statements. This ensures more complete SQL workload capture. You can also use the filtering and ranking capabilities of the STS to change the SQL statements that are included.
2. **Transport:** Here you transport the resultant workload to the test system. The STS is exported from the production system and the STS is imported into the test system.
3. **Compute “before-version” performance:** Before any changes take place, you execute the SQL statements, collecting baseline information that is needed to assess the impact that a future change might have on the performance of the workload. The information collected in this stage represents a snapshot of the current state of the system workload and includes performance data. In this phase, you can test the SQL by executing the STS or you can have the SQL Performance Analyzer task only generate the execution plans, without executing the statements.
4. **Make a change:** After you have the before-version data, you can implement your planned change and start viewing the impact on performance.

5. **Compute “after-version” performance:** This step takes place after the change is made in the database environment. Each statement of the SQL workload executes again, collecting the same information as captured in step 3.
6. **Compare and analyze SQL Performance:** After you have both versions of the SQL workload performance data, you can carry out the performance analysis by comparing the after-version data with the before-version data. The comparison is based on the execution statistics, such as elapsed time, CPU time, and buffer gets.
7. **Tune regressed SQL:** At this stage, you have identified exactly which SQL statements may cause performance problems when the database change is made. Here, you can use any of the database tools to tune the system. For example, you can use SQL Tuning Advisor or Access Advisor against the identified statements and then implement those recommendations. Alternatively, you can seed SQL Plan Management (SPM) with plans captured in step 3 to guarantee that the plans remain the same. After implementing any tuning action, you should repeat the process to create a new after-version and analyze the performance differences to ensure that the new performance is acceptable.

Capturing the SQL Workload

- Create a SQL tuning set on the original system.
- Create a staging table and pack the STS in it.
- Export the staging table to the test system.
- Unpack the staging table to the STS on the test system.

The screenshot shows the 'SQL Tuning Sets' page in Oracle Enterprise Manager. At the top, there's a search bar and a 'Go' button. Below that is a table with columns: Select, Name, Schema, Description, SQL Count, Created, and Last Modified. There are three rows in the table:

Select	Name	Schema	Description	SQL Count	Created	Last Modified
<input checked="" type="radio"/>	STS_JFV	SYS		40	11/11/13 6:54 AM	11/11/13 6:55 AM
<input checked="" type="radio"/>	CAPTURE_orcl_11GSim_1375585389	SYS	STS capture for capture with ID=1	686	10/28/13 7:31 AM	10/28/13 7:31 AM
<input checked="" type="radio"/>	CAPTURE_orcl_11GSim_c_160791	SYS	STS capture for capture with ID=1	686	10/28/13 7:16 AM	10/28/13 7:22 AM

At the bottom right of the page is the ORACLE logo.

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Capturing a SQL workload is done by using SQL tuning sets and transporting it to the target system. This workflow is briefly described in the slide. You can use either Enterprise Manager Cloud Control wizards or the DBMS_SQLTUNE PL/SQL package.

In Enterprise Manager Cloud Control, you can access the SQL Tuning Sets page by expanding the Performance menu and the SQL menu, and then selecting SQL Tuning Sets.

The workload that you capture should reflect a representative period of time (in captured SQL statements) that you wish to test under some changed condition. The following information is captured in this process:

- The SQL text
- The execution context (including bind values, parsing schema, and compilation environment), which contains a set of initialization parameters under which the statement is executed
- The execution frequency, which tells how many times the SQL statement has been executed during the time interval of the workload

Normally, the capture SQL happens on the production system to capture the workload running on it. The performance data is computed later on the test system by the compute SQL performance processes. SQL Performance Analyzer tracks the SQL performance of the same STS before and after a change is made to the database.

Creating a SQL Performance Analyzer Task

Advisor Central > SQL Performance Analyzer Logged in as SYS

SQL Performance Analyzer

Page Refreshed Nov 11, 2013 1:25:14 PM UTC Refresh View Data Real Time: 15 Second Refresh ▾

SQL Performance Analyzer allows you to test and to analyze the effects of changes on the execution performance of SQL contained in a SQL Tuning Set.

SQL Performance Analyzer Workflows

Create and execute SQL Performance Analyzer Task experiments of different types using the following links.

- [Upgrade from 9i or 10.1](#) Test and analyze the effects of database upgrade from 9i or 10.1 on SQL Tuning Set performance.
- [Upgrade from 10.2 or 11g](#) Test and analyze the effects of database upgrade from 10.2 or 11g on SQL Tuning Set performance.
- [Parameter Change](#) Test and compare an initialization parameter change on SQL Tuning Set performance.
- [Optimizer Statistics](#) Test and analyze the effects of optimizer statistics changes on SQL Tuning Set performance.
- [Exadata Simulation](#) Simulate the effects of an Exadata Storage Server installation on SQL Tuning Set performance.
- [Guided Workflow](#) Create a SQL Performance Analyzer Task and execute custom experiments using manually created SQL trials.

SQL Performance Analyzer Tasks

		Delete	View Latest Report						
Select	Name	Owner	Last Modified ▾	Current Step Name	Type	Last Run Status	SQLs Processed	Steps Completed	
	SPA_JFV1	SYS	Nov 11, 2013 9:22:51 AM	COMPARE_1384161771038	Compare	Completed	40 of 40	1 of 1	

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

With Enterprise Manager Cloud Control, you can manage each component in the SQL Performance Analyzer process and report the analysis result. You can access the SQL Performance Analyzer by expanding the Performance menu and the SQL menu, and then selecting SQL Performance Analyzer.

SQL Performance Analyzer offers the following workflows for you to test different scenarios:

- **Upgrade from 9i or 10.1:** Test and analyze the effects of database upgrade from 9i or 10.1 on SQL Tuning Set performance.
- **Upgrade from 10.2 or 11g:** Test and analyze the effects of database upgrade from 10.2 or 11g on SQL Tuning Set performance.
- **Parameter Change:** Test and compare an initialization parameter change on SQL tuning set performance. A SQL Performance Analyzer task is created and an initial trial run is performed with the parameter set to the base value. A second trial run is performed with the parameter set to the changed value. A replay trial comparison report is then run for the two trials.
- **Optimizer Statistics:** Test and analyze the effects of optimizer statistics changes on SQL Tuning Set performance.
- **Exadata Simulation:** Simulate the effects of an Exadata Storage Server installation on SQL Tuning Set performance.
- **Guided Workflow:** Create a SQL Performance Analyzer task and execute custom experiments by using manually created replay trials.

SQL Performance Analyzer: Tasks

Advisor Central > SQL Performance Analyzer Logged in as SYS

SQL Performance Analyzer

Page Refreshed Nov 11, 2013 1:25:14 PM UTC [Refresh](#) View Data Real Time: 15 Second Refresh

SQL Performance Analyzer allows you to test and to analyze the effects of changes on the execution performance of SQL contained in a SQL Tuning Set.

SQL Performance Analyzer Workflows

Create and execute SQL Performance Analyzer Task experiments of different types using the following links.

- Upgrade from 9i or 10.1 Test and analyze the effects of database upgrade from 9i or 10.1 on SQL Tuning Set performance.
- Upgrade from 10.2 or 11g Test and analyze the effects of database upgrade from 10.2 or 11g on SQL Tuning Set performance.
- Parameter Change Test and compare an initialization parameter change on SQL Tuning Set performance.
- Optimizer Statistics Test and analyze the effects of optimizer statistics changes on SQL Tuning Set performance.
- Exadata Simulation Simulate the effects of a Exadata Storage Server installation on SQL Tuning Set performance.
- Guided Workflow Create a SQL Performance Analyzer Task and execute custom experiments using manually created SQL trials.

SQL Performance Analyzer Tasks

Delete	View Latest Report	Select	Name	Owner	Last Modified	Current Step Name	Type	Last Run Status	SQLs Processed	Steps Completed
	SPA_JFV1	SYS	Nov 11, 2013 9:22:51 AM		COMPARE_1384161771038	Compare	Completed	100%	100%	Dec 14, 2007 10:04:05 AM

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

After you create your SQL Performance Analyzer task, it might take a long time for it to finish depending on the number of statements that are contained in your SQL tuning set. While your task is executing, you can click Refresh on the SQL Performance Analyzer page to update the Last Run Status field. When you see Completed in the Last Run Status column for your task, the task is complete.

After execution, you can click the link corresponding to the name of your task in the SQL Performance Analyzer Tasks table. This directs you to the corresponding SQL Performance Analyzer Task page.

SQL Performance Analyzer Task Page

SQL Performance Analyzer Task: SYS.SPA_JFV1

[View Latest Report](#) Page Refreshed **Nov 11, 2013 1:27:58 PM UTC** [Refresh](#)

The SQL Performance Analyzer Task is a container for experimental results of executing a specific SQL Tuning Set under changed environmental conditions and assessing the impact of environmental changes on STS execution performance.

▷ **SQL Tuning Set**

▽ **SQL Trials**

A SQL Trial captures the execution performance of the SQL Tuning Set under specific environmental conditions.

[Create SQL Trial](#)

SQL Trial Name	Description	Created	SQL Executed	Status
INITIAL_SQL_TRIAL	parameter optimizer_features_enable set to '11.2.0.2'	11/11/13 7:15 AM	Yes	COMPLETED
SECOND_SQL_TRIAL	parameter optimizer_features_enable set to '12.1.0.1'	11/11/13 7:18 AM	Yes	COMPLETED
AFTER_BASELINES		11/11/13 9:16 AM	Yes	COMPLETED

▽ **SQL Trial Comparisons**

Compare SQL Trials to assess change impact of environmental differences on SQL Tuning Set execution costs.

[Run SQL Trial Comparison](#)

Trial 1 Name	Trial 2 Name	Compare Metric	Created	Status	Comparison Report	SQL Tune Report
INITIAL_SQL_TRIAL	SECOND_SQL_TRIAL	Elapsed Time	11/11/13 7:21 AM	COMPLETED		
INITIAL_SQL_TRIAL	AFTER_BASELINES	Elapsed Time	11/11/13 9:22 AM	COMPLETED		

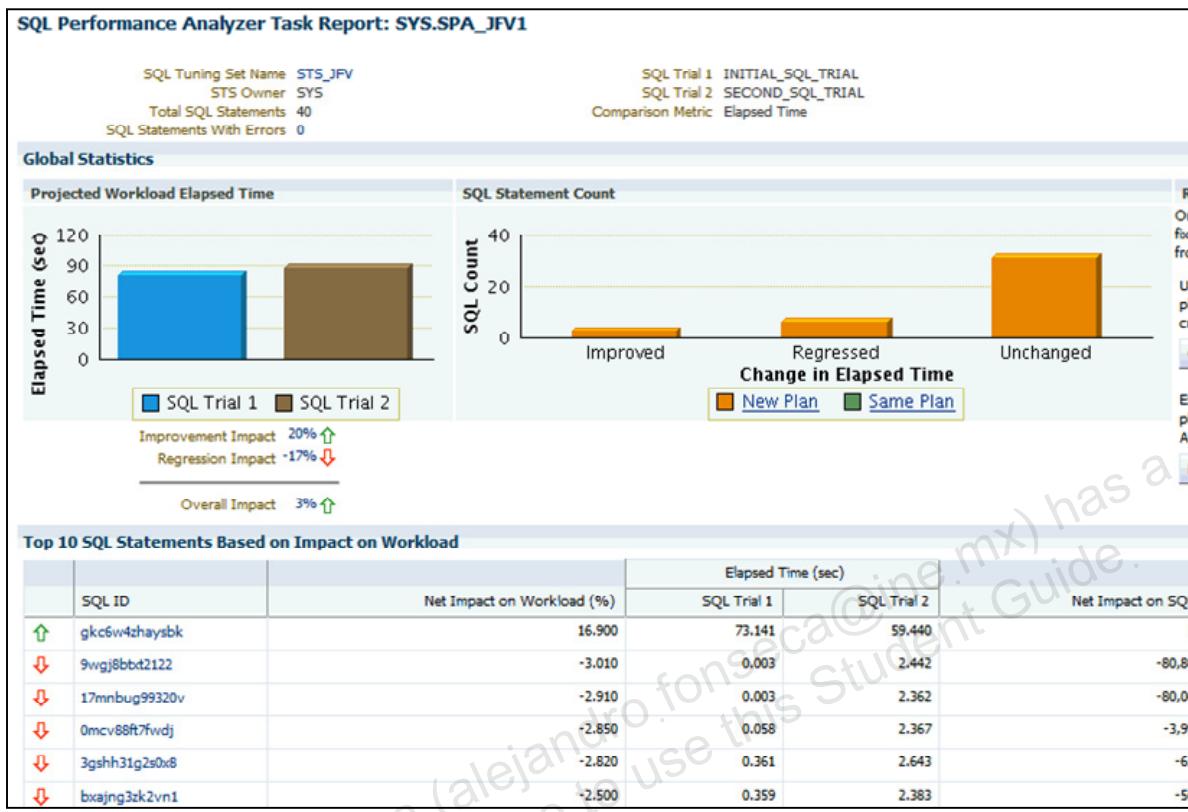
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A SQL Performance Analyzer task allows you to execute a specific SQL tuning set under changed environmental conditions. After you execute the task, you can assess the impact of these changes on the performance of the SQL tuning set. The Comparison Report is useful in assessing the impact of the changed environmental conditions on the performance of the specified SQL tuning set.

From this page, you can also perform the following:

- Create a trial to test the performance of a SQL tuning set under a specific environment. Click Create SQL Trial. Refer to the Guided Workflow page for detailed information about creating a replay trial.
- Run a trial comparison to compare the differences between the replay trials that have been created so far. A comparison report is generated for each replay trial run. Click Run SQL Trial Comparison. Refer to the Guided Workflow page for detailed information about running a replay trial comparison.
- Click the eyeglass icon in the Comparison Report column to view the trial comparison report for your task.

Comparison Report



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

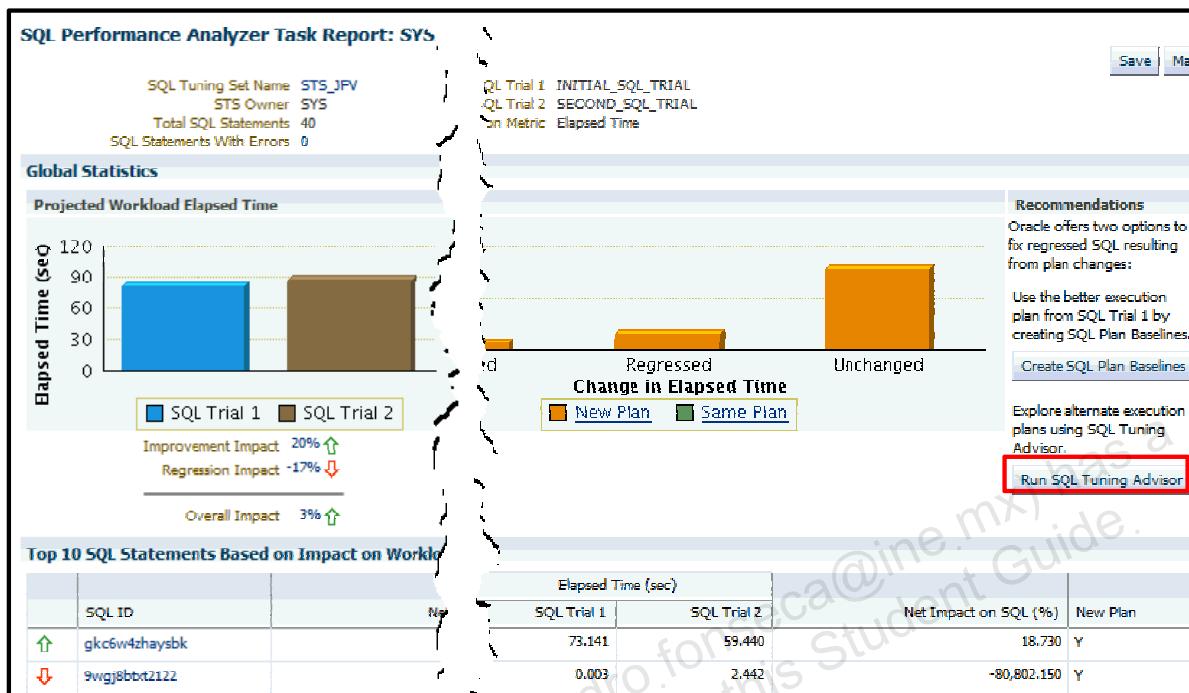
Use the SQL Performance Analyzer Task Result page to see the replay trial comparison report. The following general details are displayed:

- Task details, such as name, owner, and description of the task
- Name and owner of the SQL tuning set
- Total number of SQL statements and any SQL statements with errors. Click the SQL Statements With Errors link to access the Errors table.
- The replay trials being compared and the comparison metric being used

In addition to these details, you can view the following:

- Projected Workload [Comparison Metric]:** This chart shows the projected workload for each replay trial based on the comparison metric along with the improvement, regression, and overall impact. Click the impact links to drill down to the complete list of SQL statements in each category.
- SQL Statement Count:** This chart shows the number of SQL statements that have improved, regressed, or not changed performance based on the comparison metric. The colors of the bars indicate whether the plan changed between the two trial runs. Click the links or the data buckets to access the SQL Statement Count Details page, where you can see a list of SQL statements, and then click a SQL ID to access the SQL details.
- “Top 10 SQL Statements Based on Impact on Workload” table:** This table allows you to click a specific SQL ID to drill down to the corresponding SQL Details page.

Tuning Regressing Statements



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

From the SQL Performance Analyzer Task Result page, you can directly tune all regressing statements by invoking SQL Tuning Advisor. To do so, click the Run SQL Tuning Advisor button to access the Schedule SQL Tuning Task page, where you can specify the tuning task name and a schedule.

When you are finished, click OK. This creates a new tuning task that analyzes all regressing statements found by SQL Performance Analyzer.

Tuning Regressing Statements

SQL Performance Analyzer Task: SYS.SPA_JFV1

[View Latest Report](#) Page Refreshed **Nov 12, 2013 12:22:41 PM UTC** [Refresh](#)

The SQL Performance Analyzer Task is a container for experimental results of executing a specific SQL Tuning Set under changed environmental conditions and assessing the impact of environmental changes on STS execution performance.

▷ **SQL Tuning Set**

▽ **SQL Trial Comparisons**

Compare SQL Trials to assess change impact of environmental differences on SQL Tuning Set execution costs.

[Run SQL Trial Comparison](#)

Trial 1 Name	Trial 2 Name	Compare Metric	Created	Status	Comparison Report	SQL Tune Report
INITIAL_SQL_TRIAL	SECOND_SQL_TRIAL	Elapsed Time	11/12/13 10:57 AM	COMPLETED		

SQL Tuning Result Summary:SYS.TUNE_JFV1

Status	Completed
Started	11/12/2013 12:21:10
Completed	11/12/2013 12:23:17
Running Time (minutes)	2

Overall Task Statistics

Distinct SQL Examined 6 [Show all results](#)

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

After the SQL tuning task is created, you return to the SQL Performance Analyzer Task Results page, where you can see that you now have a SQL Tune report associated with your performance analysis in the Recommendations Section.

You can also access the SQL Tuning Results Summary page directly from the SQL Performance Analyzer Task page by clicking the eyeglass icon in the SQL Tune Report column of the Trial Comparisons section for your trial comparison. Click the “Show all results” button to see the Recommendations table that lists all recommendations for regressing statements.

SQL Tuning Recommendations

SQL Tuning Result Details: All Analyzed SQLs

Status	Completed	Tuning Set Owner	SYS
Started	11/12/2013 12:21:10	Tuning Set Name	STS_JFV
Completed	11/12/2013 12:23:17	Time Limit (seconds)	1800
Running Time (minutes)	2		

Recommendations

Only profiles that significantly improve SQL performance were implemented.

		View Recommendations	Implement All SQL Profiles							
Select	SQL Text	Parsing Schema	SQL ID	Cumulative DB Time Benefit(sec) ▾	Per-Execution % Benefit	Statistics	SQL Profile	Index	Restructure SQL	Alternative Plan
(1)	SELECT /* ORDERED INDEX(t1) USE_HASH(t1)...	APPS	amny7vvjss6pp	0.37	79		(79%) ✓			
(2)	SELECT /* ORDERED INDEX(t1) USE_HASH(t1)...	APPS	9wgj8bbxt2122	0.03	82		(82%) ✓ (71%) ✓			
(3)	SELECT /* ORDERED INDEX(t1) USE_HASH(t1)...	APPS	17mnbbug99320v	0.03	82		(82%) ✓ (71%) ✓			

Legend ✓ Recommended ⚡ Implemented

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Recommendations table lists all recommendations for regressing statements. The recommended methods for improving the performance are shown. This could be to gather statistics, implement a SQL Profile, create an index, restructure the SQL, or use an alternative plan. An alternative plan is a SQL baseline plan.

Preventing Regressions

SQL Performance Analyzer Task Report: SYS

SQL Tuning Set Name: STS_JFV
STS Owner: SYS
Total SQL Statements: 40
SQL Statements With Errors: 0

QL Trial 1: INITIAL_SQL_TRIAL
QL Trial 2: SECOND_SQL_TRIAL
On Metric: Elapsed Time

Global Statistics

Projected Workload Elapsed Time

Elapsed Time (sec)

SQL Trial 1 SQL Trial 2

Regressed Unchanged

Change in Elapsed Time

New Plan Same Plan

Recommendations

Oracle offers two options to fix regressed SQL resulting from plan changes:

- Use the better execution plan from SQL Trial 1 by creating SQL Plan Baselines.
- Create SQL Plan Baselines
- Explore alternate execution plans using SQL Tuning

SQL Performance Analyzer Task Report: SYS.SPA_JFV1

Create SQL Plan Baselines

SQL Plan Baselines enable the optimizer to avoid performance regressions by requiring new plans to be at least as good as the better plans found in SQL trial 1.

Regressed New Plan SQL Statements

SQL ID	Net Impact on Workload (%)	Elapsed Time		Net Impact on SQL (%)	% of Workload	
		INITIAL_SQL_TRIAL	SECOND_SQL_TRIAL		INITIAL_SQL_TRIAL	SECOND_SQL_TRIAL
9wgj8btxt2122	-3.010	0.003	2.442	-80,802.150		
17mnb99320v	-2.910	0.003	2.362	-80,065.220		
0mcv88ft7fwdj	-2.850	0.058	2.367	-3,998.740		

Cancel OK

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Instead of using SQL Tuning Advisor to tune your regressing statements, you can also prevent regressions by using the SQL plan baselines. You can do so from the SQL Performance Analyzer Task Result page by clicking the Create SQL Plan Baselines button.

Note: For more information about SQL plan baselines, refer to the lesson titled “SQL Performance Management.”

SQL Performance Analyzer: PL/SQL Example

1. Create a tuning task.

```
exec :tname:= dbms_sqlpa.create_analysis_task( -
    sqlset_name => 'MYSTS', -
    task_name => 'MYSPA');
```

2. Execute the task to collect the before-change data.

```
exec dbms_sqlpa.execute_analysis_task(task_name => :tname, -
    execution_type => 'TEST EXECUTE', -
    execution_name => 'before');
```

3. Generate the before-change report.

```
select dbms_sqlpa.report_analysis_task(
    task_name => :tname, -
    type=>'text',
    section=>'summary') FROM dual;
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The general example in the slide shows you how to use the DBMS_SQLPA package to invoke SQL Performance Analyzer to assess the SQL performance impact of some changes. You could easily adapt this example to run your own analysis.

1. Create the tuning task to run SQL Performance Analyzer.
2. Execute the task once to build the before-change performance data. With this call, you can specify various parameters, such as the following:
 - Set the execution_type parameter in either of the following ways: Set to EXPLAIN PLAN to generate explain plans for all SQL statements in the SQL workload. Set to TEST EXECUTE to execute all SQL statements in the SQL workload. The procedure executes only the query part of the DML statements to prevent side effects to the database or user data. When TEST EXECUTE is specified, the procedure generates execution plans and execution statistics.
 - Specify execution parameters by using the execution_params parameter that needs to be specified as dbms_advisor.arglist(name,value,...). The time_limit parameter specifies the global time limit to process all SQL statements in a SQL tuning set before timing out. The local_time_limit parameter specifies the time limit to process each SQL statement in a SQL tuning set before timing out.
3. Produce the before-change report (special settings for report: set long 100000, longchunksize 100000, and linesize 90).

SQL Performance Analyzer: PL/SQL Example

4. Execute the after-changes task.

```
exec dbms_sqlpa.execute_analysis_task(task_name => :tname, -  
    execution_type => 'TEST EXECUTE', -  
    execution_name => 'after');
```

5. Generate the after-changes report.

```
select dbms_sqlpa.report_analysis_task(task_name => :tname,  
    type=>'text', section=>'summary') FROM dual;
```

6. Compare the two executions.

```
exec dbms_sqlpa.execute_analysis_task(task_name => :tname, -  
    execution_type => 'COMPARE PERFORMANCE');
```

7. Generate the comparison report.

```
select dbms_sqlpa.report_analysis_task(task_name => :tname,  
    type=>'text', section=>'summary') FROM dual;
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

After you make the change to your test instance:

4. Execute the task again after making the changes.
5. Generate the after-changes report.
6. Compare the two executions using the following command as a pattern and substituting parameters to suit your needs:

```
BEGIN  
    DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(  
        task_name => :tname,  
        execution_type => 'compare performance',  
        execution_params => dbms_advisor.arglist(  
            'execution_name1', 'before',  
            'execution_name2', 'after',  
            'comparison_metric', 'buffer_gets'));  
END;
```

7. Generate the analysis report.

Note: For more information about the DBMS_SQLPA package, see the *Oracle Database PL/SQL Packages and Types Reference Guide*.

Tuning Regressed SQL Statements

You can also create a SQL tuning task to tune regressed SQL statements reported by the SQL Performance Analyzer by using the `DBMS_SQLTUNE.CREATE_TUNING_TASK` function:

```
BEGIN  
  DBMS_SQLTUNE.CREATE_TUNING_TASK(  
    spa_task_name => 'MYSPLA',  
    spa_compare_exec => 'MYCOMPEXEC');  
END;  
/
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

After reviewing the SQL Performance Analyzer report, you should tune any regressed SQL statements that are identified after comparing the SQL performance. If there are a large number of SQL statements that appear to have regressed, try to identify the root cause and make system-level changes to rectify the problem.

If only a few SQL statements have regressed, consider using the SQL Tuning Advisor to implement a point solution for them. Create a SQL tuning task for the SQL Performance Analyzer execution by using the `DBMS_SQLTUNE.CREATE_TUNING_TASK` function. The `CREATE_TUNING_TASK` function has the following parameters:

- `SPA_TASK_NAME`: Name of the SQL Performance Analyzer task
- `SPA_TASK_OWNER`: Owner of the specified SQL Performance Analyzer task. If unspecified, this parameter will default to the current user.
- `SPA_COMPARE_EXEC`: Execution name of the compare performance trial for the specified SQL Performance Analyzer task. If unspecified, this parameter defaults to the most recent execution of the `COMPARE PERFORMANCE` type for the given SQL Performance Analyzer task.

After tuning the regressed SQL statements, test your changes by using SQL Performance Analyzer. Run a new SQL trial on the test system, followed by a second comparison (between this new SQL trial and the first SQL trial) to validate your results. After SQL Performance Analyzer shows that performance has stabilized, you can implement the changes.

SQL Performance Analyzer: Data Dictionary Views

- DBA{USER}_ADVISOR_TASKS: Displays details about the analysis task
- DBA{USER}_ADVISOR_FINDINGS: Displays analysis findings
- DBA{USER}_ADVISOR_EXECUTIONS: Lists metadata information for task execution
- DBA{USER}_ADVISOR_SQLPLANS: Displays the list of SQL execution plans
- DBA{USER}_ADVISOR_SQLSTATS: Displays the list of SQL compilation and execution statistics



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The following views provide information about the SQL Performance Analyzer:

- DBA{USER}_ADVISOR_TASKS: Displays details about the advisor task created to perform an impact analysis of a system environment change
- DBA{USER}_ADVISOR_FINDINGS: Displays analysis findings. The advisor generates four types of findings: performance regression, symptoms, errors, and informative messages.
- DBA{USER}_ADVISOR_EXECUTIONS: Lists metadata information for a task execution. SQL Performance Analyzer creates a minimum of three executions to perform a change impact analysis on a SQL workload: one execution to collect performance data for the before-change version of the workload, the second execution to collect data for the after-change version of the workload, and a final execution to perform the actual analysis.
- DBA{USER}_ADVISOR_SQLPLANS: Displays the list of all SQL execution plans (or those owned by the current user)
- DBA{USER}_ADVISOR_SQLSTATS: Displays the list of SQL compilation and execution statistics (or those owned by the current user)

Quiz

The SQL Performance Analyzer shows the overall performance change that you can expect when changing environments.

- a. True
- b. False



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: b

The SQL Performance Analyzer will show the performance difference of individual SQL statements. You would use Database Replay to find the overall performance change.

Summary

In this lesson, you should have learned how to:

- Identify the benefits of using SQL Performance Analyzer
- Describe the SQL Performance Analyzer workflow phases
- Use SQL Performance Analyzer:
 - To ascertain performance gains following a database change
 - To test the impact of proposed changes

Practice 13: Using SQL Performance Analyzer

This practice covers using the SQL Performance Analyzer to evaluate the changes to a SQL tuning set when moved from an 11gR2 environment to a 12c environment.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2018, Oracle and/or its affiliates.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

14

SQL Performance Management

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Manage changes to optimizer statistics
- Capture SQL profiles
- Use the SQL Access Advisor
- Set up SQL Plan Management
- Set up various SQL Plan Management scenarios

Maintaining SQL Performance

Maintaining performance may require:

- Managing optimizer statistics
- Using SQL profiles
- Testing the data structure with the SQL Access Advisor
- Using SQL plan baselines



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Any number of factors that influence the optimizer can change over time. The challenge is to maintain the SQL performance levels in spite of the changes.

Optimizer statistics change for many reasons. Managing the changes to SQL performance despite the changes to statistics is the task of the DBA.

Some SQL statements on any system will stand out as high resource consumers. It is not always the same statements. The performance of these statements needs to be tuned, without having to change the code. SQL profiles provide the means to control the performance of these statements.

The SQL Access Advisor is a tool that can help the DBA choose data structure changes by executing a workload against the current structure and recommending changes that produce an overall benefit.

SQL plan baselines are the key objects that SQL Plan Management uses to prevent unverified change to SQL execution plans. When SQL Plan Management is active, there will not be drastic changes in performance even as the statistics change or as the database version changes. Until a new plan is verified to produce better performance than the current plan, it will not be considered by the optimizer. This in effect freezes the SQL plan.

Maintaining Optimizer Statistics

- Optimizer statistics change over time.
- Changes in statistics influence the optimizer.
- To maintain optimum execution plans:
 - Set statistic preferences
 - Defer statistics publishing



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

For the optimizer to choose the best execution plan for the current data set, it needs statistics that match the data set. Part of the job of the DBA is to keep the statistics current. The Optimizer Statistics Gathering automated maintenance task helps with this responsibility.

Sometimes the current statistics cause the performance to be worse with some object or SQL, or default statistic preferences do not collect enough data for the optimizer to make good choices. Certain tables, indexes, or schemas may need gathering preferences that are different from the rest of the database; perhaps the database should have preferences that are different from the defaults. Preferences can be set at a global level, and then modified for the database, schema, or individual objects.

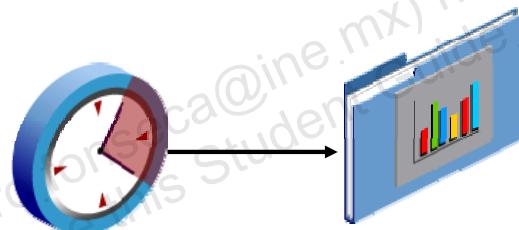
Automated Maintenance Tasks

Autotask maintenance process:

1. The maintenance window opens.
2. The Autotask background process schedules jobs.
3. The Scheduler initiates jobs.
4. Resource Manager limits the impact of Autotask jobs.

Default Autotask maintenance jobs:

- Gathering optimizer statistics
- Automatic segment advisor
- Automatic SQL advisor



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

By analyzing the information stored in the AWR, the Oracle Database server can identify the need to perform routine maintenance tasks, such as optimizer statistics refresh. The automated maintenance tasks infrastructure enables the Oracle Database server to automatically perform such operations. It uses the Scheduler to run such tasks in predefined maintenance windows.

By default, the maintenance windows start at 10 PM every night and last four hours on weekdays, and start at 6 AM and last for 20 hours on Saturday and Sunday. All attributes of the maintenance windows are customizable, including the start and end time, frequency, days of the week, and so on. Also, the impact of automated maintenance tasks on normal database operations can be limited by associating a Database Resource Manager resource plan to the maintenance window.

The examples of maintenance are as follows:

- Optimizer statistics are automatically refreshed by using the automatic maintenance task infrastructure. Statistics are gathered on the database and dictionary object on a priority basis.
Note: Statistics are not gathered on the operating system, or the fixed tables and views automatically.
- The Segment Advisor has default jobs, which run in the maintenance window.
- When creating a database with the DBCA, you can initiate regular database backups.

Statistic Gathering Options

Global Statistics Gathering Options

Database **orcl**

Statistics History

Retention Period (days) **31**

The number of days for which optimizer statistics history will be retained.

Gather Optimizer Statistics Default Options

Oracle recommends that you use the Gather Auto choice for the Gather Objects options when you use the Gather Optimizer Statistics process for Database and Schemas. If you choose not to use Gather Auto, the defaults for the other options are set here. Changing the options will impact the automated Optimizer Statistics Gathering task and user defined jobs.

Reset Defaults

Estimate Percentage Auto (Oracle recommended) 100% Percentage

Degree of Parallelism Table default Auto System default Degree

Granularity

Cursor Validation Auto (Oracle recommended) Immediate None

Cascade Auto (Oracle recommended) True False

Target Object Class (Auto Job) Auto (Oracle recommended) All Oracle

Stale Percentage

Incremental True False

Publish True False

Histograms

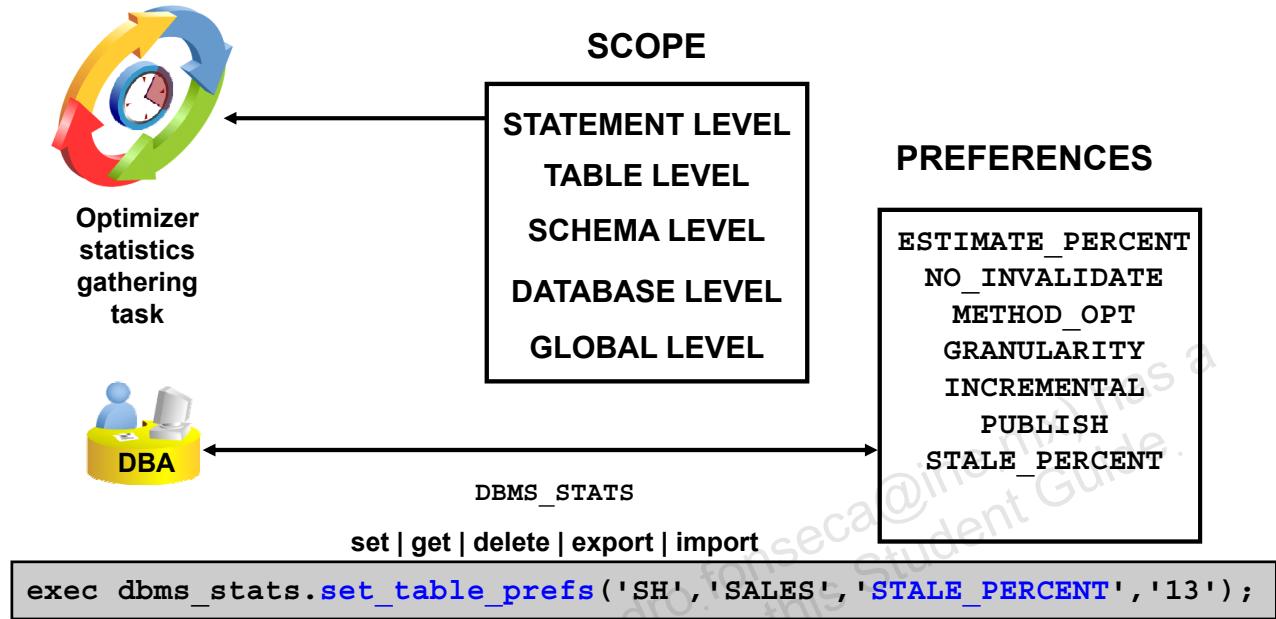
ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Global Statistics Gathering Options page in Enterprise Manager Cloud Control can be accessed from the Automated Maintenance Task Configuration page or by expanding the Performance menu and the SQL menu, selecting Optimizer Statistics, and clicking the Global Statistics Gathering Options link.

These options are applied globally. The options on this page do not override any previously set preferences at the schema or object level. The schema and object level preferences may be set on the Object Level Statistics Gathering Preferences page.

Setting Statistic Preferences



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The DBMS_STATS.GATHER_*_STATS procedures can be called at various levels to gather statistics for individual objects such as a table, or to gather statistics for an entire database. When the GATHER_*_STATS procedures are called, often several of the parameters are allowed to default. The supplied defaults work well for most of the objects in the database, but for some objects or schemas, the defaults need to be changed. Instead of running manual jobs for each of these objects, you can set the values, called preferences, for individual objects, schemas, and the database, or change the default values with a global-level command.

The preferences specify the parameters that are supplied to the gather procedures. The SET_*_PREFS procedures will create preference values for any object *not* owned by SYS or SYSTEM. The expected use is that the DBA will set any global preferences that should be databasewide. These will be applied for any parameter that is allowed to default. The SET_DATABASE_PREFS procedure iterates over all the tables and schemas in the database setting the specified preference. SET_SCHEMA_PREFS iterates over the tables in the specified schema. SET_TABLE_PREFS set the preference value for a single table. All of the object preferences, whether set at the database, schema, or table level, are held in a single table. Changing the preferences at the schema level overwrites the preferences previously set at the table level.

When the various gather procedures execute, they get the preferences set for each object. You can view the object-level preferences in the DBA_TAB_STAT_PREFS view. Any preferences that are not set at the object level will be set to the global-level preferences. You can see the global preferences by calling the DBMS_STATS.GET_PREFS procedure for each preference.

You can set, get, delete, export, and import those preferences at the table, schema, database, and global level. The preference values are expected to be set from global to table levels, applying the preferences to the smallest group last.

Global preferences and object preferences are independent. Changing the global preferences does not change database, schema, or table-level preferences. Object-level preferences always take precedence over global preferences when gathering statistics. However, changing a database preference will replace the schema and table preferences. Changing a schema preference will replace all the table preferences in that schema.

The preferences are as follows:

- PUBLISH is used to decide whether to publish the statistics to the dictionary or to store them in a private area.
- STALE_PERCENT is used to determine the threshold level at which an object is considered as having stale statistics. The value is a percentage of the rows modified since the last statistics gathering. The example changes the 10 percent default to 13 for SH.SALES only.
- INCREMENTAL is used to gather global statistics on partitioned tables in an incrementally way. When INCREMENTAL is set to TRUE, only the partitions that have been changed are scanned, instead of a full table scan.

Note: See *Oracle Database PL/SQL Packages and Types Reference* for limitations.

- METHOD_OPT determines the columns and histogram parameters used to gather column statistics.
- GRANULARITY determines the granularity of statistics to collect. (This is only pertinent if the table is partitioned.)
- NO_INVALIDATE is used to determine whether or not to invalidate cursors.
- ESTIMATE_PERCENT is used to determine the number of rows to sample to obtain good statistics. It is a percentage of the number of rows in the table.

For more details on the meaning and possible values of the preferences, see the DBMS_STATS section in *Oracle Database PL/SQL Packages and Types Reference*.

Preferences may be deleted with the DBMS_STATS.DELETE_*_PREFS procedures at the table, schema, and database levels. You can reset the global preferences to the recommended values with the DBMS_STATS.RESET_PARAM_DEFAULTS procedure.

Restoring Statistics

The DBMS_STATS package allows statistics to be restored as of a timestamp by using the RESTORE_*_STATS procedures.

- RESTORE_FIXED_OBJECTS_STATS
- RESTORE_SCHEMA_STATS
- RESTORE_SYSTEM_STATS
- RESTORE_TABLE_STATS

Manage Optimizer Statistics > Global Statistics Gathering Options

Global Statistics Gathering Options

Database orcl

Statistics History

Retention Period (days) The number of days for which optimizer statistics history will be retained.

Manage Optimizer Statistics

Database orcl

Optimizer Statistics are used by the query optimizer to improve the performance of your queries. Optimizer statistics can greatly improve the performance of your queries.

Operations

Gather Optimizer Statistics
Restore Optimizer Statistics
Lock Optimizer Statistics
Unlock Optimizer Statistics
Delete Optimizer Statistics

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

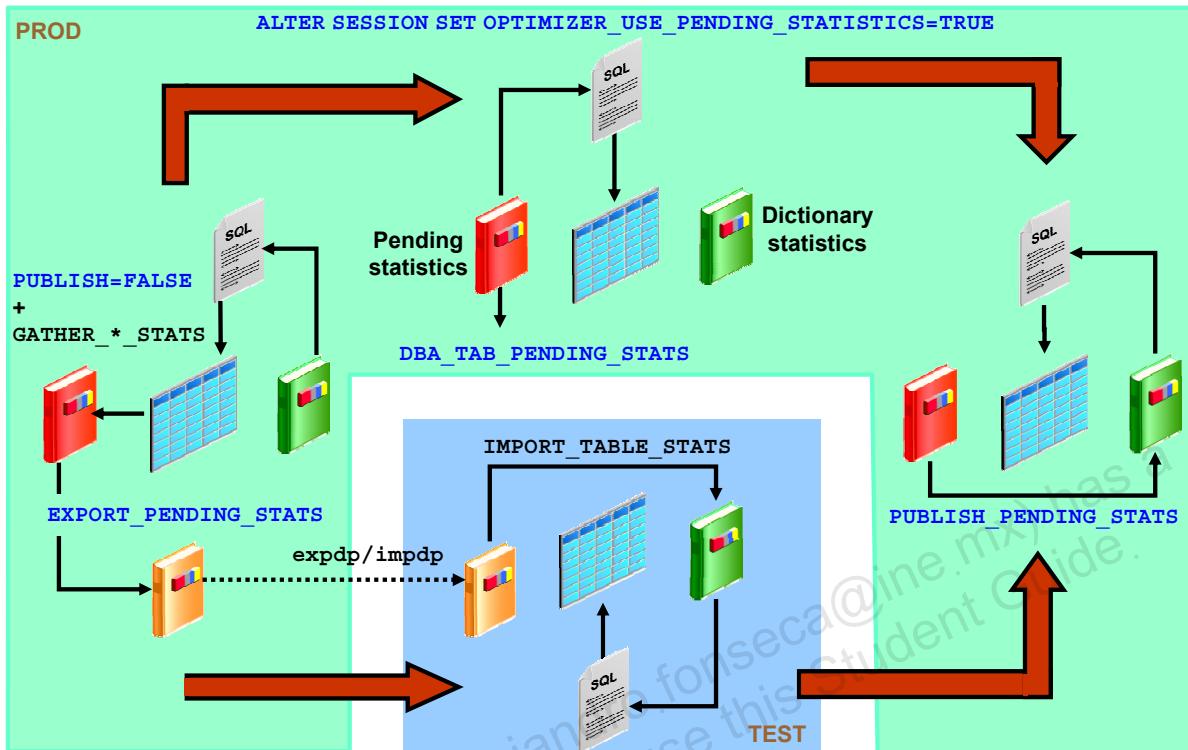
When statistics are gathered, the old statistics are automatically retained for the number of days specified by the retention period. The restore procedures will restore the statistics to the values as of a date, as long as the date is within the retention period.

The restore procedures will not restore user-defined statistics. If the statistics were locked on the specified date, the restore procedure will lock the statistics.

For usage details and parameters of the restore procedures, see *Oracle Database PL/SQL Packages and Types Reference*.

Note: User-defined statistics are set with the SET_*_STATS procedures. These procedures allow the owner or DBA to manually set the statistics to any value.

Deferred Statistics Publishing: Overview



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

By default, the statistics-gathering operation automatically stores the new statistics in the data dictionary each time it completes the iteration for one object (table, partition, subpartition, or index). The optimizer sees the new statistics as soon as they are written to the data dictionary, and these new statistics are called *current statistics*. Automatic publishing can be frustrating to the DBA, who is never sure of the impact of the new statistics. In addition, the statistics used by the optimizer can be inconsistent if, for example, table statistics are published before the statistics of its indexes, partitions, or subpartitions.

To avoid these potential issues, you can separate the gathering step from the publication step for optimizer statistics. There are two benefits in separating the two steps:

- Supports the statistics-gathering operation as an atomic transaction. The statistics of all tables and dependent objects (indexes, partitions, subpartitions) in a schema will be published at the same time. This means the optimizer will always have a consistent view of the statistics. If the gathering step fails during the gathering process, it will be able to resume from where it left off when it is restarted by using the `DBMS_STAT.RESUME_GATHER_STATS` procedure.
- Allows DBAs to validate the new statistics by running all or part of the workload using the newly gathered statistics on a test system and, when satisfied with the test results, to proceed to the publishing step to make them current in the production environment

When you specify the PUBLISH to FALSE gather option, gathered statistics are stored in the pending statistics tables instead of being current. These pending statistics are accessible from a number of views: {ALL|DBA|USER}_{TAB|COL|IND|TAB_HISTGRM}_PENDING_STATS.

The slide shows two paths for testing the statistics. The lower path shows exporting the pending statistics to a test system. The upper path shows enabling the use of pending statistics (usually in a session). Both paths continue to the production system and the publication of the pending statistics.

To test the pending statistics, you have two options:

- Transfer the pending statistics to your own statistics table by using the new DBMS_STAT.EXPORT_PENDING_STATS procedure, export your statistics table to a test system from where you can test the impact of the pending statistics, and then render the pending statistics current by using the DBMS_STAT.IMPORT_TABLE_STATS procedure.
- Enable session-pending statistics by changing your OPTIMIZER_USE_PENDING_STATISTICS session initialization parameter to TRUE. By default, this initialization parameter is set to FALSE. This means that in your session, you parse SQL statements by using the current optimizer statistics. By setting it to TRUE in your session, you switch to the pending statistics instead.

After you have tested the pending statistics and are satisfied with them, you can publish them as current in your production environment by using the DBMS_STAT.PUBLISH_PENDING_STATS procedure.

Note: For more information about the DBMS_STATS package, see the *Oracle Database PL/SQL Packages and Types Reference*.

Deferred Statistics Publishing: Example

```
exec dbms_stats.set_table_prefs('SH','CUSTOMERS','PUBLISH','false');
```

(1)

```
exec dbms_stats.gather_table_stats('SH','CUSTOMERS');
```

(2)

```
alter session set optimizer_use_pending_statistics = true;
```

(3)

Execute your workload from the same session.

(4)

```
exec dbms_stats.publish_pending_stats('SH','CUSTOMERS');
```

(5)

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

1. Use the SET_TABLE_PREFS procedure to set the PUBLISH option to FALSE. This prevents the next statistics gathering operation from automatically publishing statistics as current. According to the first statement, this is true for the SH.CUSTOMERS table only.
2. Gather statistics for the SH.CUSTOMERS table in the pending area of the dictionary.
3. Test the new set of pending statistics from your session by setting OPTIMIZER_USE_PENDING_STATISTICS to TRUE.
4. Issue queries against SH.CUSTOMERS.
5. If you are satisfied with the test results, use the PUBLISH_PENDING_STATS procedure to render the pending statistics for SH.CUSTOMERS current.

Note: To analyze the differences between the pending statistics and the current ones, you could export the pending statistics to your own statistics table, and then use the new DBMS_STAT.DIFF_TABLE_STATS function.

Automatic SQL Tuning: Overview

- Automatic SQL Tuning automates the entire SQL tuning process and replaces manual SQL tuning.
- Optimizer modes:
 - Normal mode
 - Tuning mode or Automatic Tuning Optimizer (ATO)
- SQL Tuning Advisor is used to access tuning mode.
- You should use tuning mode only for high-load SQL statements.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Automatic SQL Tuning is a capability of the query optimizer that automates the entire SQL tuning process. Using the enhanced query optimizer to tune SQL statements, the automatic process replaces manual SQL tuning, which is a complex, repetitive, and time-consuming function. The SQL Tuning Advisor exposes the features of Automatic SQL Tuning to the user. The enhanced query optimizer has two modes:

- In normal mode, the optimizer compiles SQL and generates an execution plan. The normal mode of the optimizer generates a reasonable execution plan for the vast majority of SQL statements. In normal mode, the optimizer operates with very strict time constraints, usually a fraction of a second, during which it must find a good execution plan.
- In tuning mode, the optimizer performs additional analysis to check whether the execution plan produced under normal mode can be further improved. The output of the query optimizer in tuning mode is not an execution plan but a series of actions, along with their rationale and expected benefit (for producing a significantly superior plan). When called under tuning mode, the optimizer is referred to as Automatic Tuning Optimizer (ATO). The tuning performed by ATO is called Automatic SQL Tuning.

Under tuning mode, the optimizer can take several minutes to tune a single statement. ATO is meant to be used for complex and high-load SQL statements that have a nontrivial impact on the entire system.

SQL Statement Profiling

- Statement statistics are key inputs to the optimizer.
- ATO verifies statement statistics such as:
 - Predicate selectivity
 - Optimizer settings (`FIRST_ROWS` versus `ALL_ROWS`)
- ATO uses:
 - Dynamic sampling
 - Partial execution of the statement
 - Past execution history statistics of the statement
- ATO builds a profile if statistics were generated:

```
exec :profile_name :=  
dbms_sqltune.accept_sql_profile( -  
task_name =>'my_sql_tuning_task');
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The main verification step during SQL profiling is the verification of the query optimizer's own estimates of cost, selectivity, and cardinality for the statement that is being tuned.

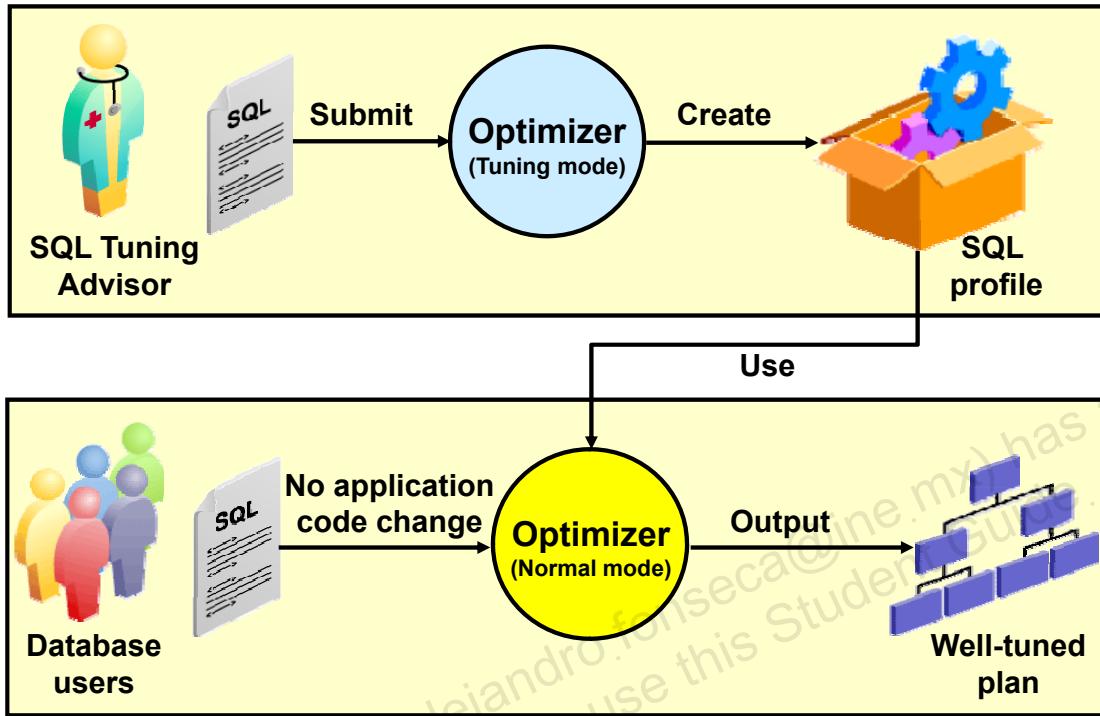
During SQL profiling, Automatic Tuning Optimizer (ATO) performs verification steps to validate its own estimates. The validation consists of taking a sample of data and applying appropriate predicates to the sample. The new estimate is compared to the regular estimate, and if the difference is large enough, a correction factor is applied. Another method of estimate validation involves the execution of a fragment of the SQL statement. The partial execution method is more efficient than the sampling method when the respective predicates provide efficient access paths. ATO picks the appropriate estimate validation method.

ATO also uses the past execution history of the SQL statement to determine correct settings. For example, if the execution history indicates that a SQL statement is only partially executed the majority of times, then ATO uses `FIRST_ROWS` optimization as opposed to `ALL_ROWS`.

ATO builds a SQL profile if it has generated auxiliary information either during statistics analysis or during SQL profiling. When a SQL profile is built, it generates a user recommendation to create a SQL profile.

In this mode, ATO can recommend the acceptance of the generated SQL profile in order to activate it.

Plan Tuning Flow and SQL Profile Creation



ORACLE

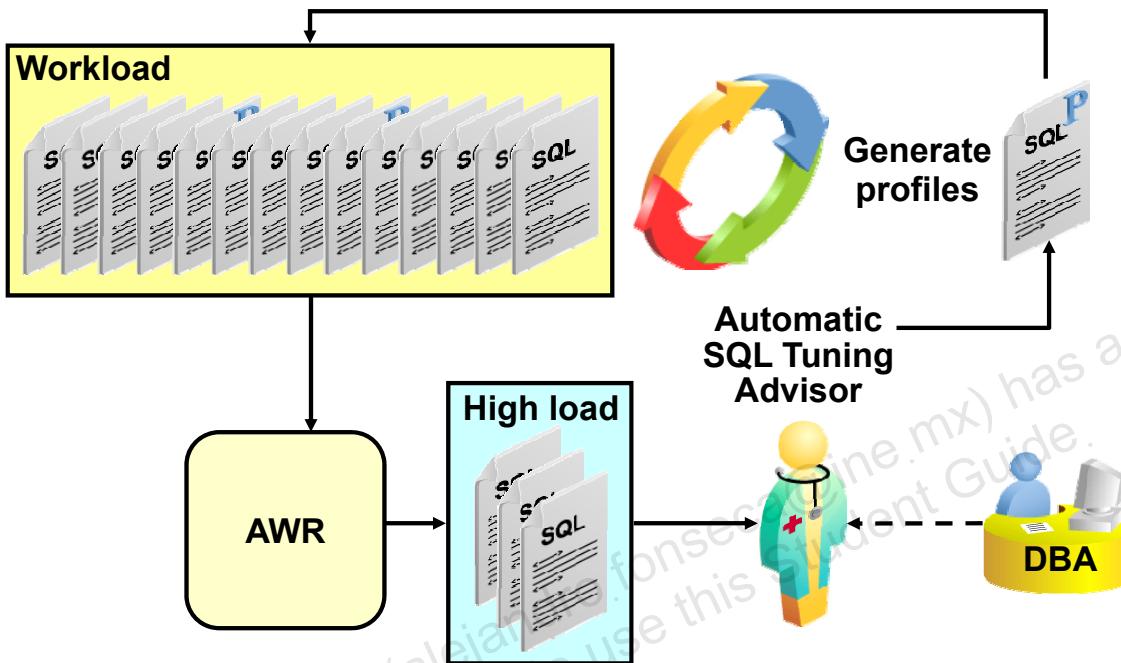
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A SQL profile is a collection of auxiliary information that is built during automatic tuning of a SQL statement. Thus, a SQL profile is to a SQL statement what statistics are to a table or index. After it is created, a SQL profile is used in conjunction with the existing statistics by the query optimizer, in normal mode, to produce a well-tuned plan for the corresponding SQL statement.

A SQL profile is stored persistently in the data dictionary. After the creation of a SQL profile, every time the corresponding SQL statement is compiled in normal mode, the query optimizer uses the SQL profile to produce a well-tuned plan.

The slide shows the process flow of the creation and use of a SQL profile. The process consists of two separate phases: SQL tuning phase and regular optimization phase. During the SQL tuning phase, you select a SQL statement for automatic tuning and run SQL Tuning Advisor by using either Database Control or the command-line interface. SQL Tuning Advisor invokes ATO to generate tuning recommendations, possibly with a SQL profile. If a SQL profile is built, you can accept it. When it is accepted, the SQL profile is stored in the data dictionary. In the next phase, when an end user issues the same SQL statement, the query optimizer (under normal mode) uses the SQL profile to build a well-tuned plan. The use of the SQL profile remains completely transparent to the end user and does not require changes to the application source code.

SQL Tuning Loop



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The auxiliary information contained in a SQL profile is stored in such a way that it stays relevant after database changes, such as addition or removal of indexes, growth in the size of tables, and periodic collection of database statistics. Therefore, when a profile is created, the corresponding plan is not frozen.

However, a SQL profile may not adapt to massive changes in the database or changes that have been accumulating over a long period of time. In such cases, a new SQL profile needs to be built to replace the old one.

For example, when a SQL profile becomes outdated, the performance of the corresponding SQL statement may become noticeably worse. In such a case, the corresponding SQL statement may start showing up as high-load or top SQL, thus becoming again a target for Automatic SQL Tuning. In such a situation, the Automatic SQL Tuning task again captures the statement as high-load SQL. If that happens, you can create a new profile for that statement.

Using SQL Profiles

- SQL profiles are:
 - Captured in categories
 - Enabled by category
- By default, all SQL profiles are enabled for all users.



```
EXEC DBMS_SQLTUNE.ALTER_SQL_PROFILE ( name=> :pname, -  
                                         attribute_name => 'CATEGORY', -  
                                         value           => 'DEV')
```

```
ALTER SESSION SET SQLTUNE_CATEGORY = 'DEV';
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The scope of a SQL profile can be controlled by the `CATEGORY` profile attribute. This attribute determines which user sessions can apply the profile. You can view the `CATEGORY` attribute for a SQL profile in the `CATEGORY` column of the `DBA_SQL_PROFILES` view. By default, all profiles are created in the `DEFAULT` category. This means that all user sessions where the `SQLTUNE_CATEGORY` initialization parameter is set to `DEFAULT` can use the profile. `DEFAULT` is the default setting.

By altering the category of a SQL profile, you can determine which sessions are affected by the creation of a profile. For example, by setting the category of a SQL profile to `DEV`, only those user sessions where the `SQLTUNE_CATEGORY` initialization parameter is set to `DEV` can use the profile. All other sessions do not have access to the SQL profile and execution plans for SQL statements are not affected by the SQL profile. This technique enables you to test a SQL profile in a restricted environment before making it available to other user sessions.

SQL Tuning Advisor: Overview

Comprehensive SQL tuning analysis and recommendations:

- Detect stale or missing statistics
- Tune the SQL plan (SQL profile)
- Add missing indexes
- Restructure SQL statements



SQL Tuning Advisor

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The SQL Tuning Advisor is the primary driver of the automated tuning process. It performs several types of analyses:

- **Statistics analysis:** This checks each query object for missing or stale statistics, and makes recommendations to gather relevant statistics.
- **SQL profiling:** The optimizer verifies its own estimates and collects auxiliary information to remove estimation errors. It builds a SQL profile using the auxiliary information and makes a recommendation to create it. When a SQL profile is created, it enables the query optimizer to generate a well-tuned plan.
- **Access path analysis:** New indexes are considered if they will significantly improve access to each table in the query. When appropriate, recommendations to create such objects are made.
- **SQL structure analysis:** SQL statements that use bad plans are identified and relevant suggestions are made to restructure them. The suggested changes can be syntactic as well as semantic.

The SQL Tuning Advisor considers each SQL statement included in the advisor task independently. Creating a new index may help a query, but may hurt the response time of DML. So a recommended index or other object should be checked with the SQL Access Advisor over a workload (a set of SQL statements) to determine whether there is a net gain in performance.

Using the SQL Tuning Advisor

- Use the SQL Tuning Advisor to analyze SQL statements and obtain performance recommendations.
- Sources for SQL Tuning Advisor to analyze:
 - **Top Activity:** Analyzes the top SQL statements currently active
 - **SQL tuning sets:** Analyzes a set of SQL statements you provide
 - **Historical SQL (AWR):** Analyzes SQL statements from statements collected by AWR snapshots



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The SQL Tuning Advisor runs automatically every night as the Automatic SQL Tuning Task. There may be times when a SQL statement needs immediate tuning action. You can use the SQL Tuning Advisor to analyze SQL statements and obtain performance recommendations at any time. Typically, you run this advisor as an ADDM performance-finding action.

Additionally, you can run the SQL Tuning Advisor when you want to analyze the top SQL statements that consume the most CPU time, I/O, and memory.

Even though you can submit multiple statements to be analyzed in a single task, each statement is analyzed independently. To obtain tuning recommendations that consider overall performance of a set of SQL, use the SQL Access Advisor.

The SQL Tuning Advisor can use enough resources on the production system to affect the normal performance. A SQL tuning set of the high load statements may be staged and exported to a test system and analyzed. The resulting SQL profiles can be returned to the production system by using the `DBMS_SQLTUNE.PACK_STGTAB_SQLPROF` procedure. The `DBMS_SQLTUNE` package contains procedures for creating the staging tables for both the STS and SQL profiles, and for packing and unpacking the staging tables in preparation for transferring the data.

SQL Tuning Advisor Options

Schedule SQL Tuning Advisor

Specify the following parameters to schedule a job to run the SQL Tuning Advisor.

* Name	<input type="text" value="SQL_TUNING_1384266192358"/>	Cancel	Submit
Description	<input type="text"/>		
* SQL Tuning Set	<input type="text"/>		
SQL Tuning Set Description			
SQL Statements Counts	0		
Overview The SQL Tuning Advisor analyzes individual SQL statements, and suggests indexes, SQL profiles, restructured SQL, and statistics that improve the performance of the SQL statements. The SQL Tuning Advisor operates on a collection of SQL. You can choose a SQL Tuning Set to run the advisor. If you do not have a SQL Tuning Set with the desired SQL for running the advisor, you can create a new one. You can click on one of the following sources, which will lead you to a data source where you can tune SQL statements using the SQL Tuning Advisor.			
Top Activity Historical SQL (AWR) SQL Tuning Sets			
SQL Statements <p>Scope</p> <p>Total Time Limit (minutes) <input type="text" value="30"/></p> <p>Scope of Analysis <input checked="" type="radio"/> Limited The analysis is done without SQL Profile recommendation and takes about 1 second per statement.</p> <p><input checked="" type="radio"/> Comprehensive This analysis includes SQL Profile recommendation, but may take a long time.</p> <p>Time Limit per Statement (minutes) <input type="text" value="5"/></p>			

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

On the Schedule SQL Tuning Advisor page, you can choose the SQL statements to include and change the automatic defaults for a tuning task. You can set the source of the SQL statements, and if you have been granted the ADVISOR system privilege, you can submit the task. Enterprise Manager then creates a tuning task for the SQL Tuning Advisor.

The SQL statement options allow you to choose one or more SQL statements from recent top activity lists, choose historical SQL stored in the AWR, or choose from a SQL tuning set that you have already created.

It is important to choose the appropriate scope for the tuning task. If you choose the Limited option, the SQL Tuning Advisor produces recommendations based on statistics check, access path analysis, and SQL structure analysis. The Limited option will not make a SQL profile recommendation. If you choose the Comprehensive option, the SQL Tuning Advisor produces all the recommendations that the Limited option produces, but it also invokes the optimizer under the SQL profiling mode to build a SQL profile. With the Comprehensive option, you can also specify a time limit for the tuning task, which by default is 30 minutes. After you select Run SQL Tuning Advisor, configure your tuning task by using the SQL Tuning Options page.

SQL Tuning Advisor Recommendations

SQL Tuning Result Details: All Analyzed SQLs

		Status	Completed	Tuning Set Owner	SYS	
		Started	11/12/2013 12:21:10	Tuning Set Name	STS_JFV	
		Completed	11/12/2013 12:23:17	Time Limit (seconds)	1800	
		Running Time (minutes)	2			
Recommendations						
Only profiles that significantly improve SQL performance were implemented.						
View Recommendations Implement All SQL Profiles						
Select	SQL Text	Parsing Schema	SQL ID	Cumulative DB Time Benefit(sec) ▾	Per-Execution % Benefit	Statistics
①	SELECT /* ORDERED INDEX(t1) USE_HASH(t1)...	APPS	amny7vwjss6pp	0.37	79	(79%) ✓
②	SELECT /* ORDERED INDEX(t1) USE_HASH(t1)...	APPS	bxajng3zk2vn1	0.33	78	(78%) ✓
③	SELECT /* ORDERED INDEX(t1) USE_HASH(t1)...	APPS	3gshh31g2s0x8	0.32	78	(78%) ✓
④	SELECT /* ORDERED INDEX(t1) USE_HASH(t1)...	APPS	0mcv88ft7fwdj	0.10	78	(78%) ✓
⑤	SELECT /* ORDERED INDEX(t1) USE_HASH(t1)...	APPS	9wgj8btxt2122	0.03	82	(82%) ✓ (71%) ✓
⑥	SELECT /* ORDERED INDEX(t1) USE_HASH(t1)...	APPS	17mnbug99320v	0.03	82	(82%) ✓ (71%) ✓

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The SQL tuning results for the task are displayed as soon as the task completes and can also be accessed later from the Advisors Home page. On the Advisors Home page, you can click View Result to select a SQL Tuning Advisor Task, or click the Name of the task to display a summary of recommendations. Click the “Show all results” button to review and implement individual recommendations. On the SQL Tuning Result Details: All Analyzed SQLs page, select the statement and click View Recommendation.

Alternative Execution Plans

SQL Tuning Advisor searches real-time and historical data for alternative execution plans for the statement.

```
2- Alternative Plan Finding
-----
Some alternative execution plans for this statement were found by searching
the system's real-time and historical performance data. The following table
lists these plans ranked by their average elapsed time. See section
"ALTERNATIVE PLANS SECTION" for detailed information on each plan.
id plan hash last seen elapsed (s) origin note
-----
1 1378942017 2009-02-05/23:12:08 0.000 Cursor Cache original plan
2 2842999589 2009-02-05/23:12:08 0.002 STS
Information
-----
The Original Plan appears to have the best performance, based on the elapsed
time per execution. 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 optimizer to pick it over any other choices in the future.
execute dbms_sqltune.create_sql_plan_baseline(task_name => 'TASK_XXXXX',
object_id => 2, task_owner => 'SYS', plan_hash => xxxxxxxxx);
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

While tuning a SQL statement, SQL Tuning Advisor searches real-time and historical performance data for alternative execution plans for the statement. If plans other than the original plan exist, then SQL Tuning Advisor reports an alternative plan finding.

SQL Tuning Advisor validates the alternative execution plans and notes any plans that are not reproducible. When reproducible alternative plans are found, you can create a SQL plan baseline to instruct the optimizer to choose these plans in the future. The example in the slide shows that SQL Tuning Advisor found two plans, one in the cursor cache and the other in a SQL tuning set. The plan in the cursor cache is the same as the original plan.

SQL Tuning Advisor recommends an alternative plan only if the elapsed time of the original plan is worse than alternative plans. In this case, SQL Tuning Advisor recommends that users create a SQL plan baseline on the plan with the best performance. In the example in the slide, the alternative plan did not perform as well as the original plan, so SQL Tuning Advisor did not recommend using the alternative plan.

The alternative plans section of the SQL Tuning Advisor output includes both the original and alternative plans and summarizes their performance. The most important statistic is elapsed time. The original plan used an index, whereas the alternative plan used a full table scan, increasing elapsed time by .002 seconds.

Plan 1

```
-----
Plan Origin :Cursor Cache
Plan Hash Value :1378942017
Executions :50
Elapsed Time :0.000 sec
CPU Time :0.000 sec
Buffer Gets :0
Disk Reads :0
Disk Writes :0
Notes:
```

1. Statistics shown are averaged over multiple executions.
2. The plan matches the original plan.

Id Operation Name
0 SELECT STATEMENT
1 SORT AGGREGATE
2 MERGE JOIN
3 INDEX FULL SCAN TEST1_INDEX
4 SORT JOIN
5 TABLE ACCESS FULL TEST

Plan 2

```
-----
Plan Origin :STS
Plan Hash Value :2842999589
Executions :10
Elapsed Time :0.002 sec
CPU Time :0.002 sec
Buffer Gets :3
Disk Reads :0
Disk Writes :0
Notes:
```

1. Statistics shown are averaged over multiple executions.

Id Operation Name
0 SELECT STATEMENT
1 SORT AGGREGATE
2 HASH JOIN
3 TABLE ACCESS FULL TEST
4 TABLE ACCESS FULL TEST1

To adopt an alternative plan regardless of whether SQL Tuning Advisor recommends it, call DBMS_SQLTUNE.CREATE_SQL_PLAN_BASELINE.

Quiz

A SQL profile is created by the SQL Tuning Advisor. It can be automatically applied or reviewed before it is implemented. Which of the following would the SQL Tuning Advisor use to find a better execution plan?

- a. Full executions of problem SQL statements
- b. Statistics from previous runs
- c. The optimizer in normal mode
- d. SQL baselines
- e. Creates additional indexes



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: b

The SQL Tuning Advisor uses partial executions of a problem SQL statement to find better access paths and statistics. The optimizer is running as Automatic Tuning Optimizer. SQL baselines are not used to create SQL profiles, though execution plans produced by the profile can be accepted as baselines. The SQL Tuning Advisor will make recommendations for indexes, but does not create them.

Using the SQL Access Advisor

The screenshot shows the Oracle SQL Access Advisor interface with three main sections:

- SQL Access Advisor: Workload Source**: Shows the workload source selected as "Current and Recent SQL Activity". A yellow callout box labeled "Specify the workload." points to this section.
- SQL Access Advisor: Recommendation Options**: Shows the "Access Structures to Recommend" set to "Indexes" and "Materialized Views". A yellow callout box labeled "Specify the scope." points to this section.
- SQL Access Advisor: Review**: Shows the task name as "SQLACCESS4245473", task description as "SQL Access Advisor", and scheduled start time as "Run Immediately". A yellow callout box labeled "Review the selected options." points to this section.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can use the SQL Access Advisor to tune your schema and improve your query performance. This advisor requires that you identify a SQL workload, which is a representative set of SQL statements that access the schema. You can select your workload from different sources including current and recent SQL activity, a SQL repository, or a user-defined workload such as from a development environment.

The SQL Access Advisor may make recommendations, such as creating indexes, materialized views, or partitioning, to improve your query performance for the given workload.

You can invoke the SQL Access Advisor by performing the following steps:

1. Expand the Performance menu, and then expand the SQL menu.
2. Click SQL Access Advisor to start the wizard.
3. The SQL Access Advisor: Initial Options page appears. Select the set of initial options and click continue.
4. The SQL Access Advisor: Workload Source page appears. Specify your workload source and click Next.
5. The SQL Access Advisor: Recommendation Options page appears as shown in the slide. Specify the access structures for the advisor to recommend: indexes, materialized views, partitions, or any combination. Specify Limited or Comprehensive mode. Limited mode runs faster by concentrating on highest-cost statements. Click Next.

6. The SQL Access Advisor: Schedule page appears. Accept the default of immediate execution, or schedule the execution for a later time. Click Next.
7. The SQL Access Advisor: Review page appears. Review the options you have selected and click Submit to start your job.

Results are posted on the Advisor Central page. The SQL Access Advisor recommendations are ordered by cost benefit. For example, a recommendation may consist of a SQL script with one or more CREATE INDEX statements, which you can implement by clicking Schedule Implementation.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

View Recommendations

Results for Task: SQLACCESS4766595

Task Name	SQLACCESS4766595	Started	Nov 12, 2013 2:46:42 PM UTC
Status	COMPLETED	Ended	Nov 12, 2013 2:47:15 PM UTC
Advisor Mode	COMPREHENSIVE	Running Time (seconds)	33
Scheduler Job	ADV_SQLACCESS4766595	Total Time Limit (minutes)	10000
Publish Point	1		

Summary **Recommendations** **SQL Statements** **Details**

Overall Workload Performance

Potential for Improvement

Workload I/O Cost

Cost Type	Value
Original Cost	67197
New Cost	14990

Query Execution Time Improvement

Improvement Factor	No Performance Improvement (%)	Potential Performance Improvement (%)
1x	~20	~10
2x	~10	~10
4x	~10	~40
6x	~10	~20
8x	~10	~10
10x	~10	~10
> 10x	~10	~10

Recommendations

Recommendations 11
Space Requirements (MB) 623.802
User Specified Space Adjustment Unlimited
Show Recommendation Action Counts

SQL Statements

SQL Statements 40
Statements remaining after filters were applied
Show Statement Counts

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The recommendations produced by SQL Access Advisor are summarized showing the difference in cost and the percentage of statements that may be improved and the amount of the estimated improvement.

View Recommendation Details

This chart and table list recommendations initially ordered by the largest cost improvement. Implementing the top recommendation will improve total performance the most.

Recommendations by Cost Improvement

ID	Cost Improvement
2	11000
4	9000
3	8500
8	7500
1	7000
9	5500
7	3500
10	2500
11	1500
6	500

Select Recommendations for Implementation

Include Retain Actions

[Recommendation Details](#) [Schedule Implementation](#) [Show SQL](#)

Previous | 1-10 of 11 | Next 1

[Select All](#) | [Select None](#)

Select	Implementation Status	ID	Actions	Action Types	Cost Improvement ▾	Cost Improvement (%)	Estimated Space Used (MB)	Affected SQL Statements
<input checked="" type="checkbox"/>	■	2	6	■	10490	20.09	64.626	8
<input checked="" type="checkbox"/>	■	4	5	■	8713	16.69	64.376	5
<input checked="" type="checkbox"/>	■	3	4	■	8213	15.73	64.346	7

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Recommendations tab displays the individual recommendations and the improvement attributed to each. You may view the SQL for each recommendation. You can also select individual recommendations to implement.

SQL Plan Management: Overview

- SQL Plan Management automatically controls SQL plan evolution.
- Optimizer automatically manages SQL plan baselines.
 - Only known and verified plans are used.
- Plan changes are automatically verified.
 - Only comparable or better plans are subsequently used.
- The plan baseline can be seeded for critical SQL with SQL tuning set (STS) from SQL Performance Analyzer.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Potential performance risk occurs when the SQL execution plan changes for a SQL statement. A SQL plan change can occur due to a variety of reasons such as optimizer version, optimizer statistics, optimizer parameters, schema definitions, system settings, and SQL profile creation.

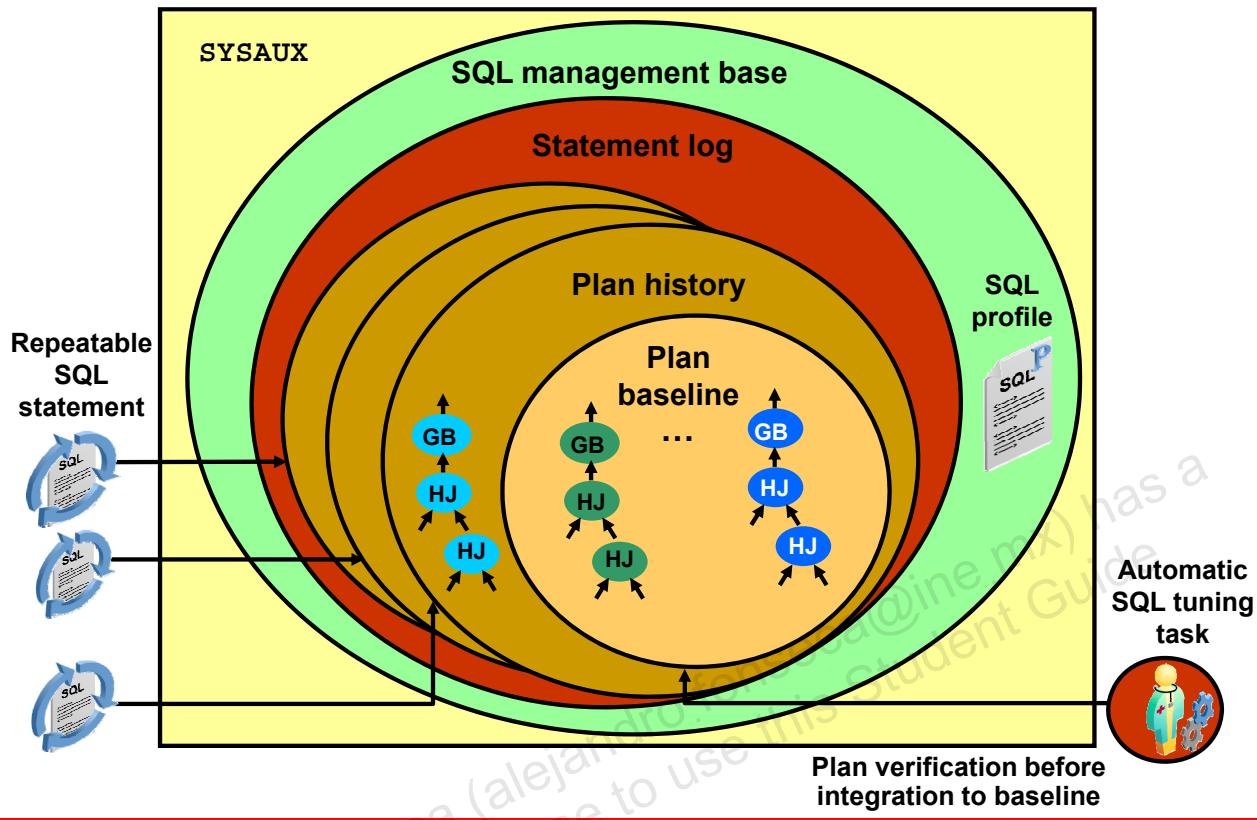
Various plan control techniques are available in Oracle Database to address performance regressions due to plan changes. The oldest is the use of hints in the SQL code to force a specific access path. Stored outlines allowed the hints to be stored separate from the code and modified. Both of these techniques focused on making the plan static. SQL profiles created by the SQL Tuning Advisor allow the optimizer to collect and store additional statistics that will guide the choice of a plan; if the plan becomes inefficient, the Tuning Advisor can be invoked to produce a new profile.

SQL Plan Management automatically controls SQL plan evolution by maintaining what is called “SQL plan baselines.” With this feature enabled, a newly generated SQL plan can integrate a SQL plan baseline only if it has been proven that doing so will not result in performance regression. So, during execution of a SQL statement, only a plan that is part of the corresponding SQL plan baseline can be used. As described later in this lesson, SQL plan baselines can be automatically loaded or can be seeded using SQL tuning sets. Various scenarios are covered later in this lesson.

The main benefit of the SQL Plan Management feature is performance stability of the system by avoiding plan regressions. In addition, it saves the DBA time that is often spent in identifying and analyzing SQL performance regressions and finding workable solutions.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

SQL Plan Baseline: Architecture



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

ORACLE

The SQL Plan Management feature introduces necessary infrastructure and services in support of plan maintenance and performance verification of new plans.

For SQL statements that are executed more than once, the optimizer maintains a history of plans for individual SQL statements. The optimizer recognizes a repeatable SQL statement by maintaining a statement log. A SQL statement is recognized as repeatable when it is parsed or executed again after it has been logged. After a SQL statement is recognized as repeatable, various plans generated by the optimizer are maintained as a plan history containing relevant information (such as SQL text, outline, bind variables, and compilation environment) that is used by the optimizer to reproduce an execution plan.

The DBA may also add plans to the SQL plan baseline by manual seeding a set of SQL statements.

A plan history contains different plans generated by the optimizer for a SQL statement over time. However, only some of the plans in the plan history may be accepted for use. For example, a new plan generated by the optimizer is not normally used until it has been confirmed that it does not cause performance regression. Plan verification is done by default as part of the Automatic SQL Tuning task running as an automated task in a maintenance window.

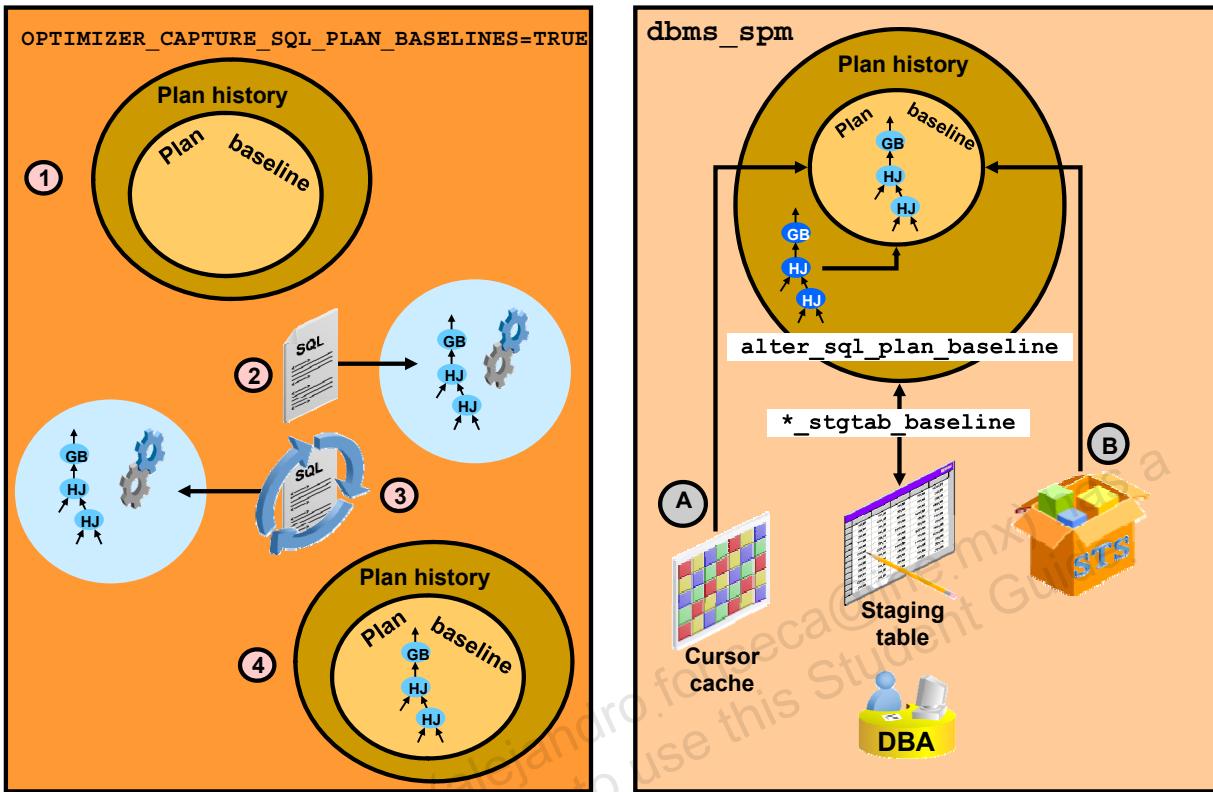
An Automatic SQL Tuning task targets only high-load SQL statements. For those statements, it automatically implements actions such as making a successfully verified plan an accepted plan. A set of acceptable plans constitutes a SQL plan baseline. The very first plan generated for a SQL statement is obviously acceptable for use; therefore, it forms the original plan baseline. Any new plans subsequently found by the optimizer are part of the plan history but not part of the plan baseline initially.

The statement log, plan history, and plan baselines are stored in the SQL management base (SMB), which also contains SQL profiles. The SMB is part of the database dictionary and is stored in the SYSAUX tablespace. The SMB has automatic space management (for example, periodic purging of unused plans). You can configure the SMB to change the plan retention policy and set space size limits.

Note: If the database instance is up but the SYSAUX tablespace is OFFLINE, the optimizer is unable to access SQL management objects. This can affect performance on some of the SQL workload.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

Loading SQL Plan Baselines



ORACLE

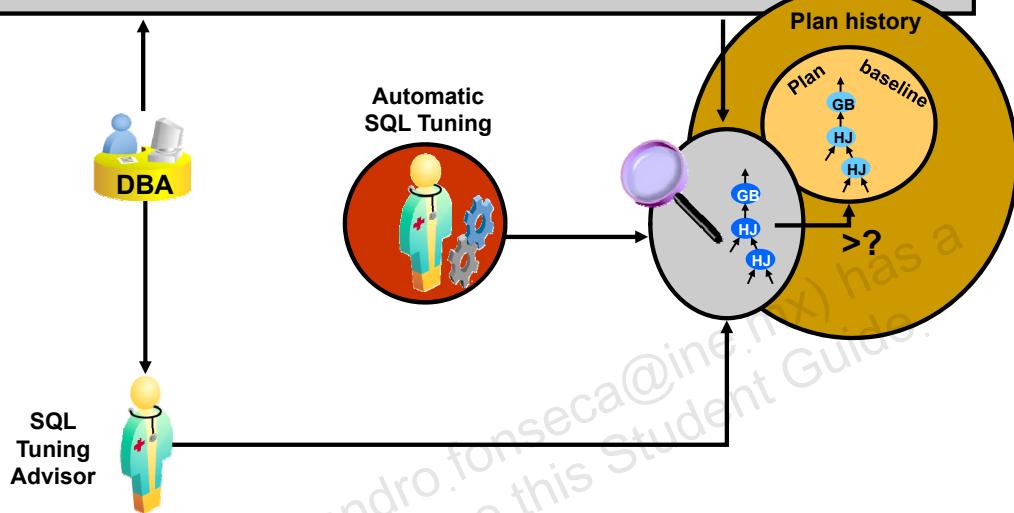
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

There are two ways to load SQL plan baselines.

- **On the fly capture:** Uses automatic plan capture by setting the `OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES` initialization parameter to `TRUE`. This parameter is set to `FALSE` by default. Setting it to `TRUE` turns on automatic recognition of repeatable SQL statements and automatic creation of plan history for such statements. This is illustrated in the graphic in left of the slide, where the first generated SQL plan is automatically integrated into the original SQL plan baseline when it becomes a repeating SQL statement.
- **Bulk loading:** Uses the `DBMS_SPM` package, which enables you to manually manage SQL plan baselines. With procedures from this package, you can load SQL plans into a SQL plan baseline directly from the cursor cache (A) using `LOAD_PLANS_FROM_CURSOR_CACHE` or from an existing SQL tuning set (STS) (B) using `LOAD_PLANS_FROM_SQLSET`. For a SQL statement to be loaded into a SQL plan baseline from an STS, the plan for the SQL statement needs to be stored in the STS. `DBMS_SPM.ALTER_SQL_PLAN_BASELINE` enables you to enable and disable a baseline plan and change other plan attributes. To move baselines between databases, use the `DBMS_SPM.*_STGTAB_BASELINE` procedures to create a staging table, and to export and import baseline plans from a staging table. The staging table can be moved between databases using the Data Pump Export and Import utilities.

Evolving SQL Plan Baselines

```
variable report clob
exec :report:=DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE(
sql_handle=>'SYS_SQL_593bc74fca8e6738');
Print report
```



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When the optimizer finds a new plan for a SQL statement, the plan is added to the plan history as a nonaccepted plan. The plan will not be accepted into the SQL plan baseline until it is verified for performance relative to the SQL plan baseline performance. Verification means a nonaccepted plan does not cause a performance regression (either manually or automatically). The verification of a nonaccepted plan consists of comparing its performance with the performance of one plan selected from the SQL plan baseline and ensuring that it delivers better performance.

There are three ways to evolve SQL plan baselines:

- **By using the DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE function:** As shown in the slide, the function returns a report that tells you which, if any, of the existing history plans were moved to the plan baseline. The example specifies a specific plan in the history to be tested. The function also allows verification without accepting the plan.
- **By running the SQL Tuning Advisor:** SQL plan baselines can be evolved by manually or automatically tuning SQL statements using the SQL Tuning Advisor. When the SQL Tuning Advisor finds a tuned plan and verifies its performance to be better than a plan chosen from the corresponding SQL plan baseline, it makes a recommendation to accept a SQL profile. When the SQL profile is accepted, the tuned plan is added to the corresponding SQL plan baseline.
- **By using the evolve autotask,** described in the following slide

Adaptive SQL Plan Management

- The new evolve auto task runs in the nightly maintenance window.
- It ranks all nonaccepted plans and runs the evolve process for them. Newly found plans are ranked the highest.
- Poor performing plans are not retried for 30 days, and then only if the statement is active.
- The new task is `SYS_AUTO_SPM_EVOLVE_TASK`.
- Information about the task is in `DBA_ADVISOR_TASKS`.
- Use `DBMS_SPM.REPORT_AUTO_EVOLVE_TASK` to view the results of the auto job.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Starting in Oracle Database 12c, the database can use adaptive SQL plan management (SPM). With adaptive SPM, DBAs no longer have to manually run the verification or evolve process for nonaccepted plans. They can go back days or weeks later and review what plans were evolved during each of the nightly maintenance window. When the SPM Evolve Advisor is enabled, it runs a verification or evolve process for all SQL statements that have nonaccepted plans during the nightly maintenance window. All the nonaccepted plans are ranked and the evolve process is run for them. The newly found plans are ranked the highest.

If the nonaccepted plan performs better than the existing accepted plan (or plans) in the SQL plan baseline, the plan is automatically accepted and becomes usable by the optimizer.

After the verification is complete, a persistent report is generated detailing how the nonaccepted plan performs when compared with the accepted plan performance.

The evolve task is now an AUTOTASK, scheduled by the same client that schedules the Automatic SQL Tuning Advisor. The DBA can schedule an evolve task at any time and use `DBMS_SPM.REPORT_AUTO_EVOLVE_TASK` to view the results.

Automatically Evolving SQL Plan Baseline

SPM Evolve Advisor

- The evolve function is now an advisory task.
- The new function allows DBAs to create a task, evolve plans in the task, and so on.

DBMS_SPM task-based functions

- DBMS_SPM.CREATE_EVOLVE_TASK
- DBMS_SPM.EXECUTE_EVOLVE_TASK
- DBMS_SPM.REPORT_EVOLVE_TASK
- DBMS_SPM.IMPLEMENT_EVOLVE_TASK

Persistent plan evolution report

- This allows the advisory task infrastructure to be implemented any time.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

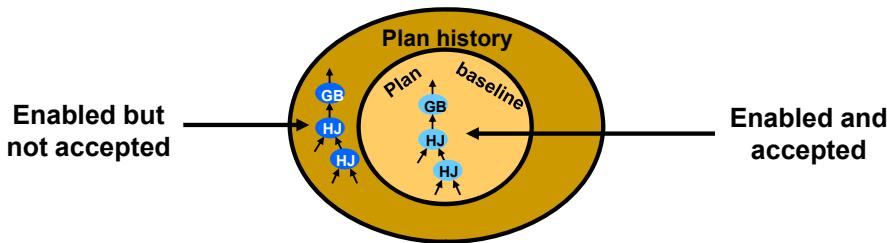
SPM Evolve Advisor is a SQL advisor that evolves plans. The advisor simplifies plan evolution by eliminating the manual chore of evolution.

New functions introduced in the DBMS_SPM package, such as

DBMS_SPM.CREATE_EVOLVE_TASK, DBMS_SPM.EXECUTE_EVOLVE_TASK, and so on, allow DBAs to create a task, evolve one or more plans in the task, fetch the report of the task, and implement the results of a task. The task can be executed multiple times, and the report of a given task can be fetched multiple times. SPM can schedule an evolve task as well as rerun an evolve task. There is also a function to implement the results of a task, which in this case would be to accept the currently nonaccepted plans.

Using the task infrastructure allows the report of the evolution to be persistently stored in the advisory framework repository. It also allows Enterprise Manager to schedule SQL plan baselines evolution and fetch reports on demand. The new function supports HTML and XML reports in addition to TEXT. In Oracle Database 12c, the task-based functions in DBMS_SPM retain the time_limit specification as a task parameter. When ACCEPT_PLANS is true (default), SQL plan management automatically accepts all plans recommended by the task. When set to false, the task verifies the plans and generates a report of its findings, but does not evolve the plans. Users can view the report by using the appropriate function, and then execute the implement function to accept the successful plans.

Important Baseline SQL Plan Attributes



```
SELECT signature, sql_handle, sql_text, plan_name, origin, enabled,
       accepted, fixed, autopurge
  FROM dba_sql_plan_baselines;
```

SIGNATURE	SQL_HANDLE	SQL_TEXT	PLAN_NAME	ORIGIN	ENA	ACC	FIX	AUT
8.062E+18	SYS_SQL_6fe2	select...	SYS_SQL_PLAN_1ea	AUTO-CAPTURE	YES	NO	NO	YES
8.062E+18	SYS_SQL_6fe2	select...	SYS_SQL_PLAN_4be	AUTO-CAPTURE	YES	YES	NO	YES
..								

```
exec :cnt := dbms_spm.alter_sql_plan_baseline(
      sql_handle => 'SYS_SQL_37e0168b0...3efe',
      plan_name   => 'SYS_SQL_PLAN_8dfc352f359901ea',
      attribute_name => 'ENABLED', attribute_value => 'NO');
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When a plan enters the plan history, it is associated with a number of important attributes:

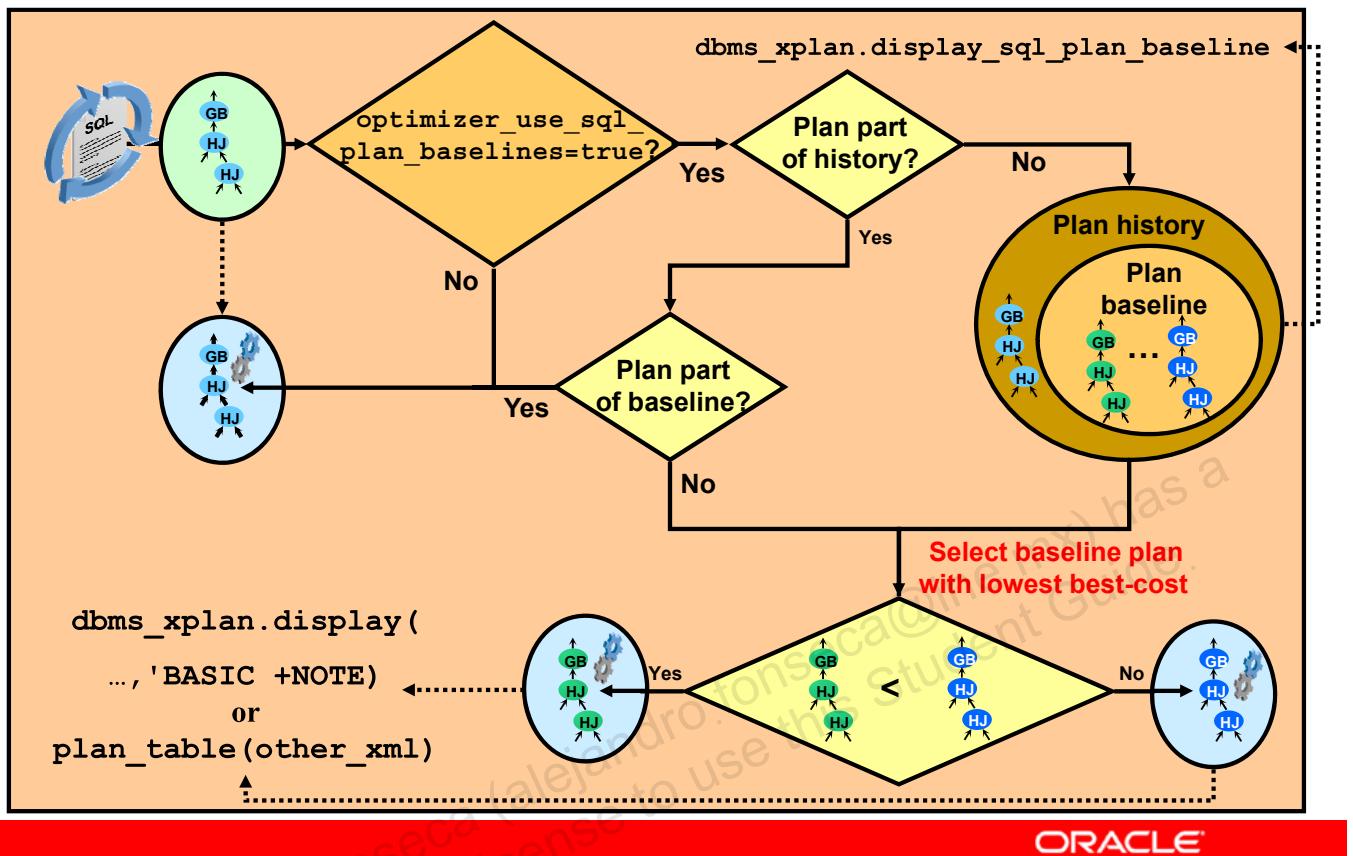
- SIGNATURE, SQL_HANDLE, SQL_TEXT, and PLAN_NAME are important identifiers for search operations.
- ORIGIN allows you to determine whether the plan was automatically captured (AUTOCAPTURE), manually evolved (MANUAL-LOAD), automatically evolved by SQL Tuning Advisor (MANUAL-SQLTUNE), or automatically evolved by Automatic SQL Tuning (AUTO-SQLTUNE).
- ENABLED and ACCEPTED: Both of the ENABLED and ACCEPTED attributes must be set to YES or the plan is not considered by the optimizer. The ENABLED attribute means that the plan is enabled for use by the optimizer. The ACCEPTED attribute means that the plan was validated as a good plan, either automatically by the system or manually when the user changes it to ACCEPTED. When a plan status changes to ACCEPTED, it will continue to be ACCEPTED until DBMS_SPM.ALTER_SQL_PLAN_BASELINE() is used to change its status. An ACCEPTED plan can be temporarily disabled by removing the ENABLED setting.
- FIXED means that the optimizer considers only those plans and not other plans. For example, if you have 10 baseline plans and three of them are marked FIXED, the optimizer uses only the best plan from these three, ignoring all the others. A SQL plan baseline is said to be FIXED if it contains at least one enabled fixed plan. If new plans are added to a fixed SQL plan baseline, these new plans cannot be used until they are manually declared as FIXED.

You can look at each plan's attributes by using the DBA_SQL_PLAN_BASELINES view as shown in the slide. You can then use the DBMS_SPM.ALTER_SQL_PLAN_BASELINE function to change some of them. You can also remove plans or a complete plan history by using the DBMS_SPM.DROP_SQL_PLAN_BASELINE function. The example shown in the slide changes the ENABLED attribute of SYS_SQL_PLAN_8DFC352F359901EA to NO.

Note: The DBA_SQL_PLAN_BASELINES view contains additional attributes that enable you to determine when each plan was last used and whether a plan should be automatically purged.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

SQL Plan Selection



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

If you are using automatic plan capture, the first time that a SQL statement is recognized as repeatable, its best-cost plan is added to the corresponding SQL plan baseline. That plan is then used to execute the statement.

The optimizer uses a comparative plan selection policy when a plan baseline exists for a SQL statement and the initialization parameter `OPTIMIZER_USE_SQL_PLAN_BASELINES` is set to `TRUE` (default value). Each time a SQL statement is compiled, the optimizer first uses the traditional cost-based search method to build a best-cost plan. Then it tries to find a matching plan in the SQL plan baseline. If a match is found, it proceeds as usual. If no match is found, it first adds the new plan to the plan history, then calculates the cost of each accepted plan in the SQL plan baseline, and picks the one with the lowest cost. The accepted plans are reproduced using the outline that is stored with each of them. So the effect of having a SQL plan baseline for a SQL statement is that the optimizer always selects one of the accepted plans in that SQL plan baseline.

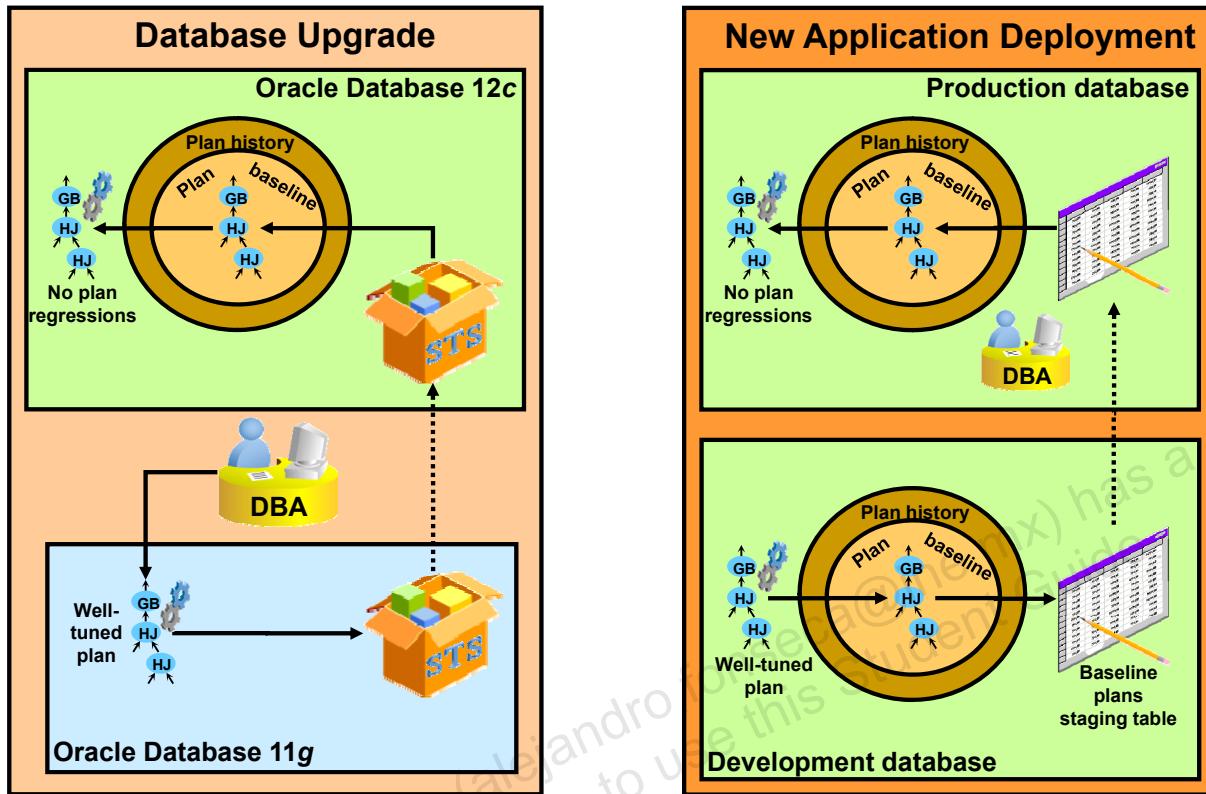
With SQL Plan Management, the optimizer can produce a plan that could be either a best-cost plan or a baseline plan. This information is dumped in the `OTHER_XML` column of the `PLAN_TABLE` upon execution of `EXPLAIN PLAN`. However, the optimizer can only use an accepted and enabled baseline plan.

In addition, you can use the DBMS_XPLAIN.DISPLAY_SQL_PLAN_BASELINE function to display one or more execution plans for the specified SQL_HANDLE of a plan baseline. If PLAN_NAME is also specified, the corresponding execution plan is displayed.

Note: The Stored Outline feature is deprecated. To preserve backward compatibility, if a stored outline for a SQL statement is active for the user session, the statement is compiled using the stored outline. In addition, a plan generated by the optimizer using a stored outline is not stored in the SMB even if automatic plan capture has been enabled for the session.

Although there is no explicit migrate procedure for stored outlines, they can be migrated to SQL plan baselines by using the LOAD_PLAN_FROM_CURSOR_CACHE or LOAD_PLAN_FROM_SQLSET procedures from the DBMS_SPM package. When the migration is complete, you should disable or drop the original stored outline.

Possible SQL Plan Manageability Scenarios



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

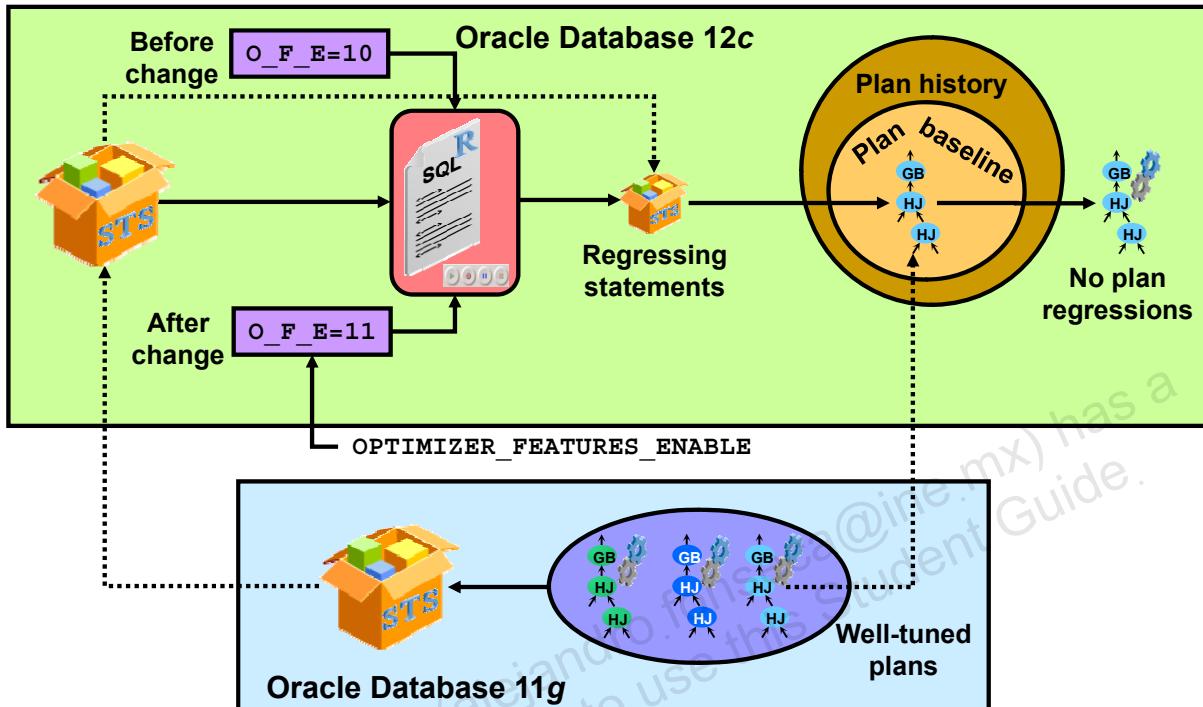
Database upgrade: Bulk SQL plan loading is especially useful when the system is being upgraded from an earlier version. For this, you can capture plans for a SQL workload into a SQL tuning set (STS) before the upgrade, and then load these plans from the STS into the SQL plan baseline immediately after the upgrade. This strategy can minimize plan regressions resulting from the use of the new optimizer version.

New application deployment: The deployment of a new application module means the introduction of new SQL statements into the system. The software vendor can ship the application software along with the appropriate SQL plan baselines for the new SQL being introduced. Because of the plan baselines, the new SQL statements will initially run with the plans that are known to give good performance under a standard test configuration. However, if the customer system configuration is very different from the test configuration, the plan baselines can be evolved over time to produce better performance.

In both the scenarios, you can use the automatic SQL plan capture after manual loading to make sure that only the better plans will be used for your applications in the future.

Note: In all scenarios in this lesson, assume that `OPTIMIZER_USE_SQL_PLAN_BASELINES` is set to `TRUE`.

SQL Performance Analyzer and SQL Plan Baseline Scenario



ORACLE

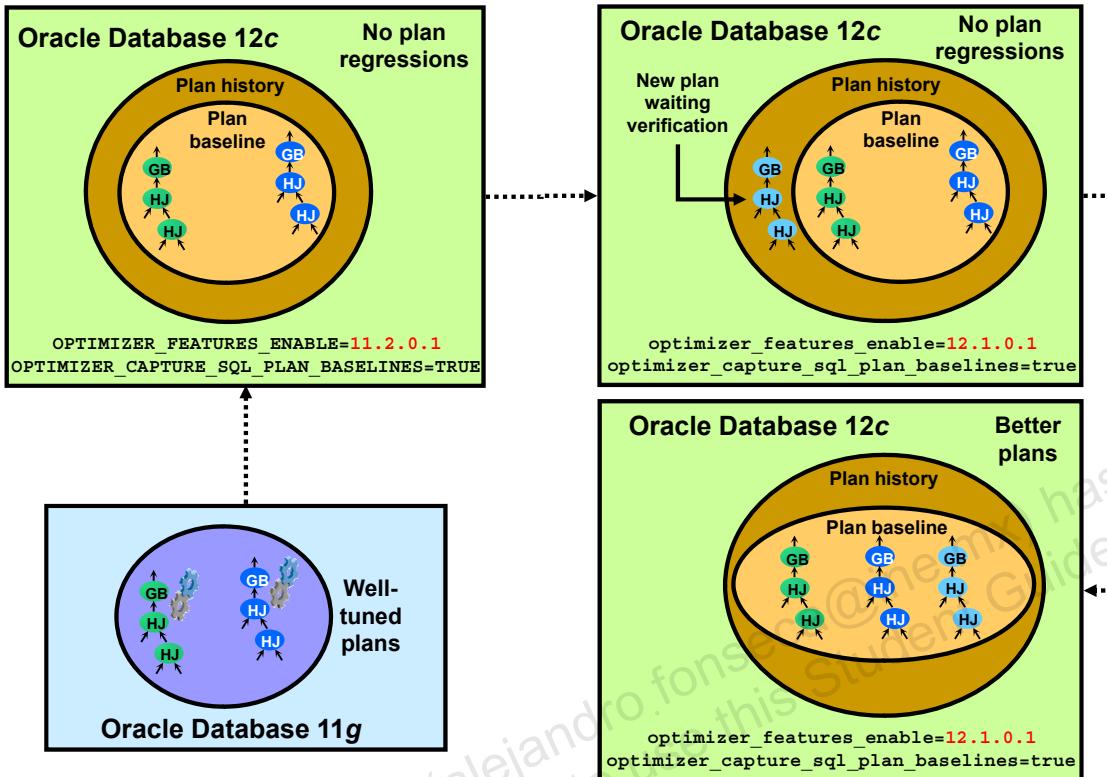
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A variation of the first method described in the previous slide is through the use of SQL Performance Analyzer. You can capture pre-Oracle Database 12c plans in an STS and import them into Oracle Database 12c. Then set the initialization parameter `OPTIMIZER_FEATURES_ENABLE` to 11g to make the optimizer behave as if this were an 11g Oracle database. Next run SQL Performance Analyzer for the STS. When that is complete, set the initialization parameter `OPTIMIZER_FEATURES_ENABLE` back to 12c and rerun SQL Performance Analyzer for the STS.

SQL Performance Analyzer produces a report that lists a SQL statement whose plan has regressed from 11g to 12c. For those SQL statements that are shown by SQL Performance Analyzer to incur performance regression due to the new optimizer version, you can capture their plans using an STS and then load them into the SMB.

This method represents the best form of the plan-seeding process because it helps prevent performance regressions while preserving performance improvements upon database upgrade.

Loading a SQL Plan Baseline Automatically



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Loading a SQL Plan Baseline Automatically: Scenario

Another upgrade scenario involves using the automatic SQL plan capture mechanism. In this case, set the initialization parameter `OPTIMIZER_FEATURES_ENABLE` (OFE) to the pre-Oracle Database 12c version value for an initial period of time such as a quarter, and execute your workload after upgrade by using the automatic SQL plan capture.

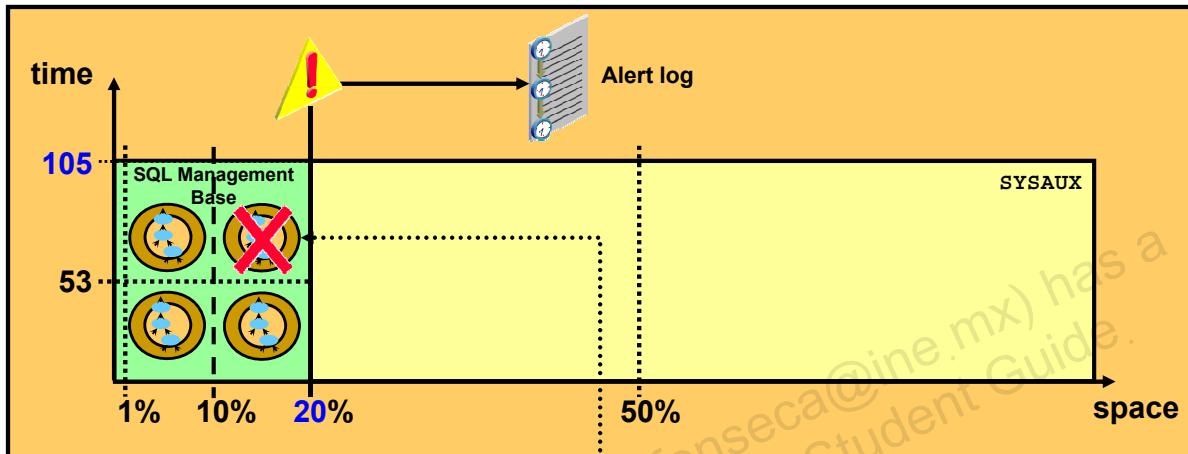
During this initial time period, because of the OFE parameter setting, the optimizer is able to reproduce pre-Oracle Database 12c plans for a majority of the SQL statements. Because automatic SQL plan capture is also enabled during this period, the pre-Oracle Database 12c plans produced by the optimizer are captured as SQL plan baselines.

When the initial time period ends, you can remove the setting of OFE to take advantage of the new optimizer version while incurring minimal or no plan regressions due to the plan baselines. Regressed plans will use the previous optimizer version; nonregressed statements will benefit from the new optimizer version.

Purging SQL Management Base Policy

```
SQL> exec dbms_spm.configure('SPACE_BUDGET_PERCENT',20);
SQL> exec dbms_spm.configure('PLAN_RETENTION_WEEKS',105);
```

DBA_SQL_MANAGEMENT_CONFIG



```
SQL> exec :cnt := dbms_spm.drop_sql_plan_baseline('SYS_SQL_37e0168b04e73ef');
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The space occupied by the SQL management base (SMB) is checked weekly against a defined limit. A limit based on the percentage size of the SYSAUX tablespace is defined. By default, the space budget limit for the SMB is set to 10 percent of the SYSAUX size. However, you can configure SMB and change the space budget to a value between 1 percent and 50 percent by using the DBMS_SPM.CONFIGURE procedure.

If SMB space exceeds the defined percent limit, warnings are written to the alert log. Warnings are generated weekly until the SMB space limit is increased, the size of SYSAUX is increased, or the size of SMB is decreased by purging some of the SQL management objects (such as SQL plan baselines or SQL profiles).

The space management of SQL plan baselines is done proactively using a weekly purging task. The task runs as an automated task in the maintenance window. Any plan that has not been used for more than 53 weeks is purged. However, you can configure SMB and change the unused plan retention period to a value between 5 weeks and 523 weeks (a little more than 10 years). To do so, use the DBMS_SPM.CONFIGURE procedure.

You can look at the current configuration settings for the SMB by examining the DBA_SQL_MANAGEMENT_CONFIG view. In addition, you can manually purge the SMB by using the DBMS_SPM.DROP_SQL_PLAN_BASELINE function (as shown in the example in the slide).

Enterprise Manager and SQL Plan Baselines

The screenshot shows the SQL Plan Control interface in Oracle Enterprise Manager. At the top, there are tabs for SQL Profile, SQL Patch, and SQL Plan Baseline, with SQL Plan Baseline selected. A message states: "A SQL Plan Baseline is an execution plan deemed to have acceptable performance for a given SQL statement." Below this is a "Settings" section with options: Capture SQL Plan Baselines (FALSE), Use SQL Plan Baselines (TRUE), and Plan Retention(Weeks) set to 53 with a "Configure" button. To the right is a "Jobs for SQL Plan Baselines" section with tabs for Pending (selected) and Completed, showing a job named BASELINE_JFV1 SPM_1384156928842. Below these are "Search" and "Actions" sections. The "Search" section includes a "SQL Text" input field, a "Go" button, and a note about case sensitivity. The "Actions" section includes buttons for Enable, Disable, Drop, Evolve, Copy To A Database, Pack, Fixed - Yes, Load, and Unpack. The main area displays a grid of SQL plan baselines:

Select	Name	SQL Text	Enabled	Accepted	Reproduced	Fixed	Auto Purge	Origin	Created	Last Modified
<input type="checkbox"/>	SQL_PLAN_djzmknc190hga71a8a7c	SELECT /* ORDERED INDEX(t1) USE_HASH(t1) */'B...	YES	YES	YES	NO	YES	MANUAL-LOAD	Nov 13, 2013 7:16:49 AM	Nov 13, 2013 7:16:49 AM
<input type="checkbox"/>	SQL_PLAN_bgq2tbchqsdd3a71a8a7c	SELECT /* ORDERED INDEX(t1)	YES	YES	YES	NO	YES	MANUAL-LOAD	Nov 13, 2013 7:16:48	Nov 13, 2013 7:16:48 AM

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Use the SQL Plan Management page to manage SQL profiles, SQL patches, and SQL plan baselines from one location rather than from separate locations in Enterprise Manager. You can also enable, disable, drop, pack, unpack, load, and evolve selected baselines.

From this page, you can also configure the various SQL plan baseline settings.

To navigate to this page, expand the Performance menu, expand the SQL menu, and then click the SQL Plan Control link.

Quiz

When `OPTIMIZER_USE_SQL_PLAN_BASELINES` is set to TRUE:

- a. The optimizer does not develop an execution plan; it uses an accepted plan in the baselines
- b. The optimizer compares the plan it develops with accepted plans in the baselines
- c. The optimizer compares the plan it develops with enabled plans in the baselines
- d. The optimizer does not develop an execution plan; it uses enabled plans in the baselines
- e. The optimizer develops plans and adds them to the baselines as verified



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: b

The optimizer always develops an execution plan. Then it compares the plan to accepted plans in the SQL baseline. If an accepted baseline exists, the baseline is used. If the plan developed by the optimizer is different, it is stored in the plan history, but it is not part of the baseline until it is verified and marked as accepted.

Summary

In this lesson, you should have learned how to:

- Manage change to optimizer statistics
- Capture SQL profiles
- Use the SQL Access Advisor
- Set up SQL Plan Management
- Set up various SQL Plan Management scenarios

Practice 14: SQL Performance Management

This practice covers:

- Seeding SQL Plan Baselines from the SQL Performance Analyzer
- Using SQL Plan Management to capture and implement plans from various sources

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

ORACLE

15

Using Database Replay

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Identify the benefits of using Database Replay
- List the steps involved in Database Replay
- Use Enterprise Manager to record and replay workloads

Using Database Replay

- Capture a workload in production:
 - Capture a full production workload with real load and concurrency.
 - Move the captured workload to the test system.
- Replay the workload in a test system:
 - Make the desired changes in the test system.
 - Replay the workload with production load and concurrency.
 - Honor the commit ordering.
- Analyze and report:
 - Errors
 - Data divergence
 - Performance divergence



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

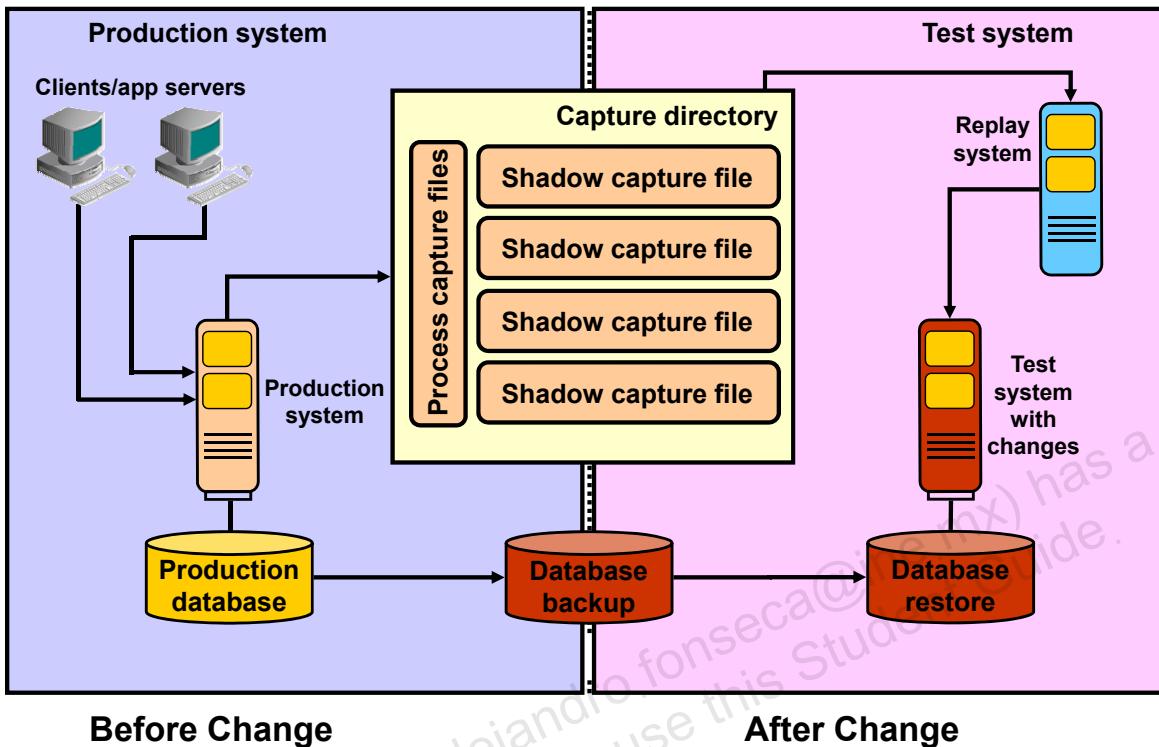
Database Replay allows you to test the impact of a system change by replaying a real-world workload on a test system before it is exposed to a production system. The production workload (including transaction concurrency and dependency) of the database server is recorded over an interesting period of time (for example, a peak period). This recorded data is used to replay the workload on a test system that has been appropriately configured. You gain a high degree of confidence in the overall success of the database or OS change by subjecting the database server in a test system to a workload that is practically indistinguishable from a production workload.

Database Replay can test the performance of a new server configuration. Consider a scenario where you want to move a single instance database to a RAC setup. You record a workload over a period of time and then set up a RAC test system for replay. During replay, you can monitor the performance benefit of the new configuration by comparing the performance to the recorded system.

You can use Database Replay in debugging to record and replay sessions, emulating an environment to make bugs more reproducible. You can also test manageability features. Self-managing and self-healing systems need to implement advice automatically ("autonomic computing model"). Multiple replay iterations allow testing and fine-tuning of the control strategies' effectiveness and stability.

Database Replay can be used for many scenarios where change-assurance functionality is required. The database administrator, or a user with privileges granted by the DBA, initiates the record-and-replay cycle and has full control over the entire procedure.

The Big Picture



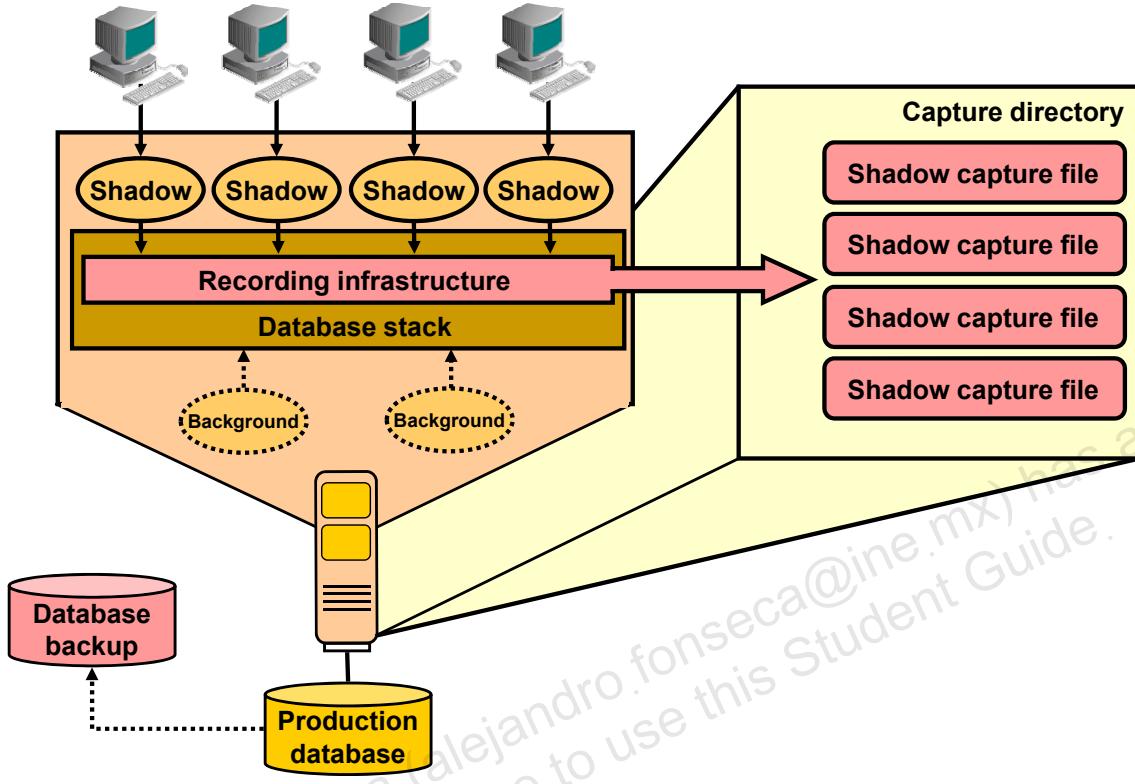
ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The following are the typical steps to use Database Replay. Steps performed by using Enterprise Manager are marked as Task *n*. Other steps are not part of the Enterprise Manager workflow:

1. Capture the workload on a database. (Task 1)
2. Optionally, export the AWR data. (Task 1)
3. Restore the replay database on a test system to match the capture database at the start of the workload capture. The DBA determines the details of this step. It depends on the backup strategy used to return the test system to the starting SCN of the capture recording.
4. Make changes (such as performing an upgrade) to the test system as required.
5. Copy the generated workload files to the replay system.
6. Preprocess the captured workload on the test system or on a system with the same version of the database that exists on the test system. (Task 2)
7. Configure the test system for the replay. (Calibrate the `wrc` processes.)
8. Replay the workload on the restored database. (Task 3)

System Architecture: Capture



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You should always record a workload that spans an “interesting” period in a production system. A special recording infrastructure built into the RDBMS records data about all external client requests while the production workload is running on the system. External requests are any SQL queries, PL/SQL blocks, PL/SQL remote procedure calls, DML statements, DDL statements, Object Navigation requests, or Oracle Call Interface (OCI) calls. During the recording, background jobs and, in general, all internal clients continue their work without being recorded. The end product is a workload recording containing all necessary information for replaying the workload as seen by the RDBMS in the form of external requests. Database Replay focuses on recording the workload at the RDBMS level.

Recording at the RDBMS level in the software stack makes it possible to exchange anything below this level and test the new setup using the record-and-replay functionality.

The recording infrastructure imposes minimal performance overhead (extra CPU, memory, and input/output) on the recording system. You should, however, plan to accommodate the additional disk space needed for the actual workload recording.

While replaying the workload, the RDBMS performs the actions observed during recording. In other words, during the replay phase, the RDBMS code is exercised in a way that is very similar to the way it was exercised during the recording phase. This is achieved by re-creating all external client requests to the RDBMS. External client requests include all the requests by all possible external clients of the RDBMS.

The following is a list of supported external requests that will be recorded:

Supported

- All SQL (DML, DDL, PL/SQL) with practically all types of binds
- Full LOB functionality (cursor-based and direct OCI)
- Local transactions
- Logins and logoffs
- Session switching
- Limited PL/SQL RPCs

The following is a list of possible requests that are not recorded:

Unsupported

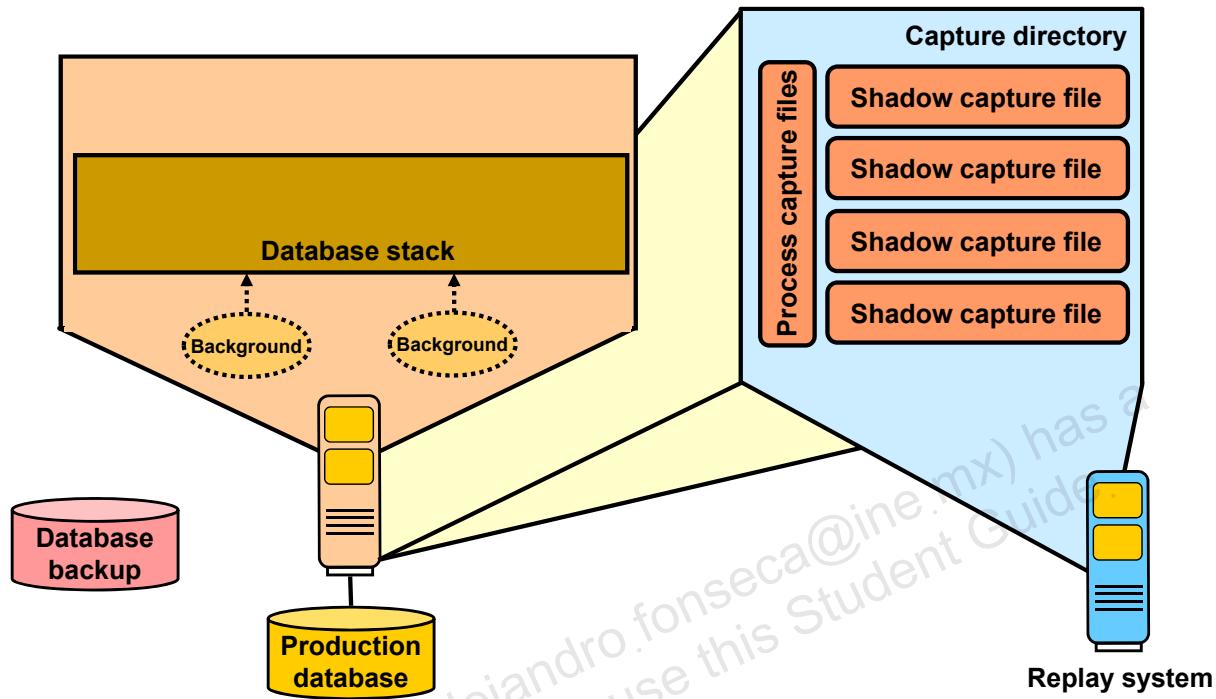
- Direct path load, import/export
- OCI-based object navigation (ADTs) and REF binds
- Streams, non-PL/SQL-based AQ
- Distributed transactions, remote describe/commit operations (will be replayed as local transactions)
- Flashback (database and query)
- Shared server
- Database Resident Connection Pooling (DRCP)
- Multi-threaded Server (MTS) and shared server sessions with synchronization mode set to OBJECT_ID
- XA transactions: XA transactions are not captured or replayed. All local transactions are captured.
- JAVA_XA transactions: If the workload uses the JAVA_XA package, JAVA_XA function and procedure calls are captured as normal PL/SQL workload. To avoid problems during workload replay, consider dropping the JAVA_XA package on the replay system to enable the replay to complete successfully.

Typically, Database Replay refrains from capturing these types of non-supported user sessions and client requests. Even when they are captured, Database Replay will not replay them. Therefore, it is usually not necessary to manually filter out non-supported user sessions and client requests. In cases where they are captured and found to cause errors during replay, consider using workload capture filters to exclude them from the workload.

In general all external client requests are recorded and internal client requests are not recorded.

Note: SQL-based XML manipulations are also captured. The system only captures explicit SQL statements (SQL statements issued by clients). Implicit calls generated by the database itself are not captured. For example, auditing is implicit, like background processes activity is implicit.

System Architecture: Processing the Workload



ORACLE

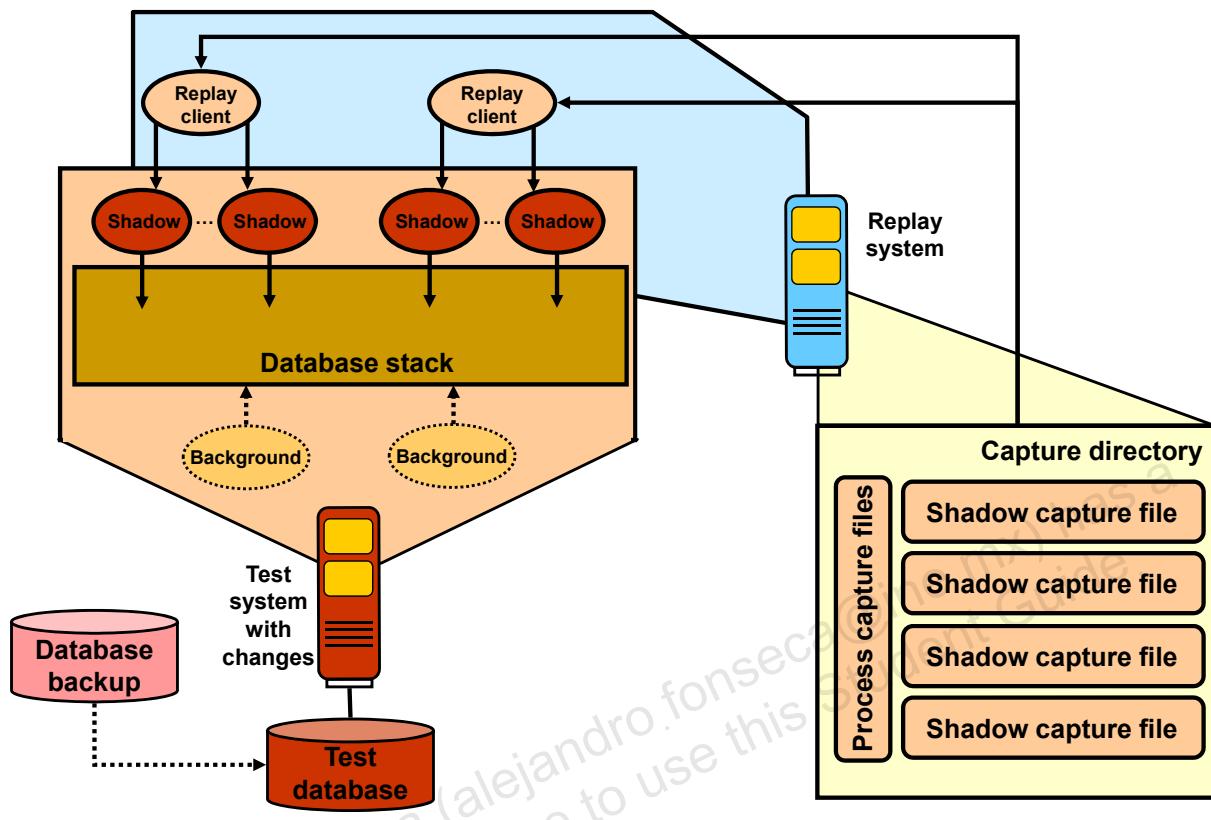
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The workload capture data is processed and new workload replay-specific metadata files are created that are required for the replay of the given workload capture. Only new files are created; no files are modified that were created during the workload capture. Because of this, you can run the preprocess multiple times on the same capture directory (for example, when the procedure encounters unexpected errors or is canceled).

External client connections are remapped at this stage. Any replay parameters that affect the replay outcome can be modified.

Note: Because processing workload capture can be relatively expensive, the best practice is to do that operation on a system other than the production database system.

System Architecture: Replay



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Before replaying the workload on the replay system, ensure that you do the following:

1. Restore the replay database on a test system to match the capture database at the start of the workload capture.
2. Make changes (such as performing an upgrade) to the test system as needed.
3. Copy the workload to the replay system.

The workload recording is consumed by a special application called the *replay driver*, which sends requests to the RDBMS on which the workload is replayed. The RDBMS on which the workload is replayed is usually a test system. It is assumed that the database of the replay system is suitable for the replay of the workload that was recorded. The internal RDBMS clients are not replayed. The replay driver is a special client that consumes the workload recording and sends appropriate requests to the test system to make it behave as if the external requests were sent by the clients used during the recording of the workload (see previous example). The use of a special driver that acts as the sole external client to the RDBMS enables the record-and-replay infrastructure to be interoperable with any client OS.

The replay driver consists of one or more clients that connect to the replay system and the replay driver sends requests based on the workload capture. The replay driver equally distributes the workload capture streams among all the replay clients based on the network bandwidth, CPU, and memory capability.

Database Replay Workflow in Enterprise Manager

Capture Task List

1. Capture the workload on a database.
2. Optional: Export the AWR data.

Replay Task List

1. Prepare the Test Database.
 - A. Restore the replay database on a test system.
 - B. Isolate the Test Database.
2. Prepare for Replay.
 - A. Preprocess the captured workload.
 - B. Deploy Replay Clients.
3. Replay the workload on the Test Database.

 ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

All steps can be performed either through Enterprise Manager Cloud Control (EMCC) or manually using the API. See the *Oracle Database Testing Guide 12c Release 1* for details.

The Capture Task Wizard in EMCC enables you to schedule an export of the AWR data, or to perform it manually.

In general, the Test database is completely separate and isolated from the database where the capture takes place.

Capture Considerations

Planning check list:

- Disk space for captured workload (binary files)
- Database restart:
 - The only way to guarantee authentic replay is to capture all transactions.
 - Startup restrict (prevents capture of partial transactions)
 - Capture will un-restrict, when it successfully starts.
 - Database restart may not be necessary depending on the workload.
- A way to restore database for replay purposes:
 - Physical restore (scn/time provided)
 - Logical restore of application data
 - Flashback/snapshot-standby



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You perform the following tasks in the planning phase of the workload recording:

- Check the database backup strategy ensuring that the database can be restored to StartSCN when the recording starts.
- Plan the capture period by selecting it based on the application and the peak periods. You can use existing manageability features, such as AWR and ASH, to select an appropriate period based on workload history. The starting time for capture should be carefully planned to reduce the capture of partial transactions. These will contribute to divergence in the replay. Restart before capture is recommended when there are a large percentage of partial (in-flight) transactions captured.
- Specify the location of the workload capture data. Create a directory that is to be used to store the workload capture data. Provide ample disk space. Recording will stop if there is insufficient disk space. However, everything captured up to that point is usable for replay.
- Define capture filters for user sessions that are not to be captured. You can specify a recording filter to skip sessions that should not be captured.
- No new privileges or user roles are needed with Database Replay. The recording user and replay user must have either the SYSDBA or SYSOPER privilege to start up or shut down the database to start the recording.

Capture Considerations

- Filters to specify a subset of the workload to capture
- SYSDBA or SYSOPER privileges and appropriate OS privileges

Overhead:

- Performance overhead for TPCC is 4.5%.
- Memory overhead is 64 KB per session.
- Disk space is required for the captured workload.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Note: A partial transaction could have a major effect on divergence if it were a dependency for subsequent transactions. If the workload can be controlled to prevent the possibility of partial or missed transaction between the starting SCN and the beginning of capture, then a database restart is not necessary.

Capturing Workload with Enterprise Manager

Database Replay

Plan Environment Database Options Storage Schedule Review

Create Capture : Review

Database
Concurrent Capture No

Capture Databases

Database Name	Capture Name	Intermediate Storage Location	Description
orcl	ORCL_11gSim	/u01/app/oracle/capture	

Options

SQL Performance Analyzer
 Capture SQL statements into a SQL Tuning Set during workload capture.

Workload Filters

Excluded Sessions

Filter Name	Type	Session Attribute	Value
Oracle Management Service (DEFAULT Excluded)	Program	OMS	
Oracle Management Agent (DEFAULT Excluded)	Module	emagent%	

Storage

Storage Host edRSr7p1.us.oracle.com
 Storage Location /u01/app/oracle/capture

Schedule

Capture Schedule

Start Immediately
 Duration For 5 minutes

Automatic Workload Repository

Start Immediately

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

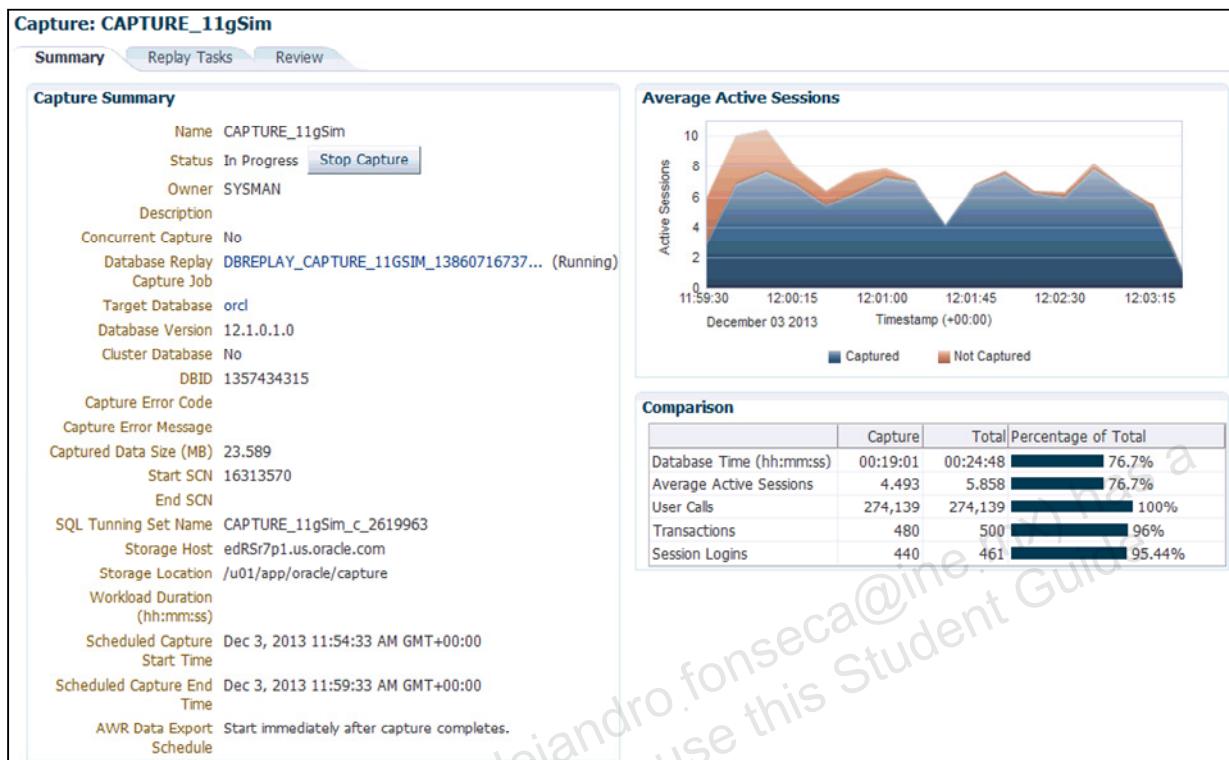
Enterprise Manager Cloud Control provides you with a user interface to manage each component in the Database Replay process. You access Database Replay from the Enterprise > Quality Management menu.

On the Database Replay page, you can create named tasks to capture workloads, or create named replay tasks. When you create a capture task, a wizard guides you through six steps:

- **Plan Environment:** Verifies that you checked the Capture Prerequisites
- **Database:** Allows you to select the databases where the workload will be captured. When you add the database, you also specify credentials, and choose a temporary storage location for the capture files. The temporary location is local to the database where the Capture is executed; on RAC systems, this should be a shared directory.
- **Options:** Enables you to choose to capture SQL Tuning Set for use with SQL Performance Analyzer, and to capture either specified sessions or exclude specified sessions
- **Storage:** Specifies the storage host name, provide host credentials, and a final location for the capture files

RAC Note: For RAC, the DBA should define the directory for captured data at a storage location accessible by all instances. Otherwise, the workload capture data needs to be copied from all locations to a single location before starting the processing of the workload recording.

Viewing Capture Progress



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Use the Capture: Capture_name page to see the progress of a workload capture that you have initiated, or to see historical information about the capture after it has completed execution. This page is available from the Database Replay home page, by clicking the name link on the Captured Workloads tab.

This page has the following elements:

- The Summary section gives you important information about the current workload capture.
 - Use the Stop Capture button while capture is in progress.
- Note:** If the AWR export is scheduled Immediately, the Status will display “In Progress,” until the AWR export is completed.
- The Review tab shows the same information as the Review page of the Create Capture Wizard.

Capture Reports Tab

The screenshot shows the 'Capture: CAPTURE_11gSim' interface. The 'Reports' tab is selected. Under 'Capture Reports', there are three links: 'Database Capture Report' (with a 'View' button), 'Workload Analyzer Report' (with a 'View' button), and 'Capture ASH Analytics Report'. Below these is a dropdown menu 'Captured Workload SQL and Wait Events by Wait Class' with a 'View' button. To the right of the reports section is a large report window titled 'Database Capture Report For ORCL'. This window contains two tables: 'Information' and 'Capture'. The 'Information' table shows details like DB Name (ORCL), DB Id (1357434315), Release (12.1.0.1.0), RAC (NO), Capture Name (CAPTURE_11gSim), and Status (COMPLETED). The 'Capture' table shows details like Start time (03-Dec-13 11:59:32), End time (03-Dec-13 12:04:34), Duration (5 minutes 2 seconds), Capture size (34.28 MB), Directory object (CAPTURE_61_1386071673740), Directory path (/u01/app/oracle/capture/DBReplay/Workload_CAPTURE_11gSim_61), Directory shared in RAC (TRUE), and Filters used (2 EXCLUSION filters). Below these tables is a section titled 'DB Capture Report' with a bulleted list: 'Capture Statistics', 'Workload Captured', 'Workload Not Captured - Contains Unreplayable Calls', and 'Workload Not Captured - User Filtered'.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Reports tab subpage provides links to capture related reports. Click View Database Capture Report to invoke a browser window to display the report. The report contains detailed information about the capture. The Captured Workload Statistics table shows cumulative values for the workload after you started the capture.

The Workload Analyzer report is not available until after the workload has been preprocessed.

Replay Considerations: Preparation

- Preprocess captured workload:
 - One-time action
 - On same DB version as replay
 - Can be performed anywhere (production, test system, or other system), if versions match
- Restore the database, and then perform the change:
 - Upgrade
 - Schema changes
 - OS change
 - Hardware change
 - Add instance



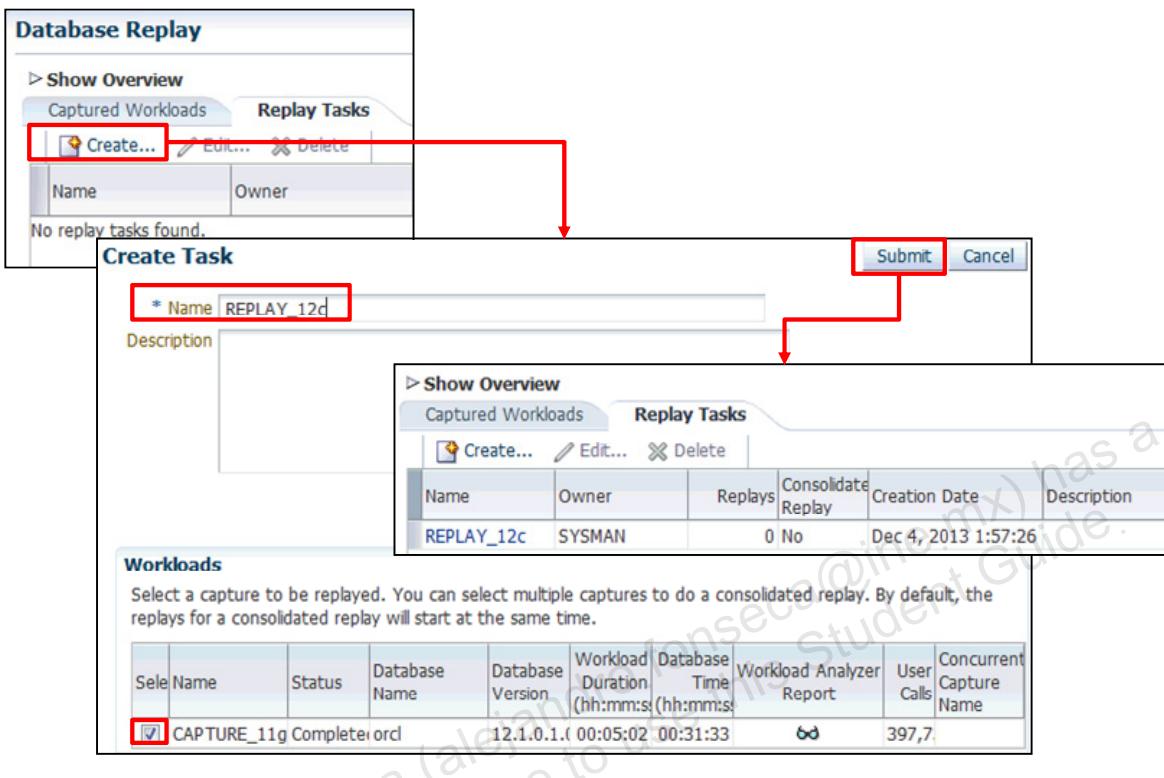
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The preprocess phase is a once-only required action for the specified target database version. After the necessary metadata has been created, you can replay the workload as many times as required.

The preprocess workload task also performs the workload analysis. The Workload Analyze report is available on the Capture Reports page after the preprocess task has been completed. This report shows important metrics, such as percentage of incomplete transactions. These metrics can help to determine if it is worthwhile to continue with a replay, or start over with a new capture.

You must restore the replay database to match the capture database at the start of the workload capture. A successful replay depends on the application transactions accessing application data that is identical to that on the capture system. You can choose to restore the application data using point-in-time recovery, flashback, and import or export.

Create Replay Task with Enterprise Manager



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The first step to replay a workload is to create a Replay Task. Every replay task will have one or more captured workloads associated with it. Give the replay task a name. Select the workload or workloads to be executed. Click Submit.

Each replay belongs to a Replay task, and will have its own name.

Create a Replay

Replay Task: REPLAY_12c

Replay Task Summary

Name	REPLAY_12c
Description	
Owner	SYSMAN

Captured Workloads

Name	Database Name	Database Version	Database Time (hh:mm:ss)	Workload Duration (hh:mm:ss)	Workload Analyzer Report	Storage Host	Storage Location
CAPTURE_1	orcl	12.1.0.1.0	00:31:33	00:05:02	6d	edRSr7p1.us.oracle.com	/u01/app/oracle/capture

Replays

Name	Owner	Status
No replays found.		

Create Replay

Name: replay_1

Description:

Target Database: orcl

OK Cancel

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can create one or more replays associated with the same Replay Task. Click Create in the Replays section, and the Create Replay popup appears. Give the replay a name and specify the target database. Click OK.

Replay Tasks

Replay: replay_1

Home **Replay Target**

Target **orcl** Database Version 12.1.0.1.0 Replay Host edRSr7p1.us.oracle.com

Database

Task List
Please click a link or click an icon under 'Go to Task' to execute a task.

Task	Description	Go To Task
Prepare Test Database	Set up and prepare a test database environment to be used for replay. Steps include cloning the production database, restoring the database to the point of capture and making any additional changes necessary to the test database environment.	
Set Up Test Database	Clone the production database to a test environment. The test database should be restored to match the capture database at the start of capture. You may make any changes to the test environment as needed.	
Isolate Test Database	Isolate the test system from the production environment prior to the workload replay.	
Prepare for Replay	Prepare (preprocess) the workload capture files for replay and deploy the Replay Clients.	
Preprocess Workload	Preprocess the captured workload. Preprocessing prepares the workload for replay and only needs to be performed once against a specific database version. A workload should be preprocessed using the target test database.	
Deploy Replay Clients	Deploy Replay Clients	
Replay Workload on Test Database	Set up the workload replay on the test database and analyze the results.	
Replay Workload	Replay the preprocessed workload on a test copy of the production database.	

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Each replay requires that the test database is set up and is verified to be isolated. Best practice is to run the replay on a database that is completely separate from the production database. It is common for the capture to be run on a production database. In general, the workload must not be replayed against the production database. These tasks can be run from Enterprise Manager Cloud Control.

Before a workload can be replayed it must be preprocessed on the same version database as the test database. The processed workload can be run multiple times, and the test database can be a different database each time. Use the Preprocess Workload to create a Preprocess workload task.

If the replay clients are run on multiple machines, the replay clients can be deployed to target systems.

Replay Considerations

- Manage external interactions:
 - Remap connection strings to be used for the workload.
 - One-to-one: For simple instance-to-instance remapping
 - Many-to-one: Use of load balancer
 - (Example: Single node to RAC)
 - Modify DB links and directory objects that point to production systems.
- Set up one or more replay clients.
 - Multithreaded clients that can each drive multiple workload sessions



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A captured workload may contain references to external systems that are meaningful only in the capture environment. A replay should be performed in a completely isolated test environment. You should make sure that all references to external systems have been resolved in the replay environment such that replaying a workload will cause no harm to your production environment.

You can make one-to-one or many-to-one remappings. For example, database links in a captured production environment may reference external production databases that should not be referenced during replay. Therefore, you should modify any external references that could jeopardize the production environment during replay.

The replay client (an executable named `wrc`) submits a captured session's workload. You should install one or more replay clients, preferably on systems other than the production host. Each replay client must be able to access the directory that holds the preprocessed workload. You can also modify the replay parameters to change the behavior of the replay.

To measure the difference between the replay system and the capture system, multiple replays can be initiated: one with the replay system set as close to the capture system as possible, and then change the replay system to match the target system and replay again.

Best Practice: Make one change per replay run.

The options can be set in the Create Replay Wizard. The steps of the wizard are shown in the slide.

Preprocess Workload



Preprocess Captured Workload: Review

Replay Name replay_1
Logged In As sys

The current database version is 12.1.0.1.0. Continue only if you intend to replay the captured workload on a database of the same version.

Workload CAPTURE_11gSim will be preprocessed on database 'orcl'.

Job Name	PREPROCESS-REPLAY_1-20131205082244
Database	orcl
Preprocessed Database Version	12.1.0.1.0
Directory Object	WORKLOAD
Directory Path	/u01/app/oracle/workload
Capture Name	CAPTURE_11gSim
Captured Data Size (MB)	34.28
Start Time	Immediately
Run Workload Analyzer	Yes

Cancel Back Step 5 of 5 **Submit**

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When you choose to preprocess the capture workload, the Preprocess Wizard in EMCC guides each step.

On the **Locate Workload** page, either copy the workload to the server where the preprocess operation will be done, or specify that the workload is already on this machine.

Copy Workload will copy the workload from a specified server to the local machine and is skipped if the workload is on the local machine.

On the **Select Directory** page, provide the current location of the capture workload directory. The directory must exist.

The **Review** page shows the parameters you have specified, and then you can submit the job.

Running a Replay

Replay: replay_1

Home

Replay Target

Target

Database Version 12.1.0.1.0 Replay Host edRSr7p1.us.oracle.com

Database

Task List

Please click a link or click an icon under 'Go to Task' to execute a task.

Task	Description	Go To Task
Prepare Test Database	Set up and prepare a test database environment to be used for replay. Steps include cloning the production database, restoring the database to the point of capture and making any additional changes necessary to the test database environment.	
Set Up Test Database	Clone the production database to a test environment. The test database should be restored to match the capture database at the start of capture. You may make any changes to the test environment as needed.	
Isolate Test Database	Isolate the test system from the production environment prior to the workload replay.	
Prepare for Replay	Prepare (preprocess) the workload capture files for replay and deploy the Replay Clients.	
Preprocess Workload	Preprocess the captured workload. Preprocessing prepares the workload for replay and only needs to be performed once against a specific database version. A workload should be preprocessed using the target test database.	
Deploy Replay Clients	Deploy Replay Clients	
Replay Workload on Test Database	Set up the workload replay on the test database and analyze the results.	
Replay Workload	Replay the preprocessed workload on a test copy of the production database.	

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

After creating the replay task, and a replay, you can run the replay with **Replay Workload**. This invokes the Replay Wizard, which enables you to set the replay options.

Replay Customized Options

- Synchronized replay:
 - Ensures minimal data divergence
 - Commit-based synchronization
- Unsynchronized replay:
 - Useful for load/stress testing
 - Original commit ordering not honored
 - High data divergence
- Think time options:
 - Auto (default)
 - Adjust think time to maintain the captured request rate:
 - 0%: No think time (highest possible request rate)
 - <100%: Higher request rate
 - 100%: Exact think time
 - >100%: Lower request rate
- Login time options
 - Percentage (default is 100%)



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The following replay options can be modified while replaying your workload:

- The `synchronization` parameter determines whether synchronization will be used during workload replay. If this parameter is set to `TRUE`, the `COMMIT` order in the captured workload will be preserved during replay and all replay actions will be executed only after all dependent `COMMIT` actions have completed. The default value is `TRUE`.
- The `think_time_scale` parameter scales the elapsed time between two successive user calls from the same session; it is interpreted as a percentage value. Use this parameter to increase or decrease the replay speed. Setting this parameter to 0 will send user calls to the database as fast as possible during replay. The default value is 100.
- The `think_time_auto_correct` parameter corrects the think time (based on the `think_time_scale` parameter) between calls, when user calls take longer to complete during replay than during capture. Its value can be `TRUE` or `FALSE`.
- The `connect_time_scale` parameter scales the elapsed time from when the workload capture started to when the session connects with the specified value; it is interpreted as a percentage. Use this option to manipulate the session connect time during replay. The default value is 100.

Note: During workload capture, elapsed time is measured by user time and user think time. User time is the elapsed time of a user call to the database. User think time is the elapsed time while the user waits between issuing calls. During workload replay, elapsed time is measured by user time, user think time, and synchronization time.

Replay: Customize Options Replay Parameters

Connection Mappings			Replay Parameters
Name	Description	Value	
synchronization	This parameter determines what type of synchronization will be used during workload replay. If this parameter is set to SCN, the COMMIT order in the captured workload will be globally preserved during replay and all replayed requests will be executed only after all COMMIT actions with a lower capture-time SCN have completed. If this parameter is set to OBJECT_ID, a finer method of synchronization is used which is based on capture-time SCN values as well as database objects to calculate the dependencies among replayed calls. The default value is SCN.	SCN	<input type="button" value="▼"/>
connect_time_scale	This parameter scales the elapsed time from when the workload capture started to when the session connects with the specified value and is interpreted as a % value. The parameter controls the rate of logon activity during replay. The default value is 100.	100 %	<input type="button" value="▼"/>
think_time_scale	This parameter scales the elapsed time between two successive user calls from the same session and is interpreted as a % value. The parameter controls the replayed request rate. Thus, setting this parameter to 0 will send user calls to the database as fast as possible during replay. The default value is 100.	100 %	<input type="button" value="▼"/>
think_time_auto_correct	This parameter reduces the think time if workload replay goes slower than workload capture, in an attempt to maintain the captured request rate. If this parameter is set to TRUE, the system will correct the think time (based on the think_time_scale parameter) between calls when user calls take longer to complete during replay than during capture. The default value is TRUE.	TRUE	<input type="button" value="▼"/>

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Replay Parameters tab of the Customize Options page in the Create Replay Wizard provides advanced parameters that control some aspects of the replay. The Workload Analysis report provides some guidance for setting replay options depending on the workload characteristics.

Viewing Workload Replay Progress



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When you submit the replay, a progress window provides comparison statistics as the replay progresses.

You can terminate the replay at any stage with the Stop Replay button.

The Review tab allows you to see all the parameters that were specified for the replay.

When the workload is complete, the Status shows Completed, and a Report Tab appears.

The Database Time Comparison chart shows how much Database time the replayed workload has taken to accomplish the same amount of work as the captured workload. This comparison is a very rough indication of the difference in efficiency between the capture and replay.

The divergence table gives information about both the data and error discrepancies between the replay and capture environments, which can be used as a measure of the replay quality. The divergence summary is not available until the replay is complete.

Replay Analysis

- Data divergence:
 - Number of rows compared for each call (queries, DML)
- Error divergence:
 - New errors
 - Mutated errors
 - Errors that have disappeared
- Performance:
 - Capture and Replay report
 - ADDM report
 - ASH report for skew analysis
 - AWR report



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

There may be some divergence of the replay compared to what was recorded. For example, when replaying on a newer version of the RDBMS, a new algorithm may cause specific requests to be faster, resulting in divergence appearing as a faster execution. This is considered a desirable divergence. Another example of a divergence is when a SQL statement returns different number of rows during replay than those returned during recording. This is clearly undesirable.

For data divergence, the result of an action can be considered as:

- The result set of SQL query
- An update to persistent database state
- A return code or an error code

Performance divergence is useful in determining how new algorithms introduced in the replay system may affect overall performance. There are numerous factors that can cause replay divergence. Though some of them cannot be controlled, others can be mitigated. It is the task of the DBA to understand the workload run-time operations and take necessary actions to reduce the level of record-and-replay divergence.

Online divergence should aid the decision to stop a replay that has diverged significantly. The results of the replay before the divergence may still be useful, but further replay would not produce reliable conclusions. Offline divergence reporting is used to determine how successful the replay was after the replay has finished.

Data divergence of the replay encompasses the results of both queries and errors. That is, errors that occurred during recording are considered proper results and any change during replay is reported. You can use existing tools, such as ADDM, to measure performance differences between the recording system and the replay system.

Additionally, error comparison reports during the replay report on the following:

- Errors that did not happen during recording
- Errors that were not reproduced during replay
- Difference in error type

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

Viewing Workload Replay Reports

The screenshot shows the Oracle Database Performance Management interface. At the top, there are three tabs: Home, Reports (which is selected), and Review. Below the tabs, the title "Replay Reports" is displayed. Under "Replay Reports", there are five report options with "View" buttons: Database Replay Report, Compare Period ADDM Report, SQL Performance Analyzer Report, Replay Compare Period Report, and Replay ASH Analytics Report. The "Replay ASH Analytics Report" section includes a dropdown menu for "Replayed Workload SQL and Wait Events by Wait Class" and a "View" button. Below these reports is a "Regenerate Reports" button. A "Information" section contains a note about generating replay reports after blocking issues are resolved. The "Replay Issues" section shows a table with columns: Step Name, Step Status, Start Time, End Time, Target Name, Target Type, and Job Name. The table displays the message "No replay issues found".

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Reports tab lists the replay issues and the reports that are available.

Note: The Workload Capture report is available from the Capture Details page after the workload has been preprocessed.

- **Database Replay Report:** Gives a detailed comparison of the replay and capture
- **Compare Period ADDM Report:** Provides a high-level comparison that includes a cause analysis
- **SQL Performance Analyzer Report:** Lists the top SQL that have regressed, and includes both execution plans from the capture and the replay providing more information than the standard SQL Analyzer
- **Replay Compare Period Report:** Is an AWR compare period report comparing the capture and the replay
- **Replay ASH Analytics Report:** Has multiple options to customize the report to the area of interest

Quiz

The Capture portion of Database Replay captures all external client requests but not internal client requests such as those generated by DBWN. How would you limit the sessions that are actually captured?

- a. Eliminate SYSDBA operations.
- b. Stop internal clients.
- c. Add a filter to exclude unwanted sessions.
- d. Stop the listener to prevent unwanted logons.
- e. Force external clients to disconnect.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: c

The Filters section on the options page of the Create Capture Wizard enables you to create named filters to either include or exclude sessions from the capture. By default, the OMS and Agent sessions are excluded. The filters are applied for a capture workload inclusively, with the filters specifying the session to include, or exclusively specifying the sessions to exclude. Inclusive and exclusive filters cannot exist in the same capture task.

Database Replay Packages

- DBMS_WORKLOAD_CAPTURE
 - START_CAPTURE
 - FINISH_CAPTURE
 - ADD_FILTER
 - DELETE_FILTER
 - DELETE_CAPTURE_INFO
 - GET_CAPTURE_INFO()
 - EXPORT_AWR
 - IMPORT_AWR()
 - REPORT()
- DBMS_WORKLOAD_REPLAY
 - PROCESS_CAPTURE
 - INITIALIZE_REPLAY
 - PREPARE_REPLAY
 - START_REPLAY
 - CANCEL_REPLAY
 - DELETE_REPLAY_INFO
 - REMAP_CONNECTION
 - EXPORT_AWR
 - IMPORT_AWR
 - GET_REPLAY_INFO
 - REPORT



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You need the EXECUTE privilege on the packages to execute the functions and procedures in these packages. These privileges are usually assigned by the DBA.

Note: For further details about the DBMS_WORKLOAD packages, see the *Oracle Database PL/SQL Packages and Types Reference*.

Data Dictionary Views: Database Replay

- DBA_WORKLOAD_CAPTURES: Lists all workload captures performed in the database
- DBA_WORKLOAD_FILTERS: Lists all workload filters defined in the database
- DBA_WORKLOAD_REPLAYS: Lists all workload replays that have been performed in the database
- DBA_WORKLOAD_REPLY_DIVergence: Is used to monitor workload divergence
- DBA_WORKLOAD_CONNECTION_MAP: Is used to review all connection strings that are used by workload replays
- V\$WORKLOAD_REPLY_THREAD: Monitors the status of external replay clients

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

For information about these views, see the *Oracle Database Reference*.

Database Replay: PL/SQL Example

```
exec DBMS_WORKLOAD_CAPTURE.ADD_FILTER(fname      => 'sessfilt', -
                                         fattribute => USER      ,-
                                         fvalue      => 'JFV');
```

```
exec DBMS_WORKLOAD_CAPTURE.START_CAPTURE(name => 'june_peak', -
                                            dir   => 'jun07');
```

Execute your workload

```
exec DBMS_WORKLOAD_CAPTURE.FINISH_CAPTURE();
```

```
exec DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE(capture_dir => 'jun07');
```

wrc userid=system password=oracle replaydir=/dbreplay

```
exec DBMS_WORKLOAD_REPLAY.INITIALIZE_REPLAY(replay_name => 'j_r' ,-
                                               replay_dir => 'jun07');
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In this example, the ADD_FILTER procedure adds a filter named sessfilt that will filter out all sessions belonging to the JFV username.

To start the workload capture, use the START_CAPTURE procedure. In this example, a capture named june_peak is captured and stored in a directory named jun07. Because the duration parameter is not specified, the workload capture continues until the FINISH_CAPTURE procedure is called. At this point, you can run your workload.

To stop the workload capture, use the FINISH_CAPTURE procedure. This procedure finalizes the workload capture and returns the database to a normal state.

You can now generate a capture report by using the REPORT function.

To preprocess a captured workload, use the PROCESS_CAPTURE procedure. In this example, the captured workload stored in the jun07 directory is preprocessed.

When finished, you can start your replay clients.

To initialize replay data, use the INITIALIZE_REPLAY procedure. Initializing replay data loads the necessary metadata into tables required by the workload replay. For example, captured connection strings are loaded into a table where they can be remapped for replay. In this example, the INITIALIZE_REPLAY procedure loads preprocessed workload data from the jun07 directory into the database.

Database Replay: PL/SQL Example

```

exec DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION(connection_id => 101,-
                                              replay_connection => 'edlin44:3434/bjava21') ;

exec DBMS_WORKLOAD_REPLAY.PREPARE_REPLAY(synchronization => TRUE,-
                                         think_time_scale=> 2) ;

exec DBMS_WORKLOAD_REPLAY.START_REPLAY () ;

DECLARE
  cap_id NUMBER;
  rep_id NUMBER;
  rep_rpt CLOB;
BEGIN
  cap_id := DBMS_WORKLOAD_REPLAY.GET_REPLAY_INFO(dir => 'jun07');
  /* Get the latest replay for that capture */
  SELECT max(id) INTO rep_id
  FROM dba_workload_replays
  WHERE capture_id = cap_id;
  rep_rpt := DBMS_WORKLOAD_REPLAY.REPORT(replay_id => rep_id,
                                             format => DBMS_WORKLOAD_REPLAY.TYPE_TEXT);
END;

```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To remap connections, use the REMAP_CONNECTION procedure. In this example, the connection that corresponds to the connection ID 101 will use the new connection string defined by the replay_connection parameter.

To prepare workload replay on the replay system, use the PREPARE_REPLAY procedure. In this example, the PREPARE_REPLAY procedure prepares the j_r replay to preserve the COMMIT order in the workload capture.

To start a workload replay, use the START_REPLAY procedure. To stop a workload replay, use the REPLAY_CANCEL procedure

To generate a workload replay report, use the REPORT function as shown in the slide.

Calibrating Replay Clients

```
$ wrc mode=calibrate replaydir=$ORACLE_BASE/replay
Report for Workload in: .

-----
Recommendation:
Consider using at least 1 clients divided among 1 CPU(s)
You will need at least 116 MB of memory per client process.
If your machine(s) cannot match that number, consider using
more clients.

Workload Characteristics:
- max concurrency: 31 sessions
- total number of sessions: 40

-Assumptions:
- 1 client process per 50 concurrent sessions
- 4 client process per CPU
- 256 KB of memory cache per concurrent session
- think time scale = 100
- connect time scale = 100
- synchronization = TRUE
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Because one replay client can initiate multiple sessions with the database, it is not necessary to start a replay client for each session that was captured. The number of replay clients that need to be started depends on the number of workload streams, the number of hosts, and the number of replay clients for each host.

For example, suppose that a workload capture has 1,000 streams, that the number of average active sessions from the workload capture is about 60, and that one host can drive only 50 concurrent connections to the database. You should use two hosts, each with a replay client.

To estimate the number of replay clients and hosts that are required to replay a particular workload, run the `wrc` executable in calibrate mode. In calibration mode, the `wrc` executable accepts the following parameters:

- `replaydir` specifies the directory that contains the preprocessed workload capture that you want to replay. If unspecified, it defaults to the current directory.
- `process_per_cpu` specifies the maximum number of client processes that can run for each CPU. The default value is 4.
- `threads_per_process` specifies the maximum number of threads that can run in a client process. The default value is 50.

The example in the slide shows how to run the `wrc` executable in the calibrate mode. In this example, the `wrc` executable is executed to estimate the number of replay clients and hosts that are required to replay the workload capture stored in the `$ORACLE_BASE/replay` directory.

Note: To list the hosts that participated in the capture, use the `list_hosts` mode.

Quiz

Database Replay can be used to determine the overall performance change for a variety of situations. Which of the following performance tests is NOT a proper use of Database Replay?

- a. A change in operating system versions
- b. A workload applied to a different database version
- c. A workload from a single instance applied to RAC database
- d. A set of SQL statements between database versions
- e. A workload applied to an instance with changed parameters



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: d

Database Replay reports give information only about the top SQL statements. To determine if particular statements regressed, use the SQL Performance Analyzer.

Summary

In this lesson, you should have learned how to:

- Identify the benefits of using Database Replay
- List the steps involved in Database Replay
- Use Enterprise Manager to record and replay workloads

Practice 15: Using Database Replay

In this practice covers in video:

- Capturing a workload
- Replaying a captured workload and viewing the reports
- Replaying a workload for comparison. SQL Profiles have been applied and the workload replayed.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

16

Tuning the Shared Pool

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Diagnose and resolve hard-parsing problems
- Diagnose and resolve soft-parsing problems
- Size the shared pool
- Diagnose and resolve shared pool fragmentation
- Keep objects in the shared pool
- Size the reserved area
- Manage the SQL query result cache

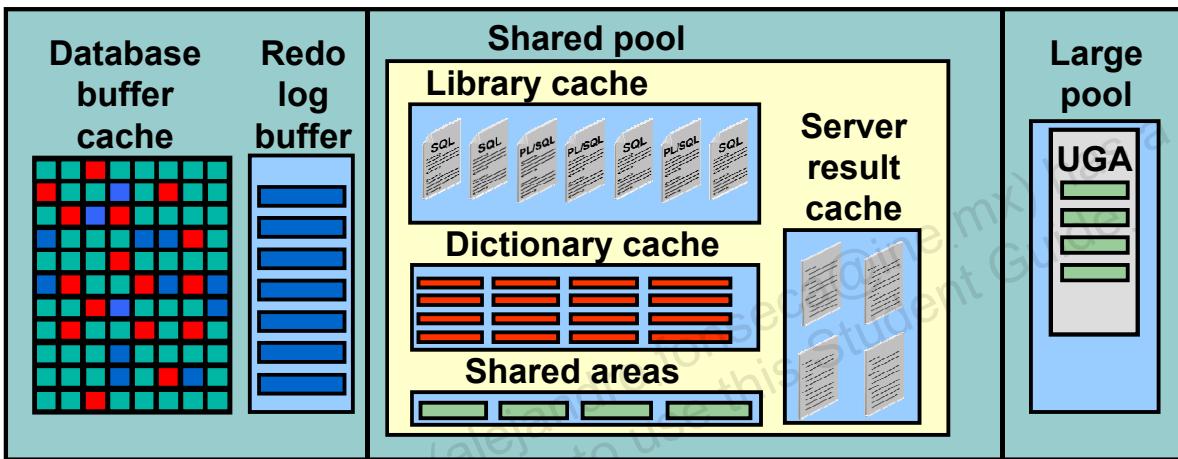
ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Shared Pool Architecture

Major components of the shared pool:

- Library cache
- Data dictionary cache
- Server result cache
- User Global Area (when large pool is not configured)



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The fundamental purpose of the shared pool is to serve as a metadata cache. Much of the shared pool usage is to support the execution of shared SQL and PL/SQL packages. For example, to build a cursor for a simple SELECT statement, the optimizer must be aware of the table, column, and index metadata and optimizer statistics. This data is cached in the shared pool independent of the cursor, so that it can be reused. The shared pool holds many named areas and fixed areas that are allocated on startup. The v\$SGASTAT view lists these areas.

The major components of the shared pool are the following:

- **Library cache:** Stores shared SQL and PL/SQL code and object metadata in areas that are distinguished by namespaces
- **Data dictionary cache:** Holds row images from the data dictionary tables, and is also called the row cache
- **Server result cache:** Holds query result sets and query fragments, so subsequent queries can retrieve the result directly from the cache

Space allocation in the shared pool is managed by a least recently used (LRU) algorithm. The DBA is most concerned about shared pool allocations that are re-creatable. The re-creatable allocations can be aged out to make space for other re-creatable allocations. For best performance, you must strike a balance between the amount of memory dedicated to the shared pool and the processing required to reload these re-creatable objects.

Shared Pool Operation

The shared pool is managed by an LRU algorithm.

- New objects require memory allocations.
- Re-creatable objects are aged out of the cache.
- Objects are made up of chunks of memory.
- A memory allocation is a whole chunk.
- A chunk is contiguous.
- LRU operations are protected by a latch or mutex.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Oracle Database server removes releasable objects from the shared pool when a new object requires memory. These new objects are most often new statements (cursors) that require memory in the library cache, but can be data dictionary objects in the row cache, and other objects. The Oracle Database server uses an LRU algorithm to determine the objects to be removed. The server processes scan the LRU list removing the least recently used objects and checking for sufficient contiguous memory to satisfy the request. If sufficient contiguous memory cannot be found, you receive an error (ORA-04031), unless the automatic shared memory manager is enabled; in that case, an additional granule of memory is requested.

SQL and PL/SQL statements do not require contiguous memory for the entire statement or block. Memory heaps used to represent the SQL statement in the shared pool are allocated in chunks. The chunks do not have to be contiguous, but each chunk allocation must be contiguous. The chunk sizes can vary, but, where possible, memory is allocated in 1 KB or 4 KB chunks creating more uniform memory allocations. As a result, at a steady state, the same kinds of objects and chunks sizes are allocated and aged out. This allocation method reduces fragmentation. Practice shows that in most cases it reduces total memory usage as well.

Shared pool LRU scans and allocations are protected operations using either a latch or a mutex. Contention during operations appear as latch or mutex waits.

Library Cache

- Stores complex object metadata associated with cursors
- Stores SQL statements and PL/SQL blocks that are to be shared by users
- Allows cursor sharing



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The library cache is the most important part of the shared pool from a tuning perspective. If the library cache is well tuned, the other components of the shared pool usually work efficiently.

Complex object metadata associated with the cursors is stored in the library cache. The Oracle database server uses the library cache to store SQL statements and PL/SQL blocks so that users can share statements, thus reducing the need to parse a similar statement. Each of these types of data associated with the cursors has a name (or namespace) assigned to the allocation. This naming allows the memory usage to be displayed as a sum of a type of allocation. For example, the SQL AREA namespace shows all of the allocation for SQL and PL/SQL executable code. The TABLE/PROCEDURE and INDEX namespaces show the allocations for each of these object types.

The instance collects statistics for each namespace. The SQL AREA namespace is usually one of the most active areas. Use the SQL AREA namespace to judge whether the shared pool is properly sized.

Each row in V\$LIBRARYCACHE contains statistics for one type of item kept in the library cache, grouped by the value of the NAMESPACE column. The statistics shown are for the entire namespace. Rows of the table with the following NAMESPACE values reflect library cache activity for SQL statements and PL/SQL blocks: SQL AREA, TABLE/PROCEDURE, BODY, and TRIGGER.

Rows with other NAMESPACE values reflect library cache activity for object definitions that the server uses for dependency maintenance: INDEX, CLUSTER, OBJECT, PIPE, JAVA SOURCE, JAVA RESOURCE, and JAVA DATA.

Keywords related to the namespace are the following:

- GETS: Shows the total number of requests for information about the corresponding item
- PINS: Shows the number of executions of SQL statements or procedures
- RELOADS: Shows the number of times, during the execution phase, that the shared SQL area containing the parsed representation of the statement is aged out of the library cache to make room for another statement. The Oracle server implicitly reloads the statement and parses it again.
- INVALIDATIONS: Shows the number of statements that have been made invalid due to the modification of a dependent object. Invalidations are caused by DDL and by gathering statistics against a dependent objects. Invalidations also cause reloads.

Latch and Mutex

- A *latch* is a memory object that allows:
 - Sharing of a resource without corruption
 - Exclusive access for update and allocation
 - Shared access for reads
- A *mutual exclusion object (mutex)* allows:
 - Sharing of a resource without corruption
 - Shared access for reads
 - Exclusive access for update
 - Each object can have its own mutex



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Latches and mutexes perform the same function, to protect resources from corruption when accessed by multiple processes. A latch is an inexpensive form of a lock. A single latch generally protects multiple objects in the SGA. Mutex is the short form of “mutual exclusion object.” A mutex, similar to a latch, is a low-level serialization mechanism used to control access to a shared data structure in the SGA. Serialization is required to avoid an object being:

- Deallocated while someone is accessing it
- Read while someone is modifying it
- Modified while someone is modifying it
- Modified while someone is reading it

Mutexes and latches are both serialization mechanisms.

Mutexes are a faster, more scalable way to manage the library cache while allowing more granular monitoring—for example, updating SQL execution statistics at the cursor level during execution instead of once at the end.

Note: Latches and mutexes are independent mechanisms; that is, a process can hold a latch and a mutex at the same time. Some latch wait events are implemented with mutexes.

Benefits of mutexes:

- **Have less potential for false contention:** Latches typically protect multiple objects. When a latch protects one or more hot objects, the latch itself can become a serialization point when accessing any of the objects protected by that latch. This can be a false contention point, where the contention is for the protection mechanism (that is, latch), rather than the target object that you are attempting to access. Unlike latches, with mutexes it is possible to create a mutex for each structure protected. This means false contention is much less likely because each structure can be protected by its own mutex.
- **Replace latches and pins:** A mutex can be concurrently referenced by many sessions, providing that all sessions reference the mutex in S (Shared) mode. The total number of sessions referencing a mutex in S mode is called the reference count (“ref count”). The ref count for a mutex is stored within the mutex itself. A mutex can also be held in X (eXclusive) mode by one session only.
Mutexes have a dual nature; they can act as a serialization mechanism (for example, latch), and also as a pin (for example, preventing an object from aging out). For example, the ref count of a mutex is a replacement for a library cache pin. Instead of each session creating and then deleting a library cache pin when executing a cursor, each session increments and decrements the ref count (so the ref count replaces n distinct pins).

Latch and Mutex: Views and Statistics

- Latch views:
 - V\$LATCH
 - V\$LATCH_MISSES
 - V\$LATCH_PARENT
 - V\$LATCH_CHILDREN
 - V\$LATCHHOLDER
 - V\$LATCHNAME
- Latch wait events:
 - latch: *latchname*
- Mutex views:
 - V\$MUTEX_SLEEP
 - V\$MUTEX_SLEEP_HISTORY
- Mutex wait events:
 - cursor:mutex X
 - cursor:mutex S
 - cursor:pin X
 - cursor:pin S
 - cursor:pin S wait on X



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Latch and mutex wait events serve as diagnostics that point to particular issues. Mutex operations are faster and have less contention than latches, but they still have waits associated with them.

When the top wait events include waits on latches, the views shown in the slide provide additional details. These views are the basis for the Statspack and AWR report sections on latches. V\$LATCH shows an aggregation of the latch waits grouped by latch name. A latch can have multiple child latches with the same name. V\$LATCH_MISSES shows statistics about missed attempts to obtain a latch.

Two common latch wait events that involve the shared pool are:

- Latch: shared pool – Protects shared pool memory allocations
- Latch: row cache objects – Protects access and updates to the data dictionary cache

Details about waits and sleeps involving mutexes can be seen in the following v\$ views:

- V\$MUTEX_SLEEP shows a summary of sleeps and wait time for a particular mutex_type/location combination.
- V\$MUTEX_SLEEP_HISTORY shows sessions sleeping for a particular mutex_type/location combination by time, while it is held by a specific holding session.

Mutex wait events have two categories:

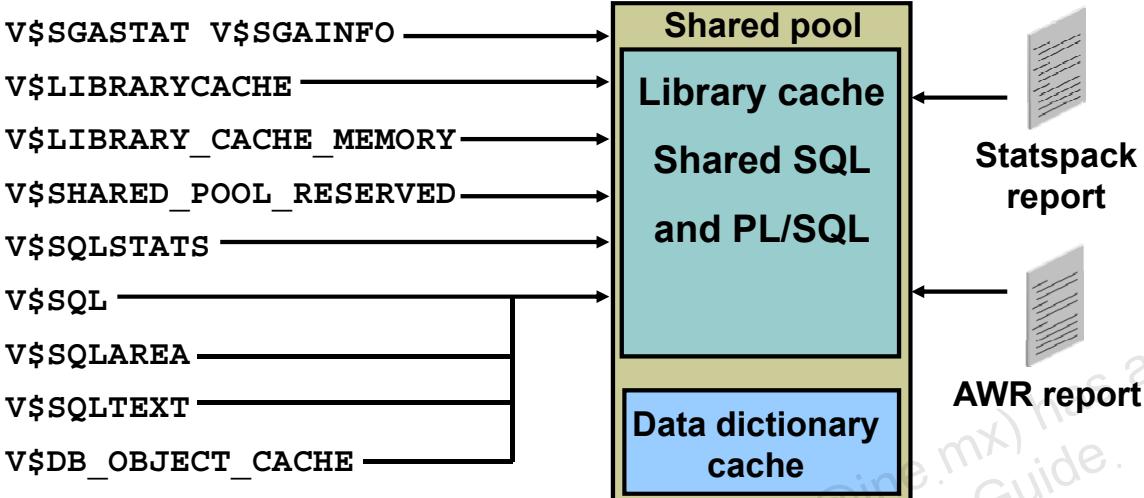
- `cursor:mutex` events are session waits on parent cursor operations and statistics operations where the mutex is being held by another session in an incompatible mode.
- `cursor:pin` events are waits for cursor pin operations, where a mutex has replaced the `latch:library cache` pin.

Mutex wait events are of two types:

- Short-duration events that should rarely be seen. These occur when one process attempts to update the mutex while it is being changed by another process. The waiting process will spin rather than sleep while waiting for the mutex to be available. For example, `cursor:pin S` is incremented when another process is updating the reference count (pin) of a shared cursor.
- Long-duration events occur when a process must wait for other processes to finish their operation. For example, `cursor: mutex X` is incremented when a process wants an exclusive access, but the mutex is being held exclusively or shared by another process.

Diagnostic Tools for Tuning the Shared Pool

Views



Parameters affecting the components:

**SHARED_POOL_SIZE, OPEN_CURSORS,
SESSION_CACHED_CURSORS,
CURSOR_SHARING, SHARED_POOL_RESERVED_SIZE**

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The V\$SGASTATS and V\$SGAINFO views display the sizes of all SGA structures. V\$SGASTATS is more detailed for the various pieces of the shared pool. V\$SGAINFO provides a summary of the SGA components including the granule size and max memory size.

The contents of the shared pool are not aged out as long as free memory is available in the shared pool. Use the following dynamic views to diagnose performance issues related to the library cache:

- V\$LIBRARYCACHE: Statistics on library cache management
- V\$LIBRARY_CACHE_MEMORY: Statistics on memory use for each of the namespaces
- V\$SQLAREA: Full statistics about all shared cursors. The first 1,000 characters of the SQL statement for backward compatibility and the full SQL statement as a CLOB are included.
- V\$SQL: Statistics about shared SQL area; contains one row for each child of the original SQL text entered. The V\$SQL view is a similar view to V\$SQLAREA, except that it does not include a GROUP BY clause that can make the V\$SQLAREA view expensive to query.
- V\$SQLSTAT: A low impact view for use instead of the other V\$SQL* views
- V\$SQLTEXT: The full SQL text without truncation, in multiple rows

- V\$SQLTEXT_WITH_NEWLINES: Is identical to the V\$SQLTEXT view except that, to improve legibility, V\$SQLTEXT_WITH_NEWLINES does not replace newlines and tabs in the SQL statement with spaces
- V\$DB_OBJECT_CACHE: Database objects cached, including packages; also objects such as tables and synonyms, where these are referenced in SQL statements

These views hold the raw information that is formatted for use in the AWR and Statspack reports. The Statspack and AWR reports use these views and also contain information about SQL statements found in the library cache when either of the snapshots was taken.

Each of the initialization parameters listed has some effect on the behavior of size of the shared pool, and are covered later in this lesson.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

AWR/Statspack Indicators

AWR and Statspack reports include indicators:

- Top Wait Events: Shows the areas that will have the greatest impact on instance performance
- Time Model: Provides the first best indicators for the root cause
- Load Profile: Shows instance activity
- Instance Efficiencies:
 - Shows the relative health of the database
 - Can be used to narrow and confirm a diagnosis of a performance problem



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When diagnosing performance problems, the AWR and Statspack reports provide indicators of the type of load and basic health of the instances, as well as diagnostics that provide clear direction for finding the root cause.

The Top Wait Events section of the reports should be the starting point for diagnosis. Top Wait Events shows the areas that will have the greatest impact on the instance performance.

The Time Model section gives the best indicators of the root cause. This is the same information that the ADDM analysis uses.

The Load Profile section shows the instance activity. The Instance Efficiencies section shows relative health of the database and can be used to narrow and confirm a diagnosis of a performance problem.

The set of AWR report sections shown in the following slides are taken from a single AWR report that shows a workload where cursors are not being shared.

Top Timed Events

AWR Report

Top 10 Foreground Events by Total Wait Time					
Event	Waits	Total Wait Time (sec)	Wait Avg(ms)	% DB time	Wait Class
DB CPU		57		92.2	
db file sequential read	11,806	1	0	1.6	User I/O
log file sync	137	.9	7	1.5	Commit
enq: KO - fast object checkpoint	2	.6	303	1.0	Application
direct path write	15	.6	38	.9	User I/O
direct path read	5,942	.1	0	.2	User I/O
direct path sync	1	.1	123	.2	User I/O
cursor: pin S wait on X	3	0	15	.1	Concurrency
buffer exterminate	2	0	10	.0	Other
latch: shared pool	245	0	0	.0	Concurrency

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The top timed event is DB CPU. This event can be in the top events in a well-tuned database so this by itself is not very useful. But when combined with the time model, it provides a clear indication of a hard-parse problem.

Contention in the shared pool shows up as waits on the latches.

- **Latch: shared pool** indicates that there were a large number of memory allocations that required holding the shared pool latch.
- **Latch: row cache objects** indicates that there is contention loading, referencing, and freeing objects in the data dictionary cache. This can be caused by excessive row-cache access operations in a hard-parse scenario.

Note: In this example, the total time waited for both latches is less than 1% of the total time. Therefore, there will not be a noticeable performance improvement, even if all the contention on these latches were removed. A larger contention on these latches would point to a shared pool issue and, in combination with the other indicators, shows that cursors are not being shared.

Time Model

AWR Report

Statistic Name	Time (s)	% of DB Time
DB CPU	57.04	92.19
sql execute elapsed time	54.29	87.74
parse time elapsed	49.17	79.46
hard parse elapsed time	43.59	70.44
PL/SQL execution elapsed time	2.61	4.21
hard parse (sharing criteria) elapsed time	1.20	1.93
connection management call elapsed time	0.34	0.54
PL/SQL compilation elapsed time	0.19	0.30
hard parse (bind mismatch) elapsed time	0.07	0.11
sequence load elapsed time	0.02	0.03
repeated bind elapsed time	0.02	0.03
DB time	61.88	
background elapsed time	13.27	
background cpu time	1.79	

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Time Model section has some of the few statistics that can be directly compared between two reports. The % of DB Time column normalizes the time to the DB Time of the report.

Recall the time model hierarchy; parse time elapsed includes hard parse elapsed time. In the report shown, it is clear that the hard parse elapsed time is taking the largest percentage of time. Many of these statistics apply directly to tuning the shared pool. The report section shown is based on the V\$SYS_TIME_MODEL view. For information at the session level, you can query V\$SESS_TIME_MODEL.

```
SQL> SELECT stat_name, value FROM V$SYS_TIME_MODEL;
```

STAT_NAME	VALUE
DB time	256524305
DB CPU	200364479
background elapsed time	59837351
background cpu time	8853650
sequence load elapsed time	53091
parse time elapsed	131949764
hard parse elapsed time	117557924

STAT_NAME	VALUE
sql execute elapsed time	217361219
connection management call elapsed time	2082373
failed parse elapsed time	337
failed parse (out of shared memory) elapsed time	0
hard parse (sharing criteria) elapsed time	3495714
hard parse (bind mismatch) elapsed time	82804
PL/SQL execution elapsed time	10308216
inbound PL/SQL rpc elapsed time	0
PL/SQL compilation elapsed time	1400307
Java execution elapsed time	0
repeated bind elapsed time	47945
RMAN cpu time (backup/restore)	0
OLAP engine elapsed time	0
OLAP engine CPU time	0

21 rows selected.

```
SQL> SELECT sid, stat_name, value FROM V$SESS_TIME_MODEL
  2 order by sid;
```

SID	STAT_NAME	VALUE
Lines deleted ...		
262	DB CPU	7944791
262	background elapsed time	0
262	background cpu time	0
262	sequence load elapsed time	0
262	parse time elapsed	10380910
262	hard parse elapsed time	8474303
262	sql execute elapsed time	6091438
262	connection management call elapsed time	23819
262	failed parse elapsed time	0
262	failed parse (out of shared memory) elapsed time	0
262	hard parse (sharing criteria) elapsed time	17519
262	hard parse (bind mismatch) elapsed time	0
262	PL/SQL execution elapsed time	20245
262	inbound PL/SQL rpc elapsed time	0
262	PL/SQL compilation elapsed time	278569
262	Java execution elapsed time	0
262	repeated bind elapsed time	3488
262	RMAN cpu time (backup/restore)	0
262	OLAP engine elapsed time	0
262	OLAP engine CPU time	0

Lines deleted ...

861 rows selected.

Load Profile

AWR Report

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	0.3	6.9	0.00	0.05
DB CPU(s):	0.3	6.3	0.00	0.05
Redo size (bytes):	39,641.8	932,961.3		
Logical read (blocks):	6,494.2	152,838.4		
Block changes:	153.5	3,613.4		
Physical read (blocks):	291.4	6,857.3		
Physical write (blocks):	8.4	196.4		
Read IO requests:	102.6	2,413.9		
Write IO requests:	6.7	156.8		
Read IO (MB):	2.3	53.6		
Write IO (MB):	0.1	1.5		
User calls:	5.6	131.8		
Parses (SQL):	399.0	9,390.1		
Hard parses (SQL):	310.5	7,307.0		
SQL Work Area (MB):	0.8	19.0		
Logons:	0.1	2.2		
Executes (SQL):	526.4	12,387.9		
Rollbacks:	0.0	0.0		
Transactions:	0.0			

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Load Profile section of both Statspack and AWR reports shows the activity of the instance. The only numbers that stand out in this example are the parses and hard parses. Hard parses are approximately 75% of the total number of parses. This can be confirmed in the Instance Efficiencies section.

A high percentage of hard parses indicates that the SQL cursors are not being shared. On an OLTP system or any environment where users are running an application, the percentage of hard parses should be very small. In decision support systems (DSS) and Data Warehouse (DWH) systems where users are accessing the database with ad hoc tools, the percentage of hard parses is much higher. Whether this number is meaningful depends on the type of system you are running.

A baseline for your system gives more meaning to the percentage of hard parses. When the percentage changes, you can ask, "What has changed in the workload?"

Instance Efficiencies

AWR Report

Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	100.00	Redo NoWait %:	100.00
Buffer Hit %:	98.65	In-memory Sort %:	100.00
Library Hit %:	66.29	Soft Parse %:	22.18
Execute to Parse %:	24.20	Latch Hit %:	99.96
Parse CPU to Parse Elapsd %:	96.37	% Non-Parse CPU:	21.71

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In the Instance Efficiencies section of the reports, there are several indicators that there is a shared pool issue:

- **Library Hit%** shows the percentage of times a requested object was found in the cache. Ideally this number is close to 100%.
- **Soft Parse%** shows the percentage of times that a requested cursor was found and reused. When this number is low, cursors are not being reused.
- **Execute to Parse%** is the number of executions to the number of parses. This number could be negative if executions have not occurred before the end of a snapshot or if the shared pool is too small and queries are aging out of the shared pool and need to be reparsed before they have finished executing. When cursors are being shared, this number will approach 100%.
- **Parse CPU to Parse Elapsed%:** The example shows that the CPU time is taking almost all of the total elapsed time for parsing. Parsing tends to be CPU intensive. The remaining elapsed parse time is likely being spent in waits.

Library Cache Activity

AWR Report

Library Cache Activity

- "Pct Misses" should be very low

Namespace	Get Requests	Pct Miss	Pin Requests	Pct Miss	Reloads	Inval- dations
ACCOUNT_STATUS	35	14.29	0		0	0
AUDIT POLICY	8	37.50	8	37.50	0	0
BODY	58	34.48	953	6.40	37	0
CLUSTER	320	2.50	320	2.50	0	0
DBLINK	23	13.04	0		0	0
EDITION	17	35.29	41	39.02	0	0
HINTSET OBJECT	204	78.92	204	79.90	2	0
INDEX	35,345	0.09	35,345	0.14	18	0
OBJECT ID	5	100.00	0		0	0
QUEUE	1	0.00	85	3.53	2	0
SCHEMA	1,325	0.91	0		0	0
SQL AREA	77,032	85.09	258,848	75.54	572	302
SQL AREA BUILD	65,274	99.91	0		0	0
SQL AREA STATS	64,942	99.83	64,942	99.83	0	0
SUMMARY	56,480	0.00	56,480	0.00	0	0
TABLE/PROCEDURE	212,564	0.51	369,211	1.20	1,858	0

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The SQL area is the most important part of the library cache. This is the area where the memory for compiled cursors is allocated. In this example, the Pct Miss value is very high, indicating that cursors are not being reused or they are being reloaded (because of age or size of the shared pool).

The number of reloads should not be more than 1% of the number of pins. The reload to pin ratio is very low in this example.

In a workload where cursors could be shared, there are two possible reasons for the reloads-to-pins ratio being greater than 1%:

- The shared parsed areas have been aged out because of lack of space. This can happen even when the cursor is being used frequently.

Solution: Avoid the frequent reloads by increasing the SHARED_POOL_SIZE initialization parameter. For manual memory tuning, use the Shared Pool Advisor to determine the proper sizing of the shared pool.

- Shared parsed areas are invalidated.

Solution: Perform housekeeping (such as creating indexes and gathering statistics) during periods of light load.

Avoid Hard Parses

In an OLTP system, reduce misses by keeping hard-parsing to a minimum:

- Make sure that users can share statements.
- Prevent frequently used statements from being aged out by allocating enough space.
- Avoid invalidations that induce reparsing.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In an OLTP environment, you should minimize reparsing.

- If an application makes a parse call for a SQL statement and the parsed representation of the statement does not already exist in a shared SQL area in the library cache, the Oracle server parses the statement and allocates a shared SQL area.
- Ensure that SQL statements can share a shared SQL area whenever possible by using as much reusable code as possible—for example, having bind variables rather than constants.
- Frequently-used statements should always be found in the shared pool. If the pool is too small, then such statements may be aged out between uses. This means the subsequent execution requires a reparse. Allocating more memory to the library cache reduces library cache misses on execution calls.
- If a schema object is referenced in the execution plan of a SQL statement and that object is later modified in any way, the parsed SQL statement held in memory is rendered invalid and must be parsed again when used.

The `latch: shared_pool` latch must be obtained for any shared pool memory allocation. Hard parses require memory allocations. Contention for the shared pool latch generally indicates a high rate of memory allocations and deallocations. This is often associated with an undersized shared pool; a high rate of hard parses also causes contention on this latch.

Are Cursors Being Shared?

- Check Get Pct Miss for SQL Area in the AWR report :

Namespace	Get Requests	Pct Miss	Pin Requests	Pct Miss	Reloads	Invali- dations
SQL AREA	9,554	35.96	635,766	0.40	745	0

- If Pct Miss is greater than 10% in an OLTP system:
 - Improve the sharing of your application code by using bind variables
 - Increase the size of the shared pool
 - Set the CURSOR_SHARING parameter

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Get Pct Miss for SQL Area in the Library Cache Activity section in the AWR or Statspack report shows the percentage of parse calls that do not find a cursor to share. The Get Pct Miss should be less than 10% in OLTP environments.

If not, you can try to:

- Improve the sharing of your application code by using bind variables. This might not be an option if you do not have access to the application code.
- Hold cursors in the SQL area longer by increasing the size of the shared pool
- Force cursor sharing by setting the CURSOR_SHARING parameter

Applications that use dynamic SQL and submit literal values create cursors that vary only in the literals. These applications have high hard-parse rates and high Get Pct Miss. There are also child cursors that are identical but cannot be shared. The V\$SQL_SHARED_CURSOR view explains why a particular child cursor is not shared with existing child cursors, with a column for each reason.

Candidate Cursors for Sharing

- Determine which statements could be shared:

```
SQL> SELECT plan_hash_value, count(*) FROM v$sql
  3 WHERE parsing_schema_name not in
  4   ('SYS','SYSPMAN','DBSNMP')
  5 GROUP BY plan_hash_value ORDER BY 2;

PLAN_HASH_VALUE      COUNT(*)
-----
2288362790          50
```

```
SQL> SELECT sql_text, executions FROM v$sqlarea
  3 WHERE plan_hash_value = 2288362790;

SQL_TEXT                           EXECUTIONS
-----
select SUM(AMOUNT_SOLD) from sh.sales           1
where cust_id = 8620
select SUM(AMOUNT_SOLD) from sh.sales           1
where cust_id = 100909
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The first statement in the slide returns the count of PLAN_HASH_VALUE for unshared cursors ordered by the number of cursors using the same plan. Note that all cursors using the same plan could be shared.

The second statement, where PLAN_HASH_VALUE = 2288362790 displays the SQL statements with identical execution plans that could share a single cursor. Note that in this example, there are 50 child cursors and the two that are shown were executed once. If the literal values could be replaced with a bind variable, there would be one cursor with 50 executions.

Sharing Cursors

Values for CURSOR_SHARING are:

- EXACT (default): SQL statements must be identical to share cursors.
- FORCE: SQL statements that are *similar* will share cursors regardless of the impact on the execution plan by using bind variables. Bind peeking is done on the first parse, and possibly later due to adaptive cursor sharing.

Similar statements are identical, except for the values of some literals.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Similar statements have the potential of being shared. Similar statements are identical, except for the values of some literals. Examine the results of the queries from the previous page to determine if you have SQL statements that are similar, differing only in literal values in their WHERE clauses. If you find statements with these characteristics, you may be able to reduce the number of statements requiring space in the library cache by setting the CURSOR_SHARING parameter.

The value of the CURSOR_SHARING parameter determines the level at which SQL statements will share cursors:

- EXACT: (default) When the parameter is set to EXACT, SQL statements must be identical to share cursors.
- FORCE: SQL statements that are similar will share cursors regardless of the impact on the execution plan by using bind variables. Bind peeking is done on the first parse, and possibly later due to adaptive cursor sharing.

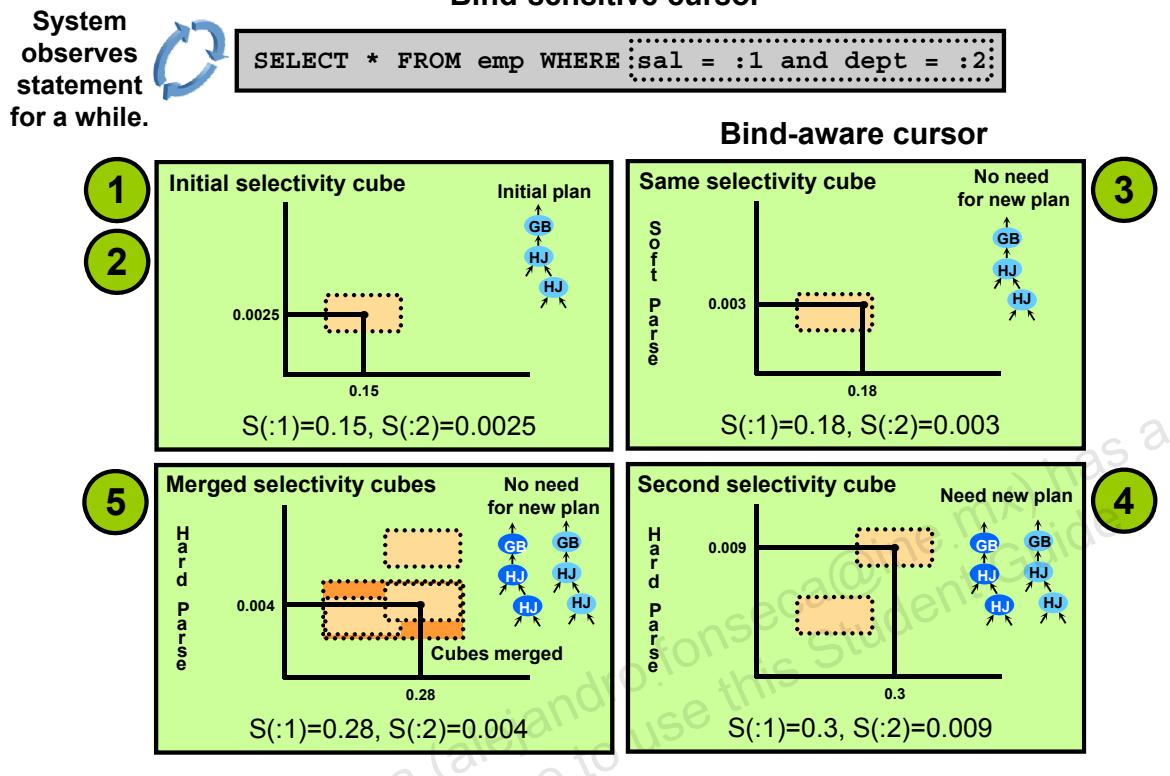
Note: It is recommended to set CURSOR_SHARING to EXACT in a DSS environment or if you are using complex queries. Set CURSOR_SHARING to FORCE for OLTP unless the SQL is consistently using bind variables.

Cursor sharing and SQL optimization are conflicting goals. Bind variables allow a single cursor to be shared by multiple SQL statements to reduce the parse time and the use of shared memory. Writing a SQL statement with literals provides more information for the optimizer and leads to better execution plans, while increasing memory for storing cursors and CPU overhead caused by hard parses. The CURSOR_SHARING parameter is a compromise solution that allows similar SQL statements using different literal values to be shared.

The optimizer uses a feature called bind peeking for statements using bind variables, to look at the bind values the first time that the statement is executed. It uses these values to determine an execution plan that will be shared by all other executions of that statement. The optimizer does not do bind peeking after the execution plan has been determined. If different invocations of the statement would significantly benefit from different execution plans, bind peeking is of no use in generating good execution plans. However, this behavior is modified by adaptive cursor sharing if histograms are present over the columns using the bind variables

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

Adaptive Cursor Sharing: Example



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In situations where a different bind value would result in a different execution plan, a histogram must be generated over the column. The Oracle Database optimizer makes use of bind variables in the predicate and histograms through adaptive cursor sharing. It generates multiple plans per SQL statement with bind variables if the benefit of using multiple execution plans outweighs the parse time and memory usage overhead. It provides a compromise between the number of child cursors generated and ability to share cursors.

Using Adaptive Cursor Sharing, the following steps take place. In each step, the bind variables get a different set of values, so the selectivity varies. For this example, assume that two bind variables are in the predicate and that there is a histogram for each variable. Because there are bind variables and histograms, the cursor is marked as a bind-sensitive cursor.

1. The cursor is created with a hard parse. The initial execution plan is determined using the peeked binds. A selectivity cube surrounding the current selectivity values is defined and stored with the cursor. After the cursor is executed, the bind values and the execution statistics of the cursor are stored in that cursor.
2. For the next execution of the statement a new set of bind values is used, the system performs a usual soft parse and finds the matching cursor for execution. At the end of execution, execution statistics are compared with the ones currently stored in the cursor.

The system then observes the pattern of the statistics over all previous runs. If the statistics are significantly worse, the system marks the cursor as *bind-aware*.

3. On the next parse of this query, the parent cursor is found for a soft parse. Bind-aware cursor matching is used. Selectivity is used as part of bind-aware cursor matching, and the selectivity is within an existing cube; therefore the existing child cursor's execution plan is used.
4. On the next parse of this query, suppose that the selectivity is not within an existing cube, no child cursor match is found. So the system does a hard parse, which generates a new child cursor with a second execution plan, and another selectivity cube is stored as part of the new child cursor. After the new child cursor executes, the system stores the bind values and execution statistics in the cursor.
5. On the next parse of this query, if the selectivity is not within one of the existing cubes, the system does a hard parse and generates a new child cursor. Suppose this new child cursor has the same execution plan as the first child cursor. Because the plans are the same, both child cursors are merged. The selectivity cubes for both child cursors are merged into a new bigger cube, and one of the child cursors is deleted. The next time there is an execution, if the selectivity falls within the new cube, the child cursor will match. This action reduces the memory usage and number of child cursors to check when performing a soft parse.

Adaptive Cursor Sharing Views

The following views provide information about Adaptive Cursor Sharing usage:

V\$SQL	Two columns show whether a cursor is bind-sensitive or bind-aware.
V\$SQL_CS_HISTOGRAM	Shows the distribution of the execution count across the execution history histogram
V\$SQL_CS_SELECTIVITY	Shows the selectivity cubes stored for every predicate containing a bind variable and whose selectivity is used in the cursor-sharing checks
V\$SQL_CS_STATISTICS	Shows execution statistics of a cursor using different bind sets.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The views listed in the slide show whether a query is *bind-aware* or *bind-sensitive*. Information about adaptive cursor sharing is exposed through V\$ views so that you can diagnose any problems. V\$SQL has two columns to show the state of the cursor:

- IS_BIND_SENSITIVE: Indicates if a cursor is bind-sensitive; value YES | NO. When the cursor is *bind-sensitive*, the optimizer peeked at bind variable values of this query when computing predicate selectivities and found that a change in a bind variable value may lead to a different plan. This implies a histogram exists.
- IS_BIND_AWARE: Indicates if a cursor is bind-aware; value YES | NO

The V\$SQL_CS_HISTOGRAM view shows the distribution of the execution count across a three-bucket execution history histogram.

The V\$SQL_CS_SELECTIVITY view shows the selectivity cubes or ranges stored in a cursor for every predicate containing a bind variable and whose selectivity is used in the cursor sharing checks. It contains the text of the predicates and the selectivity range low and high values.

The V\$SQL_CS_STATISTICS view summarizes the information that Adaptive Cursor Sharing collects to make the decision to make the cursor bind-aware. For a sample of executions, the view shows rows processed, buffer gets, and CPU time. The PEEKED column has the value YES if the bind set was used to build the cursor, and NO otherwise.

Interacting with Adaptive Cursor Sharing

- CURSOR_SHARING:
 - If CURSOR_SHARING is not set to EXACT, statements containing literals may be rewritten using bind variables.
 - If statements are rewritten, Adaptive Cursor Sharing may apply to them.
- SQL Plan Management (SPM):
 - If OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES is set to TRUE, only the first generated plan is used.
 - Workaround:
 - Set the OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES parameter to FALSE, and run your application until all plans are loaded in the cursor cache.
 - Manually load the cursor cache into the corresponding plan baseline.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Adaptive Cursor Sharing is independent of the CURSOR_SHARING parameter. Adaptive Cursor Sharing is applied anytime there are bind variables and histograms. The setting of the CURSOR_SHARING parameter determines whether literals are replaced by system-generated bind variables. If they are, then Adaptive Cursor Sharing behaves just as it would if the user supplied binds to begin with.

When using the SPM automatic plan capture, the first plan captured for a SQL statement with bind variables is marked as the SQL plan baseline. If another plan is found for that same SQL statement (which may be the case with Adaptive Cursor Sharing), it is added to the SQL statements plan history and marked for verification, but it will not be used until the plan has been verified. Only the plan generated based on the first set of bind values will be used by all subsequent executions of the statement. To avoid this behavior, run the system for some time with automatic plan capture set to false, and, after the cursor cache has been populated with all of the plans that a SQL statement with bind variables will have, load the entire set of plans directly from the cursor cache into the SQL plan baseline. By doing this, all the plans for a single SQL statement are marked as SQL baseline plans by default.

Reduce the Cost of Soft Parses

- Reducing soft parses reduces library cache latch contention.
- Keep cost of parsing to a minimum by setting:
 - SESSION_CACHED_CURSORS
 - HOLD_CURSOR in the application precompiler



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In an OLTP environment, you should minimize reparsing. Soft parses are less expensive than hard parses, but still take some time. In an OLTP environment, the number of soft parses is often in orders of magnitude greater than hard parses. Changing the following parameters may not reduce the parse calls statistic, but can make the parse do less work.

- SESSION_CACHED_CURSORS causes the session to keep a cache of recently closed cursors. The session searches the session cache first. If the cursor is found, the latch gets associated with searching the library cache are avoided. To verify that your setting is efficient, compare the session cursor cache hits and parse count session statistics in V\$SESSTAT for a typical user session. If few parses result in hits, you may increase the number. Remember that this increases the overall demands on memory.
- The HOLD_CURSOR parameter in the precompiler applications when set to YES, the link between the cursor and cursor cache is maintained after Oracle executes the SQL statement. Thus, the parsed statement and allocated memory remain available. This is useful for SQL statements that you want to keep active because it avoids unnecessary reparsing. This increases the demand for memory in the application session memory. The recommendation is to use HOLD_CURSOR selectively for individual cursors, instead of for the module as a whole.

Quiz

Cursors are stored in the shared pool, in the SQL area of the library cache. Each cursor takes some space and a place on the hash chain. When cursors are not shared:

- a. The hash chains are long and additional space is required in the shared pool
- b. The hash chains are short, and additional space is required in the shared pool
- c. Additional space is not required, but hash chains can be slow to search
- d. Additional space is required, but hash chains remain a reasonable length
- e. The CURSOR_SHARING parameter is set to FORCE



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: a

Sizing the Shared Pool

- Use Automatic Shared Memory Management.
- Use the Shared Pool Advisor and confirm using other diagnostics when data has operational history.
- Do not increase the size when free memory is available.

```
SQL> SELECT * FROM v$sgastat
  2 WHERE name = 'free memory'
  3 AND pool = 'shared pool';
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The SHARED_POOL_SIZE parameter is the only parameter available to influence the library cache size. The dictionary cache takes some portion of the shared pool as determined by an internal algorithm. The actual size required for the shared pool depends on number and size of cached objects, the number of concurrent users, concurrent open cursors, and other factors. Predicting the shared pool size from estimates is very inaccurate.

Oracle recommends using Automatic Shared Memory Management (ASMM). This is enabled by setting the SGA_TARGET parameter to the value that the entire SGA will use. The Memory Manager (MMAN) process adjusts the memory allocated to dynamic pools to obtain the best performance within the memory allowed.

When the database is already in operation, the Shared Pool Advisor gives an indication of an estimated optimal size. As with any estimate, check other indicators, such as latch waits, hard parses, and reloads, for confirmation.

The shared pool will grow as needed “stealing” from the other SGA components until the components are at minimum sizes; even when manual memory management is being used.

The shared pool requires a certain amount of memory for the instance to start. This value varies with the packages that are invoked at startup, and that depends on the features and options that are enabled.

Shared Pool Advisory

```
SQL> SELECT shared_pool_size_for_estimate AS
  2    pool_size, estd_lc_size,
  3    estd_lc_time_saved
  4  FROM v$shared_pool_advice;
```

POOL_SIZE	ESTD_LC_SIZE	ESTD_LC_TIME_SAVED
104	2	54531
116	14	71636
128	26	144857
140	38	188636
152	51	213387
164	63	214476
176	75	218280
188	87	222213



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The STATISTICS_LEVEL initialization parameter controls the shared pool advisory. The shared pool advisory statistics track the library cache's use of shared pool memory and predict the change in total instance-wide parse times for different sizes of the shared pool. Two views, V\$SHARED_POOL_ADVICE and V\$LIBRARY_CACHE_MEMORY, provide information to help you determine how much memory the library cache is using, how much is currently pinned, how much is on the shared pool's LRU list, and how much time might be lost or gained by changing the size of the shared pool. These statistics are reset if the STATISTICS_LEVEL parameter is set to BASIC, or when the instance is restarted.

The V\$SHARED_POOL_ADVICE view displays information about the estimated time saved during parsing using different shared pool sizes. The sizes range from 75% to 200%, in equal intervals of the current shared pool size, and are not configurable. If rows in the V\$SHARED_POOL_ADVICE view have the same values of parse time savings (given in the ESTD_LC_TIME_SAVED column), this indicates that there would be no additional hits on those size ranges for library cache objects. However, if time saving values increase for larger pool sizes, this indicates that it may help to increase the shared pool size.

Note: The output in the slide above is truncated.

- The V\$SHARED_POOL_ADVICE view should be the first tool to use when sizing the shared pool. If the advisory indicates that a larger pool is not useful for the library cache memory objects, you can drill down to see whether changing the SQL (to improve use of shared cursors) or other activity (such as deferring DDL to off hours) improves performance. This information is easily accessed in the AWR report or Statspack report.
- The V\$LIBRARY_CACHE_MEMORY view displays information about memory allocated to library cache memory objects in different namespaces. A memory object is an internal grouping of memory for efficient management. A library cache object may consist of one or more memory objects.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

Shared Pool Advisory in an AWR Report

Shared Pool Advisory

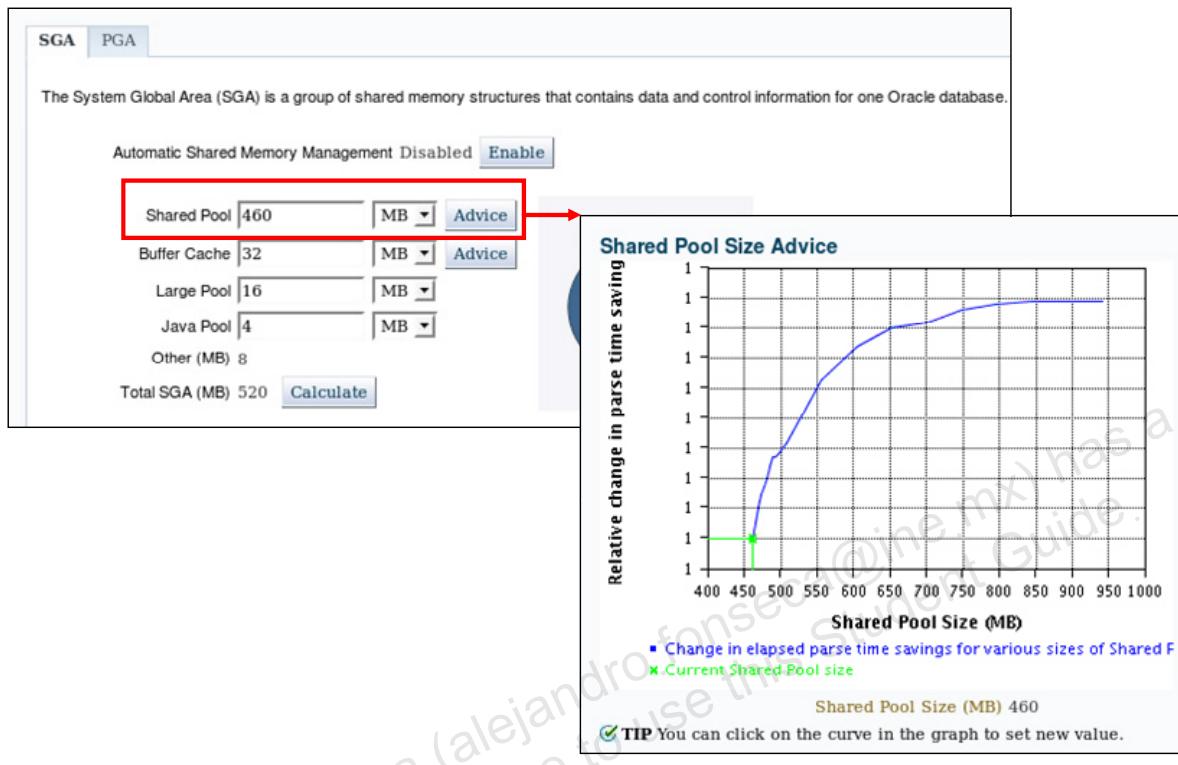
- SP: Shared Pool Est LC: Estimated Library Cache Factor
- Note there is often a 1:Many correlation between a single logical object in the Library Cache, and the physical number of memory objects associated with it. Therefore comparing the number of Lib Cache objects (e.g. in v\$librarycache), with the number of Lib Cache Memory Objects is invalid.

Shared Pool Size(M)	SP Size Fctr	Est LC Size (M)	Est LC Mem Obj	Est LC Time Saved (s)	Est LC Time Saved Fctr	Est LC Load Time (s)	Est LC Load Time Fctr	Est LC Mem Obj Hits (K)
200	1.00	18	1,806	1,610	1.00	118	1.00	684
204	1.02	21	1,834	1,612	1.00	116	0.98	685
208	1.04	25	1,916	1,612	1.00	116	0.98	685
212	1.06	28	2,001	1,612	1.00	116	0.98	685
216	1.08	32	2,274	1,612	1.00	116	0.98	685
220	1.10	36	2,523	1,612	1.00	116	0.98	685
224	1.12	40	2,775	1,612	1.00	116	0.98	685
228	1.14	44	3,050	1,612	1.00	116	0.98	685
232	1.16	48	3,325	1,612	1.00	116	0.98	685
236	1.18	52	3,600	1,612	1.00	116	0.98	685
240	1.20	56	3,875	1,612	1.00	116	0.98	685
260	1.30	76	5,324	1,612	1.00	116	0.98	685
280	1.40	96	6,868	1,612	1.00	116	0.98	686
300	1.50	116	8,419	1,612	1.00	116	0.98	686
320	1.60	136	9,975	1,613	1.00	115	0.97	686

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Statspack and AWR reports also present the shared pool advisory. The example advisory indicates that there would be some very small improvements gained by increasing the shared pool size over the current size of 200 M. The Shared Pool Advisory in the slide is truncated to fit the slide.

Shared Pool Advisor



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can access the Memory Advisors page by expanding the Performance menu and selecting Memory Advisors. On the Memory Advisors page, you can access the Shared Pool Advisor by clicking Advice next to the Shared Pool field. The Shared Pool Advisor is available only if Automatic Memory Management and Automatic Shared Memory Management are disabled.

Avoiding Fragmentation

Avoid fragmentation by:

- Keeping frequently required large objects
- Reserving space for large objects
- Eliminating large anonymous PL/SQL blocks
- Enabling the use of the large pool



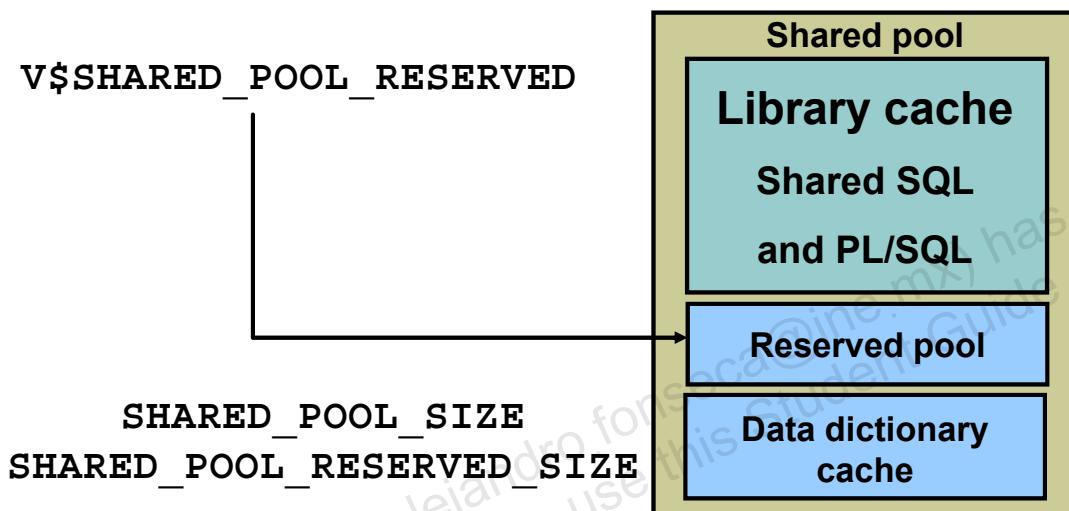
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Reduce memory fragmentation by:

- Keeping frequently required large objects such as SQL and PL/SQL areas in memory, instead of aging them out with the normal LRU mechanism by using the DBMS_SHARED_POOL.KEEP procedure.
- Ensuring availability of contiguous space for large memory requirements through the allocation of reserved space in the shared pool area
- Using small PL/SQL packaged functions instead of large anonymous blocks
- Configuring the large pool. If the large pool is configured, it is used with:
 - Oracle Shared Server connections
 - Parallel query operations
 - RMAN parallel backup and recovery

Large Memory Requirements

- Satisfy requests for large contiguous memory
- Reserve contiguous memory within the shared pool



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A large object and a large allocation are not equivalent. Most large objects are made up of 4 KB chunks. But some allocations, such as session memory allocated at the start of each session, may require 30 KB. A small percentage of allocations exceed 4 KB, but hundreds of objects of 4 KB chunks could be freed without satisfying the request.

Why Reserve Space in the Shared Pool?

A small amount of space is reserved for large allocations; by default, this is 5% of the shared pool size. When an allocation for more than 4 KB is requested, the request will be satisfied out of the shared pool if possible, and then from the reserved space. Smaller allocations are not allowed in the reserved pool. This means that the shared pool reserved pool has large contiguous chunks of memory. The DBA can change the size of this reserved memory to satisfy large allocations during operations such as PL/SQL and trigger compilation.

Initialization Parameter

The size of the shared pool reserved pool is set by the **SHARED_POOL_RESERVED_SIZE** initialization parameter. This parameter specifies how much of the value of **SHARED_POOL_SIZE** to reserve for large allocations. The default value should be adequate for most installations. If **SHARED_POOL_RESERVED_SIZE** is greater than half of **SHARED_POOL_SIZE**, the server signals an error.

The V\$SHARED_POOL_RESERVED View

This view helps in tuning the reserved pool within the shared pool. The first set of columns of the view is valid only if the SHARED_POOL_RESERVED_SIZE parameter is set to a valid value.

SQL> desc V\$SHARED_POOL_RESERVED

Name	Null?	Type
FREE_SPACE		NUMBER
AVG_FREE_SIZE		NUMBER
FREE_COUNT		NUMBER
MAX_FREE_SIZE		NUMBER
USED_SPACE		NUMBER
AVG_USED_SIZE		NUMBER
USED_COUNT		NUMBER
MAX_USED_SIZE		NUMBER
REQUESTS		NUMBER
REQUEST_MISSES		NUMBER
LAST_MISS_SIZE		NUMBER
MAX_MISS_SIZE		NUMBER

The following columns contain values that are valid even if the parameter is not set:

REQUEST_FAILURES	NUMBER
LAST_FAILURE_SIZE	NUMBER
ABORTED_REQUEST_THRESHOLD	NUMBER
ABORTED_REQUESTS	NUMBER
LAST_ABORTED_SIZE	NUMBER
CON_ID	NUMBER

Where:

- FREE_SPACE: Total free space in the reserved pool
- AVG_FREE_SIZE: Average size of the free memory in the reserved pool
- MAX_FREE_SIZE: Size of the largest free piece of memory in the reserved pool
- REQUEST_MISSES: The number of times the reserved pool did not have a free piece of memory to satisfy the request and flushed objects from the LRU pool
- REQUEST_FAILURES: The number of times that no memory was found to satisfy a request
- LAST_FAILURE_SIZE: The size of the last failed request
- ABORTED_REQUEST_THRESHOLD: Minimum size of a request that signals an ORA-04031 error without flushing objects
- ABORTED_REQUESTS: Number of requests that signaled an ORA-04031 error without flushing objects
- LAST_ABORTED_SIZE: Last size of the request that returned an ORA-04031 error without flushing objects from the LRU list

Tuning the Shared Pool Reserved Pool

IF REQUEST_FAILURES			Action
>0 and increasing	AND	REQUEST_MISSES > 0	Increase SHARED_POOL_RESERVED_SIZE
>0 and increasing	AND	FREE_SPACE => 50% of SHARED_POOL_RESERVED_SIZE	Increase SHARED_POOL_SIZE
=0	OR	FREE_SPACE => 50% of SHARED_POOL_RESERVED_SIZE	Decrease SHARED_POOL_RESERVED_SIZE



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Diagnostics with the V\$SHARED_POOL_RESERVED View

Statistics from the V\$SHARED_POOL_RESERVED view can help you to set the SHARED_POOL_RESERVED_SIZE parameter. On a system with ample free memory to increase the SGA, the goal is to have REQUEST_MISSES equal 0, so as not to have any request failures or at least to prevent this value from increasing.

SHARED_POOL_RESERVED_SIZE Too Small

When REQUEST_FAILURES is greater than zero and increasing, the reserved pool is too small. REQUEST_MISSES indicates the number of allocation requests that could not be satisfied. To resolve this, you can increase the SHARED_POOL_RESERVED_SIZE and SHARED_POOL_SIZE values accordingly. Increase the size of both the parameters by the same amount because the reserved pool comes out of the shared pool. The settings you select for these parameters depend on the SGA size constraints of your system. This option increases the amount of memory available on the reserved pool without having an effect on users who do not allocate memory from the reserved pool.

SHARED_POOL_SIZE Too Small

The V\$SHARED_POOL_RESERVED view can also indicate when the value for SHARED_POOL_SIZE is too small. This may be the case if REQUEST_FAILURES > 0 is increasing and FREE_SPACE is a large portion of SHARED_POOL_RESERVED_SIZE and not decreasing. This indicates that the requests are below the 4 KB threshold. If you have enabled the reserved area, decrease the value for SHARED_POOL_RESERVED_SIZE. If you have not enabled the reserved area, you can increase SHARED_POOL_SIZE.

SHARED_POOL_RESERVED_SIZE Too Large

Too much memory may have been allocated to the reserved pool if:

- REQUEST_MISSES = 0 or is not increasing. The goal is to have this value remain constant using the smallest amount of memory for the shared pool reserved pool.
- FREE_SPACE = > 50% of the SHARED_POOL_RESERVED_SIZE

If either of these is true, decrease the value for SHARED_POOL_RESERVED_SIZE.

Limiting the Impact of Loading Large Objects

Usually, if a request cannot be satisfied on the free list, then the RDBMS tries to reclaim memory by freeing objects from the LRU list and checking periodically to see if the request can be fulfilled. After finishing this step, the RDBMS has performed a near equivalent of an 'ALTER SYSTEM FLUSH SHARED_POOL'.

Because this impacts all users on the system, this procedure "localizes" the impact to the process failing to find a piece of shared pool memory of size greater than the threshold size. This user gets the 'out of memory' error without attempting to search the LRU list.

The ABORTED_REQUEST_THRESHOLD procedure in the DBMS_SHARED_POOL package sets the minimum size of a request that reports an ORA-4031 error without trying to flush objects and free memory. This procedure limits the extent of a flush that could occur due to a large allocation.

Note: With Automatic Shared Memory Management, an attempt to allocate another granule of memory will be made before reporting an ORA-4031 error.

Keeping Large Objects

- Find PL/SQL objects that are not kept in the library cache:

```
SQL> SELECT * FROM v$db_object_cache
  2 WHERE sharable_mem > 10000
  3 AND (type='PACKAGE' OR type='PACKAGE BODY' OR
  4       type='FUNCTION' OR type='PROCEDURE')
  5 AND kept='NO';
```

- Pin large packages in the library cache:

```
SQL> EXECUTE DBMS_SHARED_POOL.KEEP('package_name');
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The loading of large objects is the primary source of shared pool memory fragmentation. User response time increases when a large number of small objects need to be aged out of the shared pool to make room. To prevent these situations, keep these large or frequently required objects in the shared pool to make sure that they are never aged out of the shared pool.

Which objects to keep:

- Frequently required large procedural objects such as STANDARD and DIUTIL packages, and those for which sharable memory exceeds a defined threshold
- Compiled triggers that are executed often on frequently used tables
- Sequences, because cached sequence numbers are lost when the sequence is aged out of the shared pool

When to keep them: Startup time is best, because that prevents further fragmentation.

Objects marked as KEPT are not removed when flushing the shared pool by using the ALTER SYSTEM FLUSH SHARED_POOL command.

How to Keep Objects

Use the DBMS_SHARED_POOL package and the KEEP procedure to keep objects.

To create the package, run the dbmspool.sql script. The prvtpool.plb script is automatically executed at the end of the dbmspool.sql script. These scripts are run by catproc.sql as part of a default database creation.

Use the UNKEEP procedure to unpin objects from the object cache.

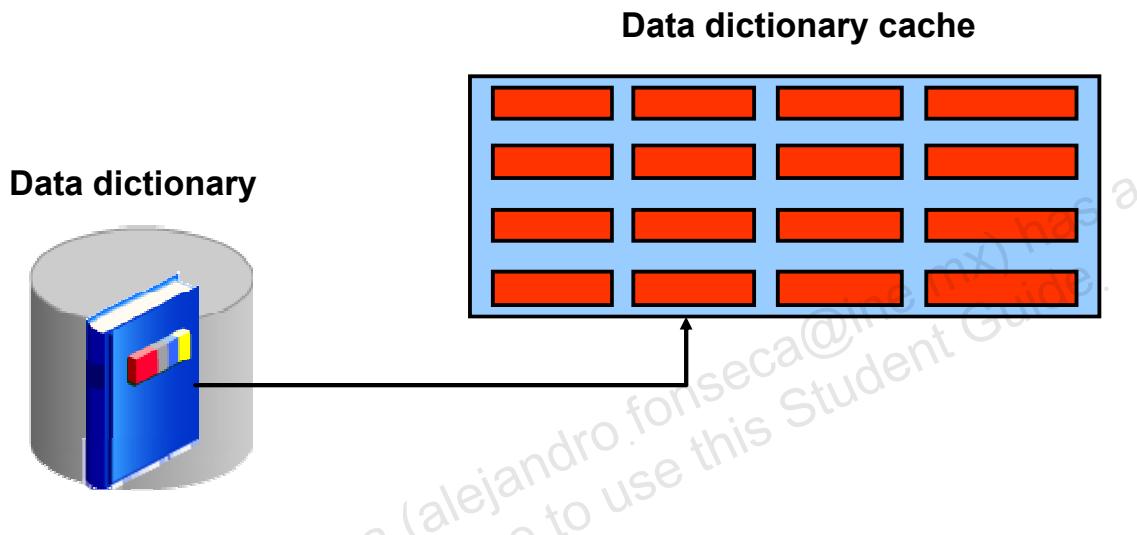
Considerations

Large objects that are constantly being used do not age out of the shared pool. The objects that are used frequently but are unused long enough, age out. These objects may occasionally raise an ORA-4031 error when loading because of fragmentation, or users may complain about the command that calls these objects as being slower. By executing the query shown in the slide periodically, you can choose candidate objects. These are objects that always appear in the object cache. Keeping an object that never ages out does not affect the performance.

Keeping objects in the shared pool prevents fragmentation, but it also reduces the available memory for re-creatable objects. After keeping objects in the shared pool, check the reloads and hard parses. SHARED_POOL_SIZE may need to be increased.

Data Dictionary Cache

The data dictionary cache holds row images of data dictionary rows.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The data dictionary cache or row cache holds copies of some of the rows of the data dictionary tables. The row cache is divided into sections, such as segment, extent, histogram, and sequence, depending on the object. Statistics are gathered on the usage of each section and reported in V\$ROWCACHE.

Three key columns of V\$ROWCACHE are:

- PARAMETER: Gives the name of the data dictionary cache that is being reported
- GETS: Shows the total number of requests for information about the corresponding item
- GETMISSES: Shows the number of data requests resulting in cache misses

Misses on the row cache are not as expensive as misses in the library cache. A row cache miss results only in a row fetch from the data dictionary. Misses on the data dictionary cache are to be expected in some cases. Upon instance startup, the data dictionary cache contains no data, so any SQL statement issued will result in a cache miss. As more data is read into the cache, the number of cache misses should decrease. Eventually, the database should reach a “steady state” in which the most frequently used dictionary data is in the cache. At this point, very few cache misses should occur. To tune the cache, examine its activity only after your application has been running for some time.

Dictionary Cache Misses

Cache	Get Requests	Pct Miss	Scan Reqs	Pct Miss	Mod Reqs	Final Usage
dc_awr_control	45	20.00	0		2	5
dc_files	204	25.00	0		0	17
dc_global_oids	26,289	0.31	0		0	58
dc_histogram_data	58,345	5.23	0		0	1,768
dc_histogram_defs	371,644	1.76	0		798	3,210
dc_object_grants	10,399	0.70	0		0	32
dc_objects	2,112,536	0.09	0		91	1,156
dc_profiles	3,095	0.03	0		0	1
dc_props	3,337	0.06			0	2
dc_rollback_segments	496	0.00	0		0	73
dc_segments	415,721	0.93	0		1	1,168
dc_sequences	72	18.06	0			
dc_tablespace_quotas	12	33.33	0			

If there are too many cache misses, increase the value of the SHARED_POOL_SIZE parameter.



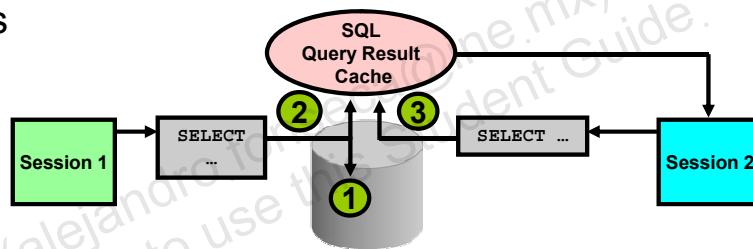
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In most cases, the dictionary cache does not need tuning if the library cache is properly tuned. Pct Miss should be less than 2% of Get Requests if there are a large number of requests. For example, in the slide, dc_sequences has a Pct Miss of 18%, but only 72 requests. This is not a concern. dc_objects is a typical row with a large number of requests and a small Pct Miss. If the Dictionary Cache section of the AWR or Statspack report indicates a high Pct Miss and a high Get Requests for a number of caches, SHARED_POOL_SIZE should be increased.

Resizing the shared pool is not always necessary. Large OLTP systems, where users log in to the database using their own user ID, can benefit from explicitly qualifying the segment owner, rather than using public synonyms. This significantly reduces the number of entries in the dictionary cache. Also, an alternative to qualifying table names is to connect to the database through a single user ID, rather than individual user IDs. User-level validation can take place locally on the middle tier. Reducing the number of distinct user IDs also reduces the load on the dictionary cache.

SQL Query Result Cache: Overview

- Cache the result of a query or query block for future reuse.
- Cache is used across statements and sessions unless it is stale.
- Benefits:
 - Scalability
 - Reduction of memory usage
- Good candidate statements:
 - Access many rows
 - Return few rows



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The SQL query result cache enables explicit caching of query result sets and query fragments in database memory. This area in the shared pool can be used for storing and retrieving the cached results. The query results stored in this cache become invalid when data in the database objects being accessed by the query is modified.

Although the SQL query cache can be used for any query, good candidate statements are the ones that need to access a very high number of rows to return only a fraction of them. This is mostly the case for data warehousing applications.

If the first session executes a query, it retrieves the data from the database and then caches the result in the SQL query result cache. If a second session executes the exact same query, it retrieves the result directly from the cache instead of using the disks.

Note

- Each node in a RAC configuration has a private result cache. Results cached on one instance cannot be used by another instance. However, invalidations work across instances. To handle all synchronization operations between RAC instances related to the SQL query result cache, the special RCBG process is used on each instance.
- With parallel query, an entire result can be cached (in RAC, it is cached on query coordinator instance) but individual parallel query processes cannot use the cache.

Managing the SQL Query Result Cache

Use the following initialization parameters:

- RESULT_CACHE_MODE values:
 - MANUAL: Use the RESULT_CACHE hint to specify results to be stored in the cache. This is the default value.
 - FORCE: All results are stored in the cache.
- RESULT_CACHE_MAX_SIZE
 - Sets the memory allocated to the result cache
 - Disabled if value is 0.
 - Limited to 75% of shared pool
 - Default is derived from other memory settings
 - 0.25% of MEMORY_TARGET or
 - 0.5% of SGA_TARGET or
 - 1% of SHARED_POOL_SIZE



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can alter various parameter settings in the initialization parameter file to manage the SQL query result cache of your database. The query optimizer manages the result cache mechanism depending on the settings of the RESULT_CACHE_MODE parameter in the initialization parameter file.

You can use this parameter to determine whether or not the optimizer automatically sends the results of queries to the result cache. You can set the RESULT_CACHE_MODE parameter at the system and session levels. The possible parameter values are MANUAL and FORCE:

- When set to MANUAL (the default), you must specify, by using the RESULT_CACHE hint, that a particular result is to be stored in the cache.
- When set to FORCE, all results are stored in the cache. If the statement contains a NO_RESULT_CACHE hint, then the hint takes precedence over the parameter setting.

By default, the database server allocates memory for the SQL query result cache in the shared pool inside the SGA. The memory size allocated to the result cache depends on the memory size of the SGA as well as the memory management system. You can change the memory allocated to the result cache by setting the RESULT_CACHE_MAX_SIZE parameter. The result cache is disabled if you set its value to 0. The value of this parameter is rounded to the largest multiple of 32 KB that is not greater than the specified value. If the rounded value is 0, then the feature is disabled.

Managing the SQL Query Result Cache

Use the following initialization parameters:

- RESULT_CACHE_MAX_RESULT
 - Sets maximum cache memory for a single result
 - Defaults to 5% of RESULT_CACHE_MAX_SIZE
- RESULT_CACHE_REMOTE_EXPIRATION
 - Sets the expiry time for cached results depending on remote database objects
 - Defaults to 0



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Other initialization parameters also control the results cache behavior.

Use the RESULT_CACHE_MAX_RESULT parameter to specify the maximum amount of cache memory that can be used by any single result. The default value is 5%, but you can specify any percentage value between 1 and 100. This parameter can be implemented at the system and session level.

Use the RESULT_CACHE_REMOTE_EXPIRATION parameter to specify the time (in number of minutes) for which a result that depends on remote database objects remains valid. The default value is 0, which implies that results using remote objects should not be cached. Setting this parameter to a nonzero value can produce stale answers—for example, if the remote table used by a result is modified at the remote database.

Using the RESULT_CACHE Hint

```
EXPLAIN PLAN FOR
SELECT /*+ RESULT_CACHE */ department_id, AVG(salary)
FROM employees
GROUP BY department_id;
```

Id	Operation	Name	Rows
0	SELECT STATEMENT		11
1	RESULT CACHE	8fpza04gtwsfr6n595au15yj4y	
2	HASH GROUP BY		11
3	TABLE ACCESS FULL	EMPLOYEES	107

```
SELECT /*+ NO_RESULT_CACHE */ department_id, AVG(salary)
FROM employees
GROUP BY department_id;
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

If you want to use the query result cache and the RESULT_CACHE_MODE initialization parameter is set to MANUAL, then you must explicitly specify the RESULT_CACHE hint in your query. This introduces the ResultCache operator into the execution plan for the query. When you execute the query, the ResultCache operator looks up the result cache memory to check whether the result for the query already exists in the cache. If it exists, the result is retrieved directly out of the cache. If it does not yet exist in the cache, the query is executed, the result is returned as output, and is also stored in the result cache memory.

If the RESULT_CACHE_MODE initialization parameter is set to FORCE, and you do not want to store the result of a query in the result cache, you must use the NO_RESULT_CACHE hint in your query. For example, when the RESULT_CACHE_MODE value equals FORCE in the initialization parameter file, and you do not want to use the result cache for the EMPLOYEES table, use the NO_RESULT_CACHE hint.

Note: Use of the [NO_] RESULT_CACHE hint takes precedence over the parameter settings.

Using Table Annotation to Control Result Caching

- Use the RESULT_CACHE clause to control result caching:
 - MODE DEFAULT
 - MODE FORCE
- Statement hints can be used to override the setting.
- RESULT_CACHE column of the DBA_, ALL_, and USER_TABLES displays result cache mode.

```
CREATE TABLE sales (...) RESULT_CACHE (MODE DEFAULT);
ALTER TABLE sales RESULT_CACHE (MODE FORCE);
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can use table annotations to control result caching. Table annotations are in effect only for the whole query, not for query segments. The primary benefit of this feature is eliminating the need of adding result cache hints to queries at the application level.

RESULT_CACHE table annotation values are as follows:

- DEFAULT: If at least one table in a query is set to DEFAULT, result caching is not enabled at the table level for this query, unless the RESULT_CACHE_MODE initialization parameter is set to FORCE or the RESULT_CACHE hint is specified. This is the default value.
- FORCE: If all the tables of a query are marked as FORCE, the query result is considered for caching. The FORCE table annotation takes precedence over the RESULT_CACHE_MODE parameter value of MANUAL set at the session level.

A table annotation has a lower precedence than a SQL hint. You can override table and session settings by using hints at the query level.

Query the RESULT_CACHE column of the DBA_, ALL_, and USER_TABLES data dictionary views to determine the result cache mode of the table.

Using the DBMS_RESULT_CACHE Package

Use the DBMS_RESULT_CACHE package to:

- Manage memory allocation for the query result cache
- View the status of the cache:

```
SELECT DBMS_RESULT_CACHE.STATUS FROM DUAL;
```

- Retrieve statistics on the cache memory usage:

```
EXECUTE DBMS_RESULT_CACHE.MEMORY_REPORT;
```

- Remove all existing results and clear cache memory:

```
EXECUTE DBMS_RESULT_CACHE.FLUSH;
```

- Invalidate cached results depending on specified object:

```
EXEC DBMS_RESULT_CACHE.INVALIDATE('JFV','MYTAB');
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The DBMS_RESULT_CACHE package provides statistics, information, and operators that enable you to manage memory allocation for the query result cache. You can use the DBMS_RESULT_CACHE package to perform various operations such as viewing the status of the cache (OPEN or CLOSED), retrieving statistics on the cache memory usage, and flushing the cache. For example, to view the memory allocation statistics, use the following SQL procedure:

```
SQL> set serveroutput on
SQL> execute dbms_result_cache.memory_report
R e s u l t   C a c h e   M e m o r y   R e p o r t
[Parameters]
Block Size      = 1024 bytes
Maximum Cache Size = 720896 bytes (704 blocks)
Maximum Result Size = 35840 bytes (35 blocks)
[Memory]
Total Memory = 46284 bytes [0.036% of the Shared Pool]
... Fixed Memory = 10640 bytes [0.008% of the Shared Pool]
... State Object Pool = 2852 bytes [0.002% of the Shared Pool]
... Cache Memory = 32792 bytes (32 blocks) [0.025% of the Shared Pool]
..... Unused Memory = 30 blocks
..... Used Memory = 2 blocks
..... Dependencies = 1 blocks
..... Results = 1 blocks
..... SQL = 1 blocks
```

Note: For more information, refer to the *PL/SQL Packages and Types Reference Guide*.

Viewing SQL Result Cache Dictionary Information

The following views provide information about the query result cache:

(G) V\$RESULT_CACHE_STATISTICS	Lists the various cache settings and memory usage statistics
(G) V\$RESULT_CACHE_MEMORY	Lists all the memory blocks and the corresponding statistics
(G) V\$RESULT_CACHE_OBJECTS	Lists all the objects (cached results and dependencies) along with their attributes
(G) V\$RESULT_CACHE_DEPENDENCY	Lists the dependency details between the cached results and dependencies



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Note: For further information, see the *Oracle Database Reference* guide.

SQL Query Result Cache: Considerations

- Result cache is disabled for queries containing:
 - Temporary or dictionary tables
 - Non-deterministic PL/SQL functions
 - Sequence CURRVAL and NEXTVAL
 - SQL functions CURRENT_DATE, SYSDATE, SYS_GUID, and so on
- DML or DDL on remote database does not expire cached results.
- Flashback queries can be cached.
- Result cache does not automatically release memory.
 - It grows until maximum size is reached.
 - DBMS_RESULT_CACHE.FLUSH purges memory.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In addition to the restrictions in the slide:

When the statements with bind variables are cached:

- The cached result is parameterized with variable values.
- The cached results can be found only for the same variable values. That is, different values or bind variable names cause cache miss.

Cached result will not be built if:

- The query is built on a noncurrent version of data (read consistency enforcement)
- The current session has outstanding transactions on tables in query

Note

- Any user-written function used in a function-based index must have been declared with the DETERMINISTIC keyword to indicate that the function will always return the same output value for any given set of input argument values.
- The purge works only if the cache is not in use; disable (close) the cache for flush to succeed.

Quiz

The symptoms of an undersized shared pool are:

- a. Contention for buffer blocks
- b. Excessive reloads in the SQL area
- c. Waits for log file sync requests
- d. Large values for parse time elapsed in the time model
- e. DB file sequential reads in the top timed events



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: b, d

Excessive reloads can be caused by cursors being aged out of the pool when the space is required for new cursors. When the cursors are being aged out and the application tries to use them again the cursor must be re-parsed requiring more parse time.

Summary

In this lesson, you should have learned how to:

- Diagnose and resolve hard-parsing problem
- Diagnose and resolve soft-parsing problem
- Size the shared pool
- Diagnose and resolve shared pool fragmentation
- Keep objects in the shared pool
- Size the reserved area
- Manage the SQL query result cache



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Practice 16: Tuning the Shared Pool

This practice covers:

- Sizing the Shared Pool
- Tuning a Hard-Parse Workload
- Keeping Objects in the Shared Pool

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2018, Oracle and/or its affiliates.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

17

Tuning the Buffer Cache

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

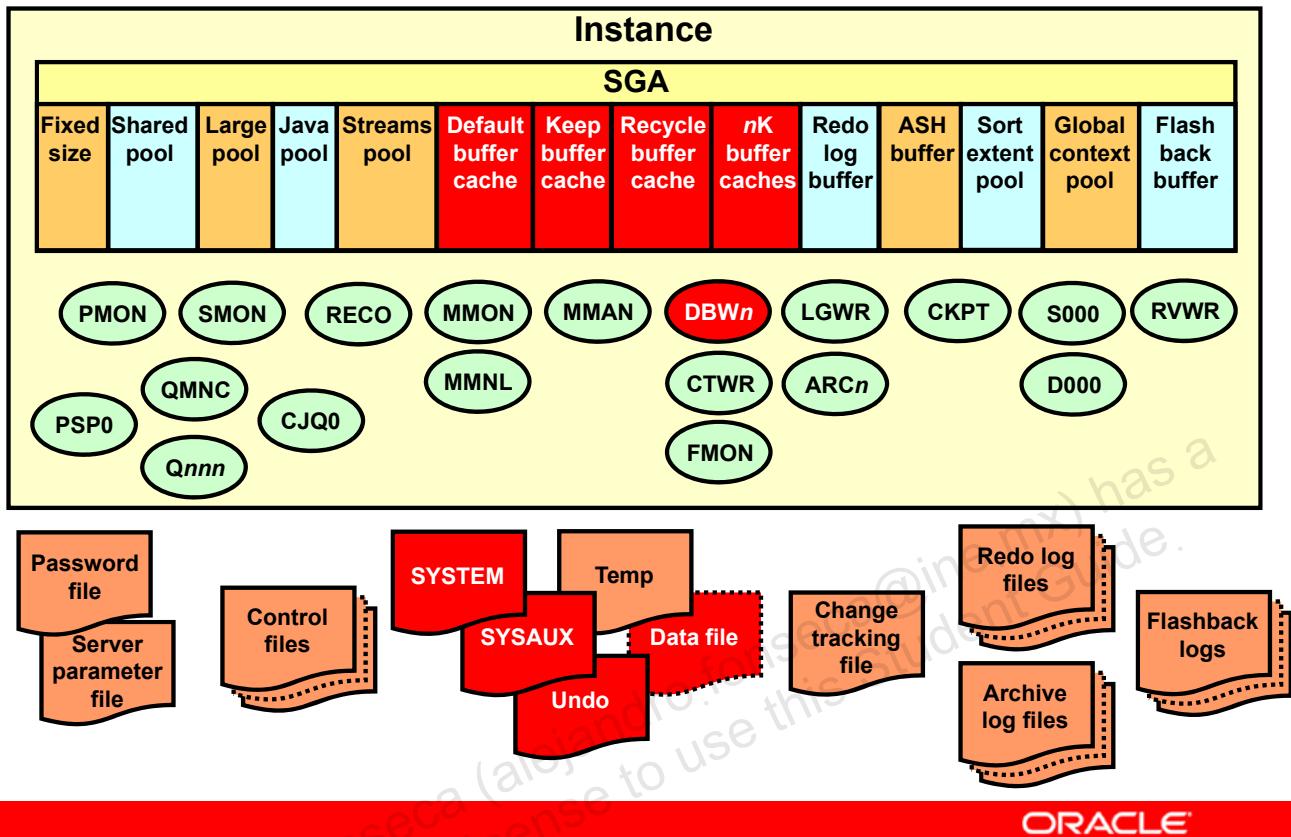
After completing this lesson, you should be able to do the following:

- Describe the buffer cache architecture
- Size the buffer cache
- Resolve common performance issues related to the buffer cache
- Use common diagnostic indicators to suggest a possible solution



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Oracle Database Architecture



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Buffer Cache: Highlights

- Scalable architecture:
 - Multiversion concurrency control
 - Proprietary LRU-based replacement policy
 - Cache fusion
- Incremental checkpointing mechanism
- Advisors
- Private pool for I/O-intensive operations

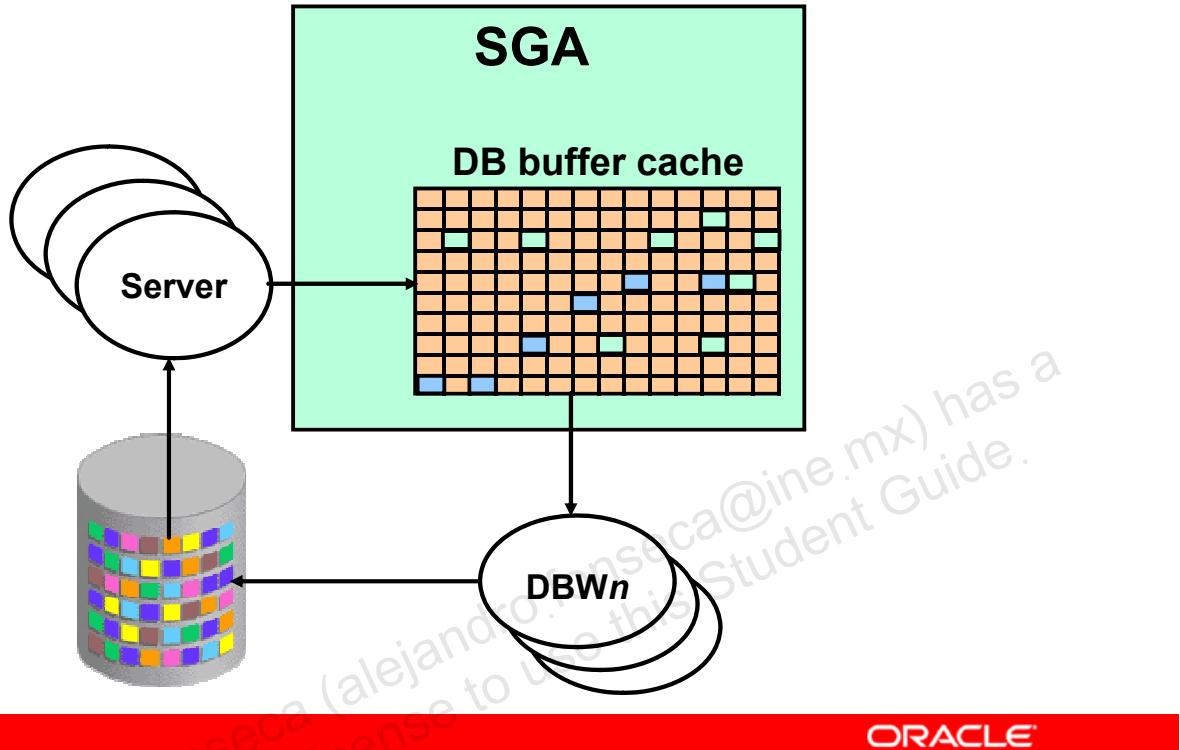


Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The buffer cache is an extremely important component for the performance, scalability, and reliability of the database. Some of the highlights of the buffer cache architecture include the following:

- The least recently used (LRU)-like replacement algorithm provides excellent hit rates on a wide range of workload with little overhead.
- A multiversion concurrency control algorithm makes it possible for readers and writers to access different versions of the same block at the same time.
- Cache fusion is a distributed cache coherency protocol that allows the Oracle database to transparently run on a cluster of machines.
- Incremental checkpointing is a mechanism that tries to maintain a smooth I/O rate for achieving predictable recovery time.
- Where appropriate, the Buffer Cache Advisor assists in configuration and eliminates trial-and-error in performance tuning.
- Direct-path I/O operations use private buffer pools and asynchronous prefetch mechanisms to achieve high bandwidths for decision support systems (DSS) and scan-oriented workloads.

Database Buffers



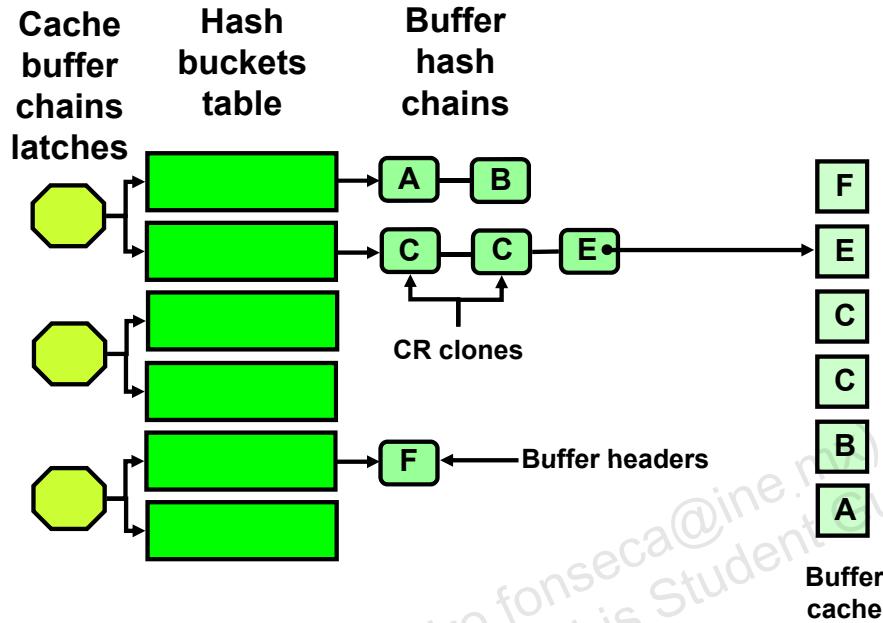
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The database buffer cache is a set of memory buffers in the System Global Area (SGA). Each buffer is sized to hold one database block. The default size of a database block is set at database creation to the size specified in the `DB_BLOCK_SIZE` parameter. Server processes are responsible for moving blocks into the buffers and the `DBWn` processes are responsible for writing blocks that have changed out to the data files.

Every buffer can be in one of three states: free, dirty, or pinned. A free buffer is either empty or identical to the block on disk. A free buffer is available to be overwritten with a different database block at any time. A dirty buffer holds a block that has changed; a server process has made some modification to this block. A pinned block is being accessed by a process. Only one process at a time is allowed to access a buffer to write. A pin for read can be shared. The pin is a very short operation and is not related to a lock. When another process tries to access a buffer that is pinned, a buffer busy wait is recorded.

There may be several versions of a block in different buffers of the cache at the same time. When a block is changed by a DML statement, the change is immediately written to the block in memory. A `SELECT` statement requires a view of that block consistent with all committed changes. A Consistent Read (CR) clone block is created by copying the block to another buffer and applying undo data, to make it consistent with the time the `SELECT` statement started.

Buffer Hash Table for Lookups



ORACLE

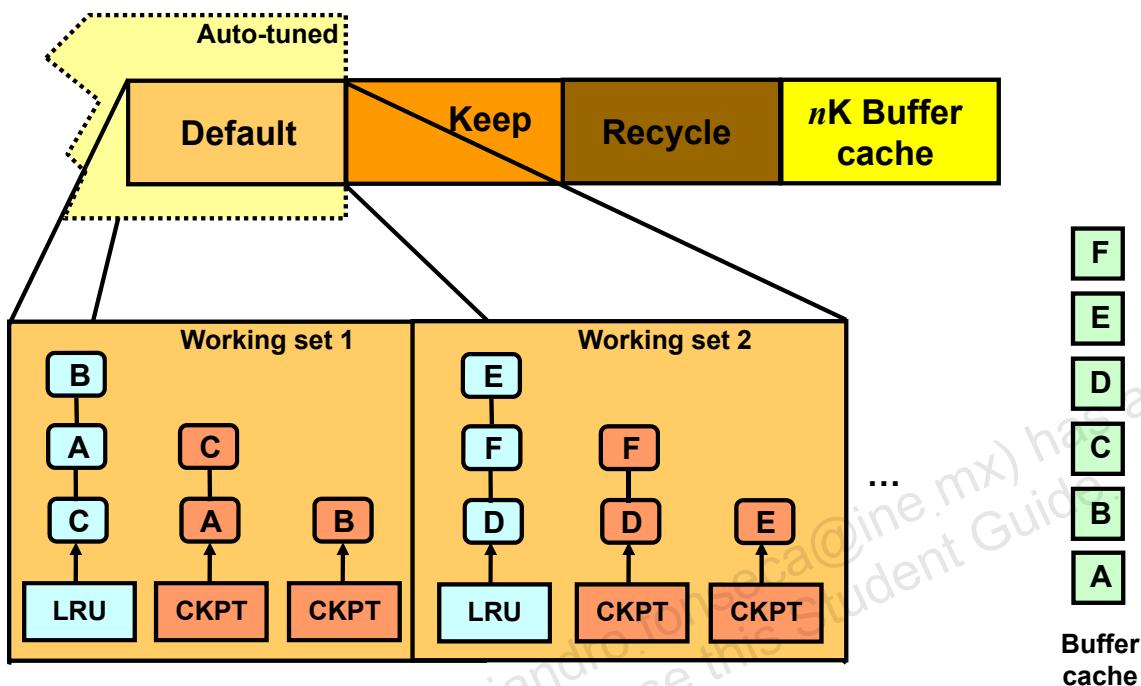
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The buffer hash table facilitates buffer lookup. When a server process needs to access a data block in the cache, it requests the block by the data block address (DBA), which is the physical address of a database block. The data block address is hashed to produce an index into a table of hash buckets. For scalability, multiple hash latches protect access to this hash table. These latches are referred to as cache buffer chains latch in V\$LATCH. The hash index is hashed to produce the index of the corresponding latch. As depicted in the slide, each latch protects several buckets, and each hash bucket contains a linked list of all the buffers whose data block addresses produce the same hash value. Consistent Read copies (CR clones) of the same block all have the same data block address; thus, they are all in the same hash chain. The lookup algorithm returns either the current buffer for the requested data block address or an existing CR clone, or a new CR clone created through the Consistent Read mechanism.

Readers acquire the hash latch in shared mode so that all readers can concurrently scan for buffers. Only when a process needs to modify the hash chain it needs to acquire the latch in exclusive mode.

Note: The hashing algorithm provides a very good distribution of blocks across buckets with a very low probability of systematic collisions.

Working Sets



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Working sets are subsets of buffers in the buffer cache that are created to improve scalability of common buffer cache operations. Each working set contains a least recently used (LRU) list and two checkpoint queues, as illustrated in the slide.

Note: Hash buffer chains for buffer lookup is independent of the LRU list. Each buffer in the cache is assigned to a working set at instance startup. The number of working sets is $\frac{1}{2}$ number of CPUs for each buffer cache (DEFAULT, KEEP, etc.). Concurrent operations by processes on different CPUs can proceed without conflicting with one another by using different working sets.

A *cache miss* occurs when a requested block is not found in the buffer cache. When this happens, an available buffer must be found by searching the LRU list. The LRU list is a linked list of buffer headers arranged by time since the last access and frequency of access. The frequency of access is measured by the number of times a buffer has been accessed. This is called touch count. A buffer that is frequently accessed and has been recently accessed is called a hot buffer or block. A buffer that has not been accessed recently is a cold buffer or block. The LRU list is arranged so that the cold buffers are at one end and hot buffers are at the other. Accesses to the LRU lists are protected by the LRU latch known as the *cache buffer lru chains latch* in *V\$Latch*. There is one latch per working set. On a cache miss, a user process picks a random working set and overwrites the coldest buffer in that set with the copy read from disk.

The replacement policy is not strictly LRU based because this would require a manipulation of the list each time a buffer is accessed. In fact, a touch count mechanism is used for every buffer in the list, and a high touch count buffer is moved from the cold part to the hot part only during a replacement operation. This approximate LRU scheme is considerably faster while producing comparable cache hit rates.

To allow buffer replacements to occur, it is necessary to write cold dirty buffers that have aged out to the cold part of the list. Such writes are referred to as aging writes and are performed by DBW n processes. The DBW n processes work to write enough cold buffers to guarantee a uniform supply of free buffers for replacement purposes. However, because a hot buffer located in the hot part can have the first change in the redo log, aging writes might not be able to advance the thread checkpoint used for crash recovery purposes. That is why another list called the checkpoint queue, which orders buffers in lowest redo block address (RBA) order, is used in addition to the LRU list. Each working set contains two checkpoint queues, each protected by a separate checkpoint queue latch of type checkpoint queue. This allows user processes to add buffers to one list while DBW n is writing buffers from the other list.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

Tuning Goals and Techniques

- Recommendation: Use Automatic Memory Management (AMM).
- Tuning goals:
 - Servers find data in memory.
 - No waits on the buffer cache
- Diagnostic and monitoring measures:
 - Verify ADDM recommendations.
 - Use Statspack or AWR.
 - Use the `V$DB_CACHE_ADVICE` view.
- Tuning techniques:
 - Implement ADDM recommendations.
 - Reduce the number of blocks required by SQL statements.
 - Increase the buffer cache size; use multiple buffer pools.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Oracle recommends automatic memory configuration for your system by using the `MEMORY_MAX` and `MEMORY_TARGET` initialization parameters to control the entire memory usage of the Oracle instance. However, you can manually adjust the memory pools on your system. The manual tuning process is discussed later in this lesson. Automatic memory management is discussed in the lesson titled “Automatic Memory Management.”

Tuning Goals

Because physical I/O takes significant time and CPU, Oracle server performance is improved when the servers find the blocks that they need in memory. The cache hit ratio measures the performance of the database buffer cache. The cache hit ratio is the ratio of the number of blocks found in memory to the number of blocks accessed. A low cache hit ratio indicates that the system may be slow because it is performing unnecessary I/Os.

Diagnostic and Monitoring Measures

To effectively monitor the usage of the buffer cache, use the following mechanisms:

- Verify and implement ADDM recommendations.
- Use the Statspack or AWR utilities, to monitor buffer cache-related events.
- Use the `V$DB_CACHE_ADVICE` view.

Tuning Techniques

If you are using Automatic Memory Management (AMM) or Automatic Shared Memory Management (ASMM), you should review the recommendations from Automatic Database Diagnostic Monitor for modifying the size of the total allocated memory or total shared memory.

If manual SGA tuning is in use, you monitor the buffer cache with the information provided in the AWR or Statspack reports. Be sure to review the top timed events related to the buffer cache.

The first step in tuning the buffer caches is to minimize the top timed events shown in the AWR report. After the wait events have been minimized, examine the utilization of the buffer cache. To improve the cache utilization, you can:

- Ensure that correctly tuned SQL statements are executed, thus minimizing the number of blocks that have to be accessed
- Examine the `V$DB_CACHE_ADVICE` view to determine whether the sizes of the buffer caches should be modified
- Examine the need for indexes to ensure that blocks are not needlessly read into the buffer cache. Adding indexes could reduce full table scans.
- Use multiple buffer pools to separate blocks by access characteristics
- Configure the tables to be cached in memory

Increasing the size of the data buffer cache does not always improve performance. The characteristics of the application may prevent further improvement of the cache hit ratio. For example, in large data warehouses or decision support systems, which routinely use many scans of large tables, most of the data is read from disk. For such systems, tuning the buffer cache is less important and tuning I/O is vital.

Technical Note

You need to consider the impact of operating system caching. For example, the Oracle server may show a high rate of physical I/O that does not appear at the operating system level. This could mean that database blocks, aged out of the buffer cache, are kept in the operating system cache and can be accessed very quickly. However, as a general rule it is best to bypass the operating system cache because of the following reasons:

- More memory may be required to maintain duplicate blocks in memory (one block in the operating system cache and one in the database buffer cache).
- There is the CPU overhead of copying blocks from the operating system cache to the database buffer cache.

OS caching can be bypassed by using Automatic Storage Management (ASM), and Direct I/O. If the file system is configured to bypass OS caching, much of that file system cache memory can be reassigned to the Oracle buffer cache (or PGA and SGA in general). The Oracle DBMS should be able to utilize that memory more efficiently for database activity than a general purpose file system cache and operating system.

Symptoms of a Buffer Cache Issue

The symptoms that indicate a buffer cache problem:

- Latch:cache buffer chains
- Latch:cache buffer LRU chains
- Buffer busy waits
- Read waits
- Free buffer waits
- Cache hit ratio



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

There are several symptoms that point to the buffer cache as a problem. If the following are observed in the Top 10 Foreground Events by Total Wait Time, then investigate further:

- **Latch:cache buffer chains:** This generally points to hot blocks in the cache.
- **Latch:cache buffer LRU chains:** Contention indicates that the buffer cache is too small, and block replacement is excessive.
- **Buffer busy waits:** This is almost always an application tuning issue. Multiple processes are requesting the same block.
- **Read waits:** There are a variety of read waits and multiple causes.
- **Free buffer waits:** When the DBW n processes do not make sufficient free buffers to meet the demand, free buffer waits are seen.

A low cache hit ratio, below 80%, is another symptom of a small buffer cache. More investigation is needed. A low cache hit ratio seldom occurs by itself, and should be used as a confirming symptom.

Cache Buffer Chains Latch Contention

Contention for this latch indicates:

- Multiple processes attempting to access the same “hot” block
- Excessive block replacement



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Because each process that accesses the block must obtain the latch to search the chain, multiple processes accessing the same block will produce contention. This block is called a “hot” block. Hot blocks are often caused by application design choices or inefficient SQL that rereads blocks. This usually calls for application tuning. If the hot block is in an index, a reverse key index or a global hash partitioned index may be a solution.

Contention for the cache buffer chains latch may indicate that the caller (server process) is attempting to modify the cache buffer chain. This happens when a buffer in the chain must be replaced with another block. Contention can occur when many more blocks are required by the application than there is space in the buffer cache. Check the size of the buffer cache.

Finding Hot Segments

- Characteristics of cache buffer chains latch contention:
 - Many accesses to one or more block under the same latch
 - Worse with larger block sizes
- To find hot segments: Check the “Segments by Logical Read” section of the AWR report or Statspack level 7 report.

AWR Report

Segments by Logical Reads						
<ul style="list-style-type: none"> • Total Logical Reads: 1,103,916 • Captured Segments account for 77.6% of Total 						
Owner	Tablespace Name	Object Name	Subobject Name	Obj. Type	Logical Reads	%Total
SPC	TBSSPC	SPCT		TABLE	749,152	67.86
SYS	SYSTEM	SEGS\$		TABLE	56,960	5.16
SYS	SYSTEM	LOB\$		TABLE	21,712	1.97
SYS	SYSTEM	OBJ\$		TABLE	10,496	0.95
SYS	SYSTEM	SYS_C00819		INDEX	1,840	0.17

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

If a particular block is looked up or updated by a large number of processes concurrently, you may see latch contention on the cache buffer chains latches. In Oracle Database, waits for this latch type appear in V\$SESSION_EVENT and V\$SYSTEM_EVENT as latch: cache buffer chains. This contention can be caused by one or more extremely hot buffers, such as application metadata blocks, undo header blocks, or index root or branch blocks.

There is very little you can do at the block level to alleviate this contention. You must identify the segment to which these blocks belong. The AWR report shows the most accessed segments, in various sorts for a particular time period. The Statspack report shows similar information if you use level 7 snapshots. The V\$SEGMENT_STATISTICS view shows segment statistics since the instance was started if the STATISTICS_LEVEL parameter is set to TYPICAL or ALL. This query shows the top 10 statistic values, by object and statistic name.

```
SELECT * FROM ( SELECT owner, object_name,
          object_type, statistic_name, sum(value)
     FROM v$segment_statistics
    GROUP BY owner, object_name, object_type, statistic_name
   ORDER BY SUM(value) DESC)
 WHERE ROWNUM <= 10;
```

ADDM reports this type of contention automatically. Most of the time, by tuning your application SQL, you should be able to remove this type of contention.

Buffer Busy Waits

- Application-level contention for buffers in the buffer cache
- Identify buffer busy wait contention:

AWR Report

Top 10 Foreground Events by Total Wait Time					
Event	Waits	Total Wait Time (sec)	Wait Avg(ms)	% DB time	Wait Class
buffer busy waits	40,150	253.4	6	63.5	Concurrency
library cache lock	14	58.4	4168	14.6	Concurrency
DB CPU		22.6		5.7	
eng: HW - contention	683	13.3	20	3.3	Configuration
free buffer waits	16				
db file sequential read	2,448				
local write wait	128				
log file sync	27				
log buffer space	47				
log file switch (private strand flush incomplete)	1				

Buffer Wait Statistics

- ordered by wait time desc, waits desc

Class	Waits	Total Wait Time (s)	Avg Time (ms)
data block	57,945	254	4
segment header	1,400	0	0
undo header	75	0	0
1st level bmb	1	0	0

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In the Oracle Database, buffer busy waits occur when one session needs to use a buffer in an incompatible mode with the current user of the buffer. For example, one session needs to access a buffer that a second session is modifying. This is, therefore, a case of application-level contention for buffers. You can identify this type of contention by looking at the AWR or Statspack report. Start in the Top 10 Timed Foreground Events, then using the Buffer Wait Statistics section, you can determine the class of block that is being impacted (DATA, UNDO, ...).

The Statspack and AWR reports obtain these statistics for a period by taking snapshots of V\$WAITSTAT.

The next step is to determine the segments with the highest number of buffer busy waits in the Segments By Buffer Busy Waits section of the AWR report. These statistics come from snapshots of the V\$SEGMENT_STATISTICS view with rows corresponding to STATISTIC_NAME='buffer busy waits'.

Index maintenance may add to buffer busy waits in transactional systems. Indexes on a table have to be maintained for inserts and deletes with a few exceptions.

Note: In Oracle Database, when one session needs a buffer that is presently being read into the buffer cache by a second session, the first session is waiting on an event called read by other session. This is a read-sharing case, and not a contention case.

Buffer Cache Hit Ratio

AWR Report

Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	99.98 R
Buffer Hit %:	71.13 D
Library Hit %:	99.91 S
Execute to Parse %:	99.75 L
Parse CPU to Parse Elapsd %:	35.65 %

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Buffer Hit % or buffer cache hit ratio is the percentage of how often a requested block has been found in the buffer cache without requiring disk access. This ratio is computed using snapshots taken from the v\$SYSSTAT dynamic performance view. The buffer cache hit ratio can be used to verify the physical I/O as predicted by V\$DB_CACHE_ADVICE. The following are the statistics used to calculate the hit ratio when flash cache is not being used:

- consistent gets from cache: Number of times a Consistent Read is requested for a block from the buffer cache
- db block gets from cache: Number of times a CURRENT block is requested from the buffer cache
- physical reads cache: Total number of data blocks read from disk into buffer cache

Note: Direct reads, direct gets, and direct temp reads bypass the buffer cache and are not included in the calculation.

Buffer Cache Hit Ratio Is Not Everything

- A badly tuned database can still have a hit ratio of 99% or better.
- Hit ratio is only one part in determining tuning performance.
- Hit ratio does not determine whether a database is optimally tuned.
- Tune SQL statements.
- Use the AWR/Statspack report to examine what is causing a bottleneck:
 - Top 10 Foreground Timed Events
 - Load Profile (logical and physical reads)
 - Instance Efficiency Percentage



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A common pitfall is to base cache resizing decisions on hit ratios. Many users mistakenly believe the hit ratio to be an absolute indicator of performance. In fact, the only thing that matters is how much time is being spent on performing disk I/Os due to cache misses.

An application could have a good hit ratio but have many more physical reads than necessary. A logical read is less expensive than a physical read, but there is still an associated cost to retrieving the block. Besides the access cost, there is the cost of memory space for extra blocks that are kept in memory but not used to resolve the query.

For example, suppose two applications (A and B) return the same result set. Application A has a cache hit ratio of 99%, whereas Application B has a hit ratio of 60 %. Which application has the best performance? On further investigation, you notice that Application A requires 100,000 logical reads and 1,000 physical reads, whereas Application B has 100 logical reads and 40 physical reads. Which application is better tuned now? With the additional information, the answer is B. There is clearly something wrong with Application A, when it does so many logical reads to return a small result set.

Therefore, having a good cache hit ratio is only a part of the tuning sequence. Examine the performance reports to determine which areas require your attention. The greatest benefit generally comes by tuning the SQL statements to perform fewer reads.

Interpreting Buffer Cache Hit Ratio

- Hit ratio is affected by data access methods:
 - Full table scans
 - Repeated scans of the same tables
 - Large table with random access
 - Data or application design
- Investigate increasing the cache size if:
 - Hit ratio is low
 - Application is tuned to avoid full table scans

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To interpret the buffer cache hit ratio, you should consider the following effects of data access methods:

- Database blocks accessed during a long full table scan (FTS) are not always loaded into the buffer cache. If a table is a large table and there is not enough room in the buffer cache for it, the full table scan will bypass the cache and become a direct path read into the PGA of the requesting server process. This conversion will increase the cache hit ratio and the number of physical reads.
- Repeated scanning of the same table or index can artificially inflate the cache hit ratio. Examine frequently executed SQL statements with a large number of buffer gets to ensure that the execution plan for such SQL statements is optimal.
- In any large database running OLTP applications in any given unit of time, most rows of a large table are accessed either one or zero times. On this basis, there is little purpose in keeping the block in memory following its use.
- If possible, avoid requerying the same data by caching frequently accessed data in the client program or middle tier.

A common mistake is to continue increasing the buffer cache size. Such increases have no effect if the application is causing full table scans or operations that do not use the buffer cache. Consider increasing the buffer cache when:

- The application is already tuned to avoid full table scans
- The hit ratio is low and the previous access methods have been corrected

Note: Short table scans are scans performed on tables under a certain size threshold. A short table is either less than 20 blocks or less than two percent of the buffer cache.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

Read Waits

- List of wait events performing disk reads into the buffer cache:
 - db file sequential read
 - db file parallel read
 - db file scattered read
- If the wait time for reads is high:
 - Tune the SQL statement that issues most disk reads; check SQL Ordered by Reads in the SQL Statistics section of the AWR report
 - Increase the buffer cache if needed
 - Reduce writes due to checkpointing
 - Add more disk capacity



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

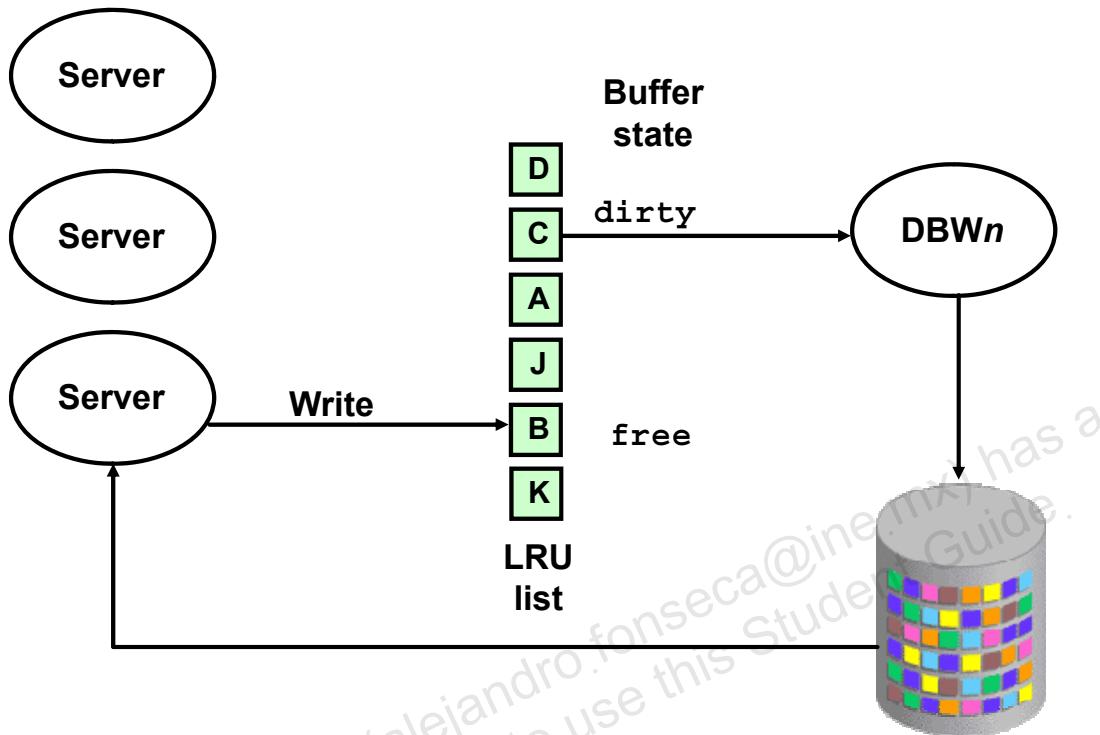
Wait events such as db file sequential read (wait for a single block to be read synchronously from disk), db file parallel read (wait for a synchronous multiblock read), and db file scattered read (wait for multiple blocks to be read concurrently from disk) constitute the set of wait events for performing disk reads into the buffer cache. These are normal waits and are always present even in well-tuned databases. Excessive waits on read wait events can be determined only by comparing the waits to a baseline when the database had acceptable performance. If the total time for these events appears to be an excessive fraction of the total database time, there are two possible investigative paths:

- When the average time for each read is normal (< 10 ms or the average read time for your disk subsystem):
 - Your workload is simply issuing too many disk reads and it is necessary to try to reduce this number. Find the SQL statements that are issuing the most logical and physical reads and tune them.
 - Additional disk reads may be caused by an undersized buffer cache. Use the Buffer Cache Advisor.

- When the average time for each read is abnormally high (> 15 ms or the average read time for your disk subsystem):
 - Consider reducing the number of writes due to an aggressive checkpointing policy. Reads are delayed when your system performs writes because the disks are busy servicing writes.
 - Add disk bandwidth by striping the existing disks to spread the I/O load, or add disk devices to reduce disk contention.

Note: ADDM can notify you when average read times are excessive. And you can adjust the expected I/O time by setting the DBIO_EXPECTED value as shown in the lesson titled “Using AWR-Based Tools.”

Free Buffer Waits



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

For every block moved into the buffer cache, a server process must scan the LRU list to find a free buffer that can be used to hold the data block. Ideally, the server process starts from the “cold” end of the LRU list and finds that the first buffer examined is a free buffer. If the first buffer is not free, the next buffer is examined, and so forth, until a free buffer is found or four buffers have been examined. If a free buffer is not found, then the server process requests that DBW n write more blocks out to disk, resulting in more free buffers, and the server process goes into a free buffer wait. A large number of free buffers waits indicates that the DBW n processes are not creating free buffers fast enough. You can resolve this problem by:

- Reducing the number of buffers required (tuning the application SQL)
- Increasing the total number of buffers (allowing DBW n more time to write)
- Increasing the rate that blocks are written to disk

Suppose you have already tuned the application, or you are not allowed to tune the application. Making the buffer cache larger does not always help. The `DB_CACHE_ADVICE` view shows that adding memory to the cache will give only a slight benefit. The next step is to speed up the rate at which DBW n writes blocks to disk.

Solutions for Buffer Cache Issues

The buffer cache solutions are applied depending on the symptoms:

- Properly size the buffer cache.
- Cache objects.
- Use the keep and recycle pools.
- Increase the writing speed of DBW n .
- Use the private I/O pool.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Sizing the Buffer Cache

Buffer cache size affects several tuning diagnostics. If the cache is too small, this can cause the following:

- Extra reads due to block replacement
- Extra writes to move dirty blocks to disk
- Buffer cache LRU chains contention



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When the buffer cache is too small, several tuning diagnostics can be affected. There will be more physical reads than would be required for a larger cache. There may be additional writes to move the dirty blocks to disk to make room for the other blocks to come into the cache. The buffer cache LRU chains latches will probably show contention because they are being frequently searched for free blocks.

The buffer cache size is controlled by a set of parameters. You can use Automatic Shared Memory Management (ASMM) to automatically size the SGA and the Buffer Cache Advisor to assist in choosing an efficient cache size for manual tuning.

Automatic Shared Memory Management (ASMM) is a subset of the Automatic Memory Management (AMM) feature.

Buffer Cache Size Parameters

- Set the primary block size for the recycle, keep, and default buffer pools: DB_BLOCK_SIZE
- Set the size of the buffer pools:
 - DB_CACHE_SIZE
 - DB_KEEP_CACHE_SIZE
 - DB_RECYCLE_CACHE_SIZE
- Represent all memory for the buffer cache
- Are required to use buffer cache features:
 - Dynamic grow/shrink
 - Buffer cache advice
 - Multiple block sizes



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The DB_BLOCK_SIZE parameter defines the default database block size. It is set for the database during database creation. This is the block size of the SYSTEM tablespace. Other permanent tablespaces may be of different block sizes. The undo and temporary tablespaces must use the default block size.

Three parameters set the size of three buffer pools defined with the default block size. DB_CACHE_SIZE sets the size of the default buffer pool. The DB_KEEP_CACHE_SIZE and DB_RECYCLE_CACHE_SIZE parameters set the size of the keep and recycle buffer pools, respectively. Only the default pool is required to be configured.

These values of the CACHE parameters are specified in units of memory (KB or MB), not in number of blocks. These values represent all of the memory used by the corresponding pools, including the memory used for the buffer cache metadata such as buffer headers. The *_CACHE_SIZE parameters must be used to enable dynamic memory sizing, buffer cache advice, or the use of nondefault block sizes.

Quiz

The buffer cache is used to hold database blocks in memory while the applications read and update the blocks. Identify the characteristics of a properly sized buffer cache.

- a. The buffer cache holds all the database blocks.
- b. Disk I/O is reduced.
- c. The DBWR process will write changed blocks to disk more frequently.
- d. The requested block is found in the buffer cache most of the time.
- e. The server processes will seldom find a free buffer.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: b, d

When the database buffer cache is properly sized, the server processes will request blocks for read or write and the block will already be in the cache. If the block is not in the cache, a disk I/O is required. The server process will find a free buffer, and then the server process performs a disk I/O to retrieve the database block into the buffer cache.

If the cache is undersized, DBWR will have to write changed buffers to disk more frequently to provide free buffers to bring other blocks into the cache, and server processes may have to wait for DBWR to create the free buffers.

Dynamic Buffer Cache Advisory Parameter

- The buffer cache advisory feature enables and disables statistics gathering for predicting behavior with different cache sizes.
- Use the information provided by these statistics to size the buffer cache optimally for a given workload.
- The buffer cache advisory is enabled by using the DB_CACHE_ADVICE initialization parameter:
 - This parameter is dynamic and can be changed using ALTER SYSTEM.
 - Three values are allowed: OFF, ON, and READY.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The recommended tool for sizing the buffer cache is called the Buffer Cache Advisor. Set the DB_CACHE_ADVICE parameter to enable the Buffer Cache Advisor as follows:

- OFF: The advisor is turned off and the memory for the advisor is not allocated.
- READY: The advisor is turned off, but the memory for the advisor remains allocated. Allocating the memory before the advisor is actually turned on avoids the risk of an ORA-4031 error (inability to allocate from the shared pool). If the parameter is switched to this state from OFF, an ORA-4031 error may be generated.
- ON: The advisor is turned on and the advisor publishes its results through the V\$DB_CACHE_ADVICE view. Attempting to set the parameter to this state when it is in the OFF state may lead to an ORA-4031 error. If the parameter is in the READY state, it can be set to ON without error because the memory is already allocated. The advisor mechanism incurs less than 0.1% overhead in terms of CPU and memory utilization and shows very accurate predictions. Before querying V\$DB_CACHE_ADVICE, make sure that you are running a representative workload.

Note: If STATISTICS_LEVEL is set to TYPICAL or ALL, DB_CACHE_ADVICE is automatically set to ON. It is set to OFF if STATISTICS_LEVEL is set to BASIC.

Buffer Cache Advisory View

- Buffer cache advisory information is collected in the V\$DB_CACHE_ADVICE view.
- The view contains different rows that estimate the number of physical reads for cache sizes between 10% and 200% of the current cache size.
- The rows also compute a physical read factor, which is the ratio of the number of estimated reads to the number of actual reads.
- Simulation is done for all buffer pools.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Support Buffer Cache Advisory View

In the V\$DB_CACHE_ADVICE view, each cache size simulated has its own row, with the predicted physical I/O activity that would take place for that size. The columns identify the buffer pool, the block size of the pool, and the size of the pool for the estimate. The estimates are:

- ESTD_PHYSICAL_READ_FACTOR: Physical read factor for this cache size, which is a ratio of the number of estimated physical reads to the number of reads in the real cache. If there are no physical reads in the real cache, the value of this column is null.
- ESTD_PHYSICAL_READS: Estimated number of physical reads for this cache size
- ESTD_PHYSICAL_READ_TIME: Estimated disk read time
- ESTD_PCT_OF_DB_TIME_FOR_READS: Estimated disk time as a percentage of the total time

Note: If multiple buffer pools or multiple block size caches are in use, the advisor contains similar entries for all nonstandard buffer caches.

Using the V\$DB_CACHE_ADVICE View

```
SELECT size_for_estimate, buffers_for_estimate,
       estd_physical_read_factor, estd_physical_reads
  FROM V$DB_CACHE_ADVICE
 WHERE name = 'DEFAULT' AND
       advice_status = 'ON' AND
       block_size = (SELECT value FROM V$PARAMETER
                      WHERE name = 'db_block_size');
```

Cache Size (MB)	Estd Phys		Estd Phys	
	Buffers	Read Factor	Reads	
(10%)	30	3,802	18.70	192,317,943
...				
	243	30,416	1.33	13,720,149
	273	34,218	1.13	11,583,180
(Current)	304	38,020	1.00	10,282,475
	334	41,822	.93	9,515,878
...				
	577	72,238	.67	6,895,122
(200%)	608	76,040	.66	6,739,731



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

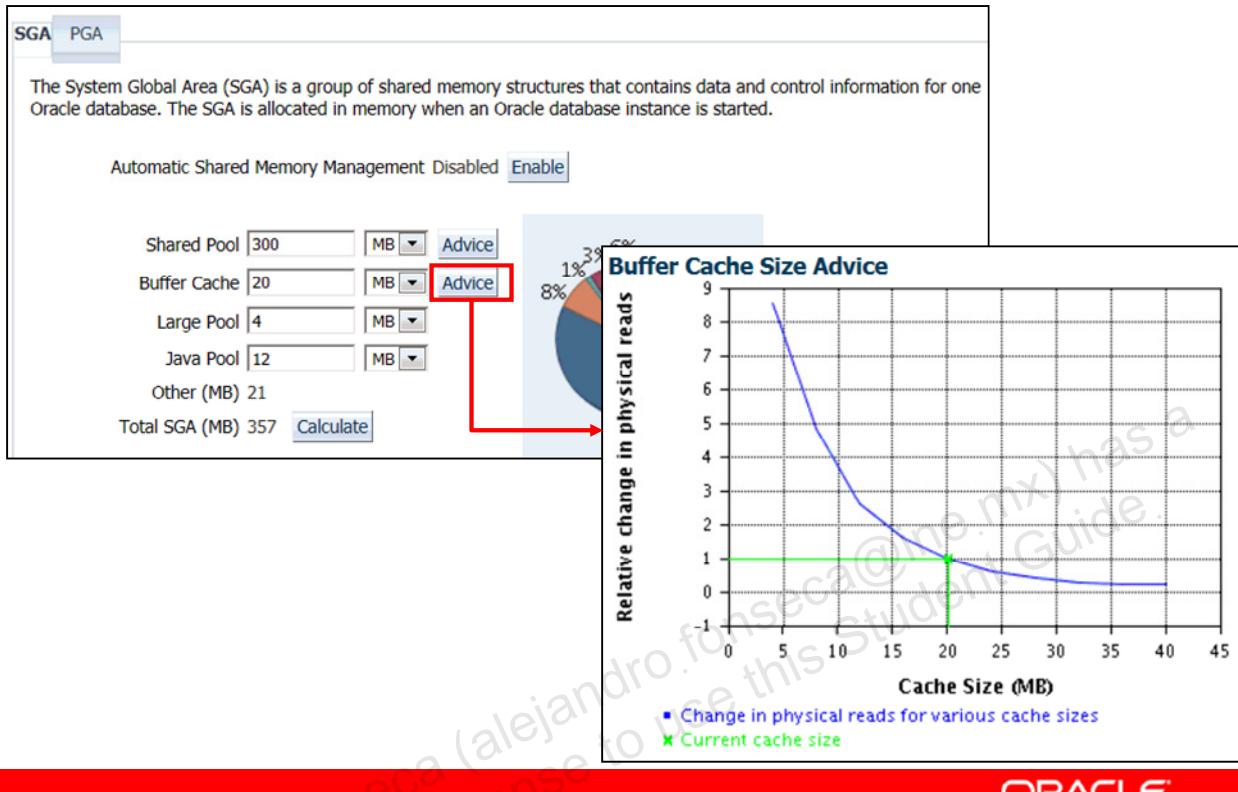
The output in the slide shows that if the cache was 243 MB, rather than the current size of 304 MB, the estimated number of physical reads would increase by a factor of 1.33 or 33%. This means it would not be advisable to decrease the cache size to 243 MB.

However, increasing the cache size to 334 MB would potentially decrease reads by a factor of .93 or 7%. If an additional 30 MB memory is available on the host machine and the SGA_MAX_SIZE setting allows the increment, it would be advisable to increase the default buffer cache pool size to 334 MB.

Choosing a cache size from the V\$DB_CACHE_ADVICE view is the same, regardless of whether the cache is the default standard block size cache, the KEEP or RECYCLE cache, or a nonstandard block size cache. Each cache that is allocated memory will be included in the V\$DB_CACHE_ADVICE view.

Note: With Oracle Database, physical reads do not necessarily indicate disk reads because physical reads may be satisfied from the file system cache.

Using the Buffer Cache Advisor



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can access the Buffer Cache Advisor in Enterprise Manager Cloud Control by expanding the Performance menu, selecting Memory Advisors, and clicking Advice next to the Buffer Cache field. From the Buffer Cache Size Advice graph, it is easy to see that increasing the cache size to 32-40 MB reduces the physical reads, but beyond that point more memory yields a much smaller benefit. The graph also shows that reducing the cache to about 12 MB will increase the number of physical reads somewhat, but a cache smaller than 12 MB will cause a large increase in physical reads.

Click the graph to set the DB_CACHE_SIZE parameter to a new value. The new value is applied immediately when the Apply button is clicked if the total size of the SGA is less than the SGA_MAX_SIZE parameter.

Note: The slide reflects the size of the classroom environment. A production environment would likely have orders of magnitude more buffer cache in use.

Caching Tables

- Enable caching during full table scans by:
 - Creating the table with the CACHE clause
 - Altering the table with the CACHE clause
 - Using the CACHE hint in a query
- Caching tables put blocks at the MRU end of the LRU list.
- Guideline: Do not overcrowd the buffer cache.
- Use a keep pool.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When the server retrieves blocks using a full table scan, the buffers go into the buffer cache only if there is room. Blocks read into the database buffer cache as the result of a full scan of a large table are treated differently from other types of reads. The blocks are immediately available for reuse to prevent the scan from effectively cleaning out the buffer cache.

To change this behavior, you must perform one of the following tasks:

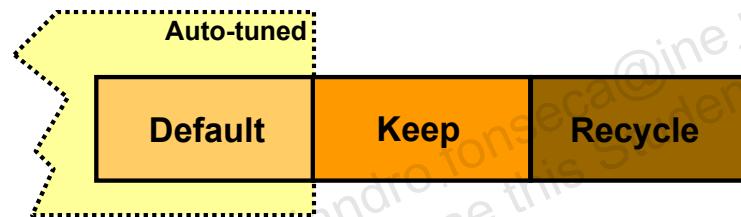
- Create a table by using the CACHE clause.
- Alter a table by using the CACHE clause.
- Code the CACHE hint clause into a query.

In this case, the database does not force or pin the blocks in the buffer cache, but ages them out of the cache in the same way as any other block. Use the CACHE clause when you create small lookup tables used by many users. You may overcrowd the buffer cache if you have too many cached tables. Cached tables should be less than 10% of the buffer cache.

Tables can also be effectively cached by using a keep pool.

Multiple Buffer Pools

- Three buffer pools:
 - Default: SYS and nonflagged table or indexes
 - Keep: Hot objects
 - Recycle: Infrequent access
- Useful for small, simple schemas with well-known access paths



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

With the multiple buffer pools feature, you can configure separately managed buffer caches. The benefit of multiple buffer pools is to reduce physical reads without increasing the total SGA. This feature can be very useful for small and simple schemas with well-defined access patterns. Three buffer pools are supported:

- **Default:** This pool always exists and its size is controlled by the DB_CACHE_SIZE parameter or by Automatic Shared Memory Management. All blocks read by the SYS user use this pool. In addition, if the buffer pool attribute of an object is not set, or is set and the specified pool does not exist, then the object uses this pool. Most objects are cached in this pool.
- **Keep:** This pool is intended for small or frequently accessed objects that would benefit from being memory-resident. It is sized to the sum of all objects that you want to keep plus a small percentage for Consistent Read blocks. Its size is controlled by the DB_KEEP_CACHE_SIZE parameter. The LRU list behavior is the same for each cache. If the keep pool is smaller than the number of blocks to be kept, some blocks will be forced to age out of the cache.

- **Recycle:** This pool is used for objects that gain no benefits from caching. These are usually large tables with random access, where the blocks are seldom used beyond a single transaction—for example, an event log table where rows are continuously added but is seldom accessed otherwise. Only enough buffers to hold the rows involved in an active transaction are needed, usually a small percentage of the blocks in the object. The recycle pool size is controlled by the `DB_RECYCLE_CACHE_SIZE` parameter.

Note: There is still only one common hash table used for all buffer pools for the purpose of buffer lookups.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

Enabling Multiple Buffer Pools

- Use the BUFFER_POOL clause.
- This clause is valid for tables, clusters, and indexes.
- When altered, buffer pool is used for future reads.
- Objects can use more than one buffer pool.

```
CREATE INDEX cust_idx ...
  STORAGE (BUFFER_POOL KEEP ...);

ALTER TABLE customer
  STORAGE (BUFFER_POOL RECYCLE);

ALTER INDEX cust_name_idx
  STORAGE (BUFFER_POOL KEEP);
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The BUFFER_POOL clause is used to define the default buffer pool for an object. It is part of the STORAGE clause and is valid for CREATE and ALTER TABLE, CLUSTER, and INDEX statements. The blocks from an object without an explicitly set buffer pool go into the default buffer pool.

The syntax is BUFFER_POOL {KEEP | RECYCLE | DEFAULT}.

When the default buffer pool of an object is changed using the ALTER statement, blocks that are already cached remain in their current buffers until they are flushed out by the normal cache management activity. Blocks read from disk are placed into the newly specified buffer pool for the segment. If the specified pool is not defined, the blocks of the object are placed in the default pool.

Because buffer pools are assigned to a segment, objects with multiple segments can have blocks in multiple buffer pools. For example, an index-organized table can have different pools defined on both the index and the overflow segment.

Note: If a buffer pool is defined for a partitioned table or index, then each partition of the object inherits the buffer pool from the table or index definition, unless you override it with a specific buffer pool.

Calculating the Hit Ratio for Multiple Pools

```
SQL> SELECT name, 1 - (physical_reads /  
2   (db_block_gets + consistent_gets)) "HIT_RATIO"  
3   FROM v$buffer_pool_statistics  
4  WHERE db_block_gets + consistent_gets > 0;  
  
NAME          HIT_RATIO  
-----  
KEEP          .983520845  
RECYCLE        .503866235  
DEFAULT        .790350047
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The V\$BUFFER_POOL_STATISTICS view displays statistics (physical writes, consistent gets, free buffer waits) against the multiple buffer caches (if allocated). When examining the different hit ratios of these pools, you should notice that:

- The keep pool has the highest hit ratio. The hit ratio should be high because you have sized the keep pool larger than the total size of the segments assigned to the pool and, therefore, no buffers (or very few) should need to be aged out. You should confirm that segments assigned to the keep pool are being utilized frequently and warrant the memory space used. You can query the V\$BH view joined to the DBA_OBJECTS view to confirm which objects have blocks in the SGA, and then query V\$SEGMENT_STATISTICS for the segments that are assigned to the keep pool to verify that the number of logical reads is much higher than the number of physical reads for those segments.

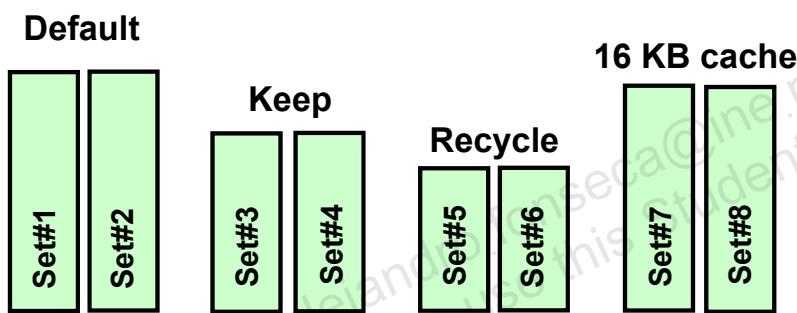
- The recycle pool has the lowest hit ratio. The hit ratio should be low because you have sized the recycle pool much smaller than the total size of the segments assigned to the pool; therefore, buffers have to be aged out due to blocks being read into the pool. You should confirm that segments assigned to the recycle pool are being kept in the pool long enough for the statement to complete processing. Use the Buffer Cache Advisor if the recycle pool is not keeping blocks long enough for the statements to finish. If the blocks are aging out too rapidly, the DB Buffer Cache Advisor will show that the number of physical reads is reduced with a larger recycle pool.

The values of the hit ratios are relative. The purpose of multiple buffer pools is not to increase the hit ratios, but to increase the performance by reducing the total number of physical I/Os.

Note: The data in V\$SYSSTAT reflects aggregation of the logical and physical reads for all buffer pools in one set of statistics.

Multiple Block Sizes

- Allow buffer caches for nonstandard block sizes
- Parameters: DB_nK_CACHE_SIZE { $n = 2, 4, 8, 16, 32$ }
- BLOCKSIZE attribute of CREATE TABLESPACE storage clause
- Intended for transportable tablespaces



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The standard block size of the database is specified by the DB_BLOCK_SIZE parameter. This is the block size of the SYSTEM tablespace and the default block size for any additional tablespaces. It is also possible to create nonstandard block size tablespaces. The supported set of block sizes are 2 KB, 4 KB, 8 KB, 16 KB, and 32 KB. To accommodate buffers from tablespaces with nonstandard block sizes, additional buffer caches need to be created for those block sizes. The set of five parameters DB_nK_CACHE_SIZE is used to specify the size of these buffer caches.

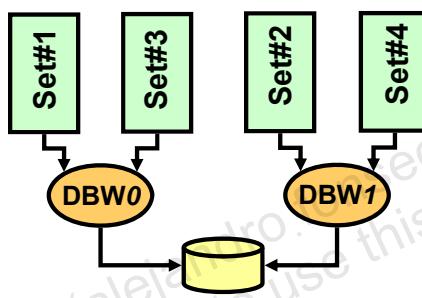
Similar to multiple buffer pools, caches for multiple block sizes are represented internally as ranges of working sets. However, multiple buffer pools are not supported for nonstandard block sizes. Again, buffers of nonstandard block sizes use the common hash table along with the buffers of the standard block size.

The primary intention of the multiple block size feature is to be able to support transportable tablespaces across databases with different block sizes. It is not intended as a performance feature.

Note: The V\$BUFFER_POOL view shows the detailed configuration of the buffer cache and its subdivision into buffer pools and multiple block size caches. Statistics corresponding to each cache can be found in V\$BUFFER_POOL_STATISTICS.

Multiple Database Writers

- Multiple database writers are a means to increase write throughput useful for large SMP systems.
- Buffer cache is partitioned between database writers by working sets.
- Each DBW n process scans its own assigned sets.
- The number of database writers can be manually controlled by DB_WRITER_PROCESSES.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

For concurrency and throughput, dirty buffers in the cache can be written to disk by more than one database writer process (DBW n).

The purpose of multiple database writers is to parallelize the CPU cycles of writing buffers to disk between multiple processes when the total CPU for disk writes exceeds the capacity of one single CPU. This is especially true for large (> 8 CPUs) SMP and NUMA systems for which a single database writer may not be enough to keep up with the rate of buffer dirtying by processes running on other processors. For this reason, the Oracle Database server automatically configures multiple database writers: one for every eight CPUs on the system.

These processes wake up and scan their assigned LRU and checkpoint queues for dirty buffers for aging and checkpoint operations. They gather a batch of buffers to write and issue the writes asynchronously using the most efficient available OS mechanism. As shown in the slide, working sets are the units of partitioning between database writers.

The number of database writers can be controlled by the DB_WRITER_PROCESSES initialization parameter. A maximum of 20 database writers is supported.

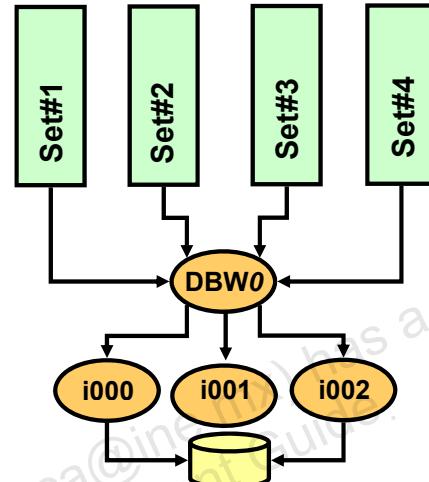
Multiple I/O Slaves

Multiple I/O slaves:

- Allow DBW0 to write in parallel when asynchronous I/O is not well supported
- Cannot be combined with multiple DBW n processes
- Are controlled by:
 - DBWR_IO_SLAVES
 - DISK_ASYNCH_IO

With I/O slaves, DBW0:

1. Gathers a batch of buffers
2. Queues the buffers in round-robin order to I/O slaves
3. Waits for all the slaves to complete



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Another mechanism for obtaining increased write throughput is to use I/O slaves (i_n). I/O slaves parallelize the writes from a single batch issued by DBW0. Thus, if DBW0 builds a batch of 1,000 buffers to write, with 10 I/O slaves, each of them will write 100 buffers.

I/O slaves are designed to provide additional write throughput when the system does not support asynchronous I/O, or when the initialization parameter `DISK_ASYNCH_IO` has been set to `FALSE`. In such a situation, it would be prohibitively slow for even multiple database writer processes to issue synchronous writes one by one, and I/O slaves should be used.

The parameter controlling the number of I/O slaves is `DBWR_IO_SLAVES`. The maximum supported number of I/O slaves is 999.

Note: Combining multiple database writer processes and I/O slaves is not supported.

Using Multiple Writers and I/O Slaves

To reduce free buffer waits:

- Implement asynchronous I/O first.
- Use multiple writers when a database writer is consuming 100% of a CPU.
- Use I/O slaves when asynchronous I/O is not supported.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When you observe free buffer waits, first eliminate checkpointing and I/O bandwidth as causes.

- Implement asynchronous I/O, if your system supports it.
- If one or more DBW n processes is consuming 100% of a CPU, increase the number of database writer processes. Start with 1 DBW n process per 8 CPUs and increase slowly, monitoring the changes. If there are not multiple CPUs, investigate further.
- If asynchronous I/O is not supported on your system, implement I/O slaves.

Writing buffers is not directly in the performance critical path because user processes rarely have to wait for a write of a buffer to complete. Your database writer configuration is adequate if you do not have free buffer waits. If there are no free buffer waits, increasing the number of database writers or I/O slaves does not increase the number of writes.

Symptoms of I/O bandwidth saturation are the following:

- The average time waited for the “db file sequential read” wait event is more than 10–15 ms.
- Large values for free buffer waits, but none of the database writers are consuming anywhere close to a full CPU.

Private Pool for I/O-Intensive Operations

- Used for operations that:
 - Do not require caching
 - Require efficient I/O
- Used for large asynchronous I/O
- Requires global checkpoint to ensure consistent data
- Used only for direct path I/O:
 - Parallel direct inserts
 - SQL*Loader direct path
 - Parallel full table scans
 - Serial full table scans



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Oracle Database provides a mechanism for bypassing the buffer pool for I/O-intensive operations, such as bulk loads. This is done for performance reasons and primarily to be able to issue large I/Os without flooding the buffer cache. This mechanism is referred to as direct path I/O.

Consistency must be maintained between the private and the shared buffer pools. By bypassing the buffer cache, the system must make sure that the blocks read from disk by the direct path I/O operation are current. Before launching the direct path I/O operation, the system must issue a global checkpoint across all instances to checkpoint all changes to the object up until to the time stamp needed by the direct path I/O operation.

However, not all operations can use the direct path I/O interface. For example, updates must go through the buffer cache because they have to update the current version of the blocks. One exception to this rule is for bulk insert operations that insert data above high-water marks of an object. The best candidates for using this mechanism are full table scans.

Note: This mechanism is used by the database internally. There is no user configuration needed.

Automatically Tuned Multiblock Reads

- DB_FILE_MULTIBLOCK_READ_COUNT is automatically tuned. The automatic setting:
 - Is enabled if the parameter is not set or is set to zero
 - Simplifies the determination of the optimal I/O size
- Optimal I/O size is:
 - Platform dependent
 - The size of the prefetch
- Rules for the automatic parameter behavior are:
 - Prefetch is limited to 64 KB, if prefetch blocks exceed 10% of the cache.
 - Value of eight blocks is used for calculating the cost of full table scans.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The DB_FILE_MULTIBLOCK_READ_COUNT parameter controls the number of blocks prefetched into the buffer cache during scan operations, such as full table scan and index fast full scan.

Because this parameter has a significant impact on the overall performance, Oracle Database automatically selects an appropriate value for this parameter depending on the operating system's optimal I/O size and the size of the buffer cache.

The optimal I/O size is the size of the prefetch, as long as the total number of prefetched blocks due to full scans does not exceed 10% of the cache. This limit ensures that prefetches do not swamp the buffer cache and age-out more useful data. If the number of prefetched blocks exceeds 10% of the cache, then the prefetch size is limited to 64 KB. When the DB_FILE_MULTIBLOCK_READ_COUNT parameter is not set or is explicitly set to 0, this is the default behavior. If you explicitly set a value, that value is used and is consistent with the behavior of previous releases.

Note: Because this automatic setting can influence the optimizer to use more full table scan plans when it is a large value, the optimizer will not favor plans with full table scans if you do not explicitly set this parameter.

Database Smart Flash Cache Overview

- Database Smart Flash Cache:
 - Is an extension of the buffer cache that resides on a flash disk
 - Has the following advantages:
 - Large capacity and cheaper compared to DRAM
 - Faster throughput and lower latency compared to disks
 - Performance improvements at moderate cost
- A flash disk must have write IOPs comparable to read IOPs to be used as a flash cache.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Database Smart Flash Cache feature is conceptually an extension of the buffer cache that resides on a flash disk. The concept is similar to the hierarchical caching system in modern CPU design, where most frequently accessed data resides in Level 1 cache (here in memory) and less frequently accessed data resides in a larger Level 2 cache (here on flash disk). This allows you to have a buffer cache larger than the memory on the server.

A flash disk is a solid state device (SSD) made of flash memory, like those in the modern MP3 devices. A regular disk has spinning magnetic media. With a flash disk, random I/O is very fast because disk seek time is eliminated. Flash devices appear in different forms, from the low-end USB sticks to the high-end PCIe cards.

Flash disks take advantage of the large capacity of flash devices compared to DRAM, and the faster throughput and latency compared to disks. In addition, flash disks are cheaper than memory.

This can result in enormous performance improvements at moderate cost. Many more data blocks can be cached in the system using memory and flash disk, providing higher IOPs and lower read latency.

However, there are many varieties of flash disks and some have very low write performance compared to read performance. To be used as a flash cache, the flash disk must have write IOPs comparable to read IOPs.

Using Database Smart Flash Cache

- Benefits:
 - Provides better performance for the same price
 - Is easy to set up
 - Provides an interface to fine-tune object-level granularity control of the LRU mechanism
- Mainly for read-intensive OLTP workloads



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

This feature helps you to get better performance. In a database system when the buffer cache memory is at capacity and the I/O starts thrashing, your only option is to buy more memory. But the GB/\$ ratio on main memory is not cheap and the system might not have enough DRAM slots. So you may be forced to upgrade the machine.

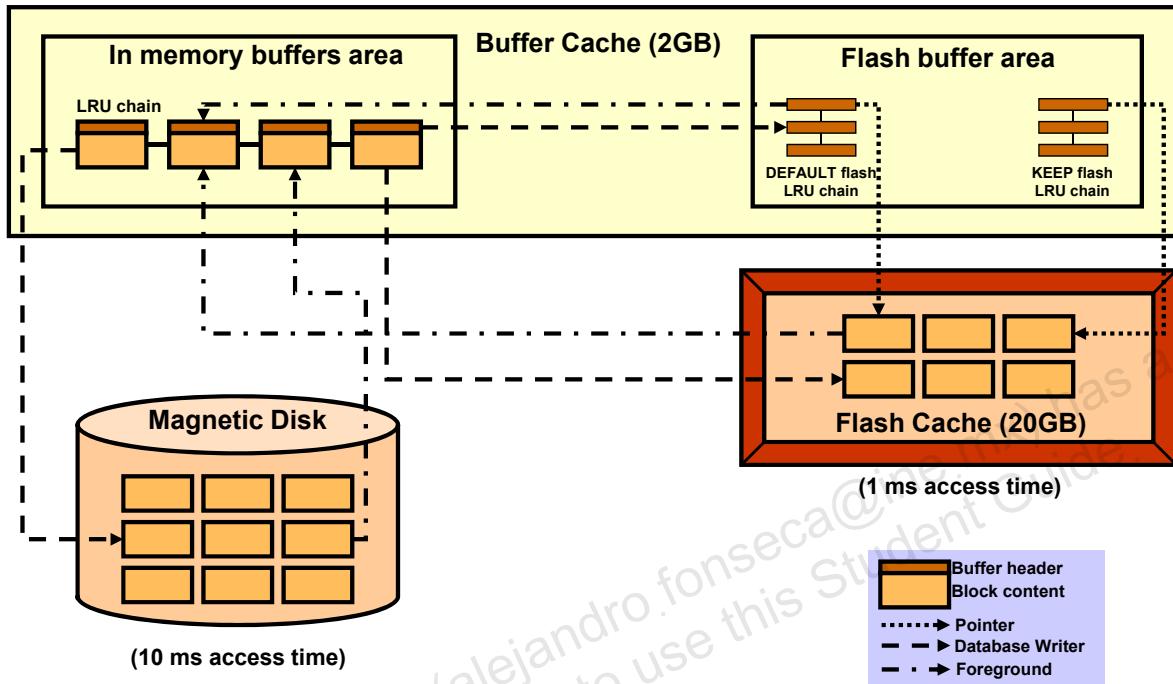
Flash disks are the perfect middle ground in this situation. For the same price as main memory, you can acquire 5X-10X flash disk in size. Using the flash cache feature, the flash disk can be configured with the main memory buffer cache to provide a much larger combined buffer cache. If the whole working set of the database can fit into the flash cache, after the flash cache has been warmed up, there is very few cache I/Os to the magnetic disks besides for checkpoint writes and direct I/Os. The response time and overall throughput for read-intensive OLTP workloads are expected to improve in a system with the same amount of main memory but additional SSD flash devices configured.

The feature is easy to use because only two initialization parameters are specified before startup.

This feature also provides you with an interface to fine-tune the object-level granularity control of the LRU in the flash cache. If needed, you can also specify object-level control over the flash cache LRU mechanism by specifying two storage clause keywords.

Note: A physical read from the magnetic disk typically takes about 10 ms, while flash cache has a typical access time of 1 ms.

Database Smart Flash Cache Architecture Overview



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Without Database Smart Flash Cache, when a process tries to access a block and the block does not exist in the buffer cache, the block will first be read from disk into memory (physical read). When the in-memory buffer cache gets full, a buffer will get evicted out of the memory based on an LRU mechanism.

With Database Smart Flash Cache, when a clean in-memory buffer is aged out, the buffer content is written to the flash cache in the background by the Database Writer (DBWn), and the buffer header is kept in memory as metadata in either the flash DEFAULT or KEEP LRU list, depending on the value of the FLASH_CACHE object attribute. The flash KEEP LRU list is used to maintain the buffer headers on a separate list to prevent the regular buffer headers from replacing them. Thus, the flash buffer headers belonging to an object specified as KEEP tend to stay in the flash cache longer. If the FLASH_CACHE object attribute is set to NONE, the system does not retain the corresponding buffers in the flash cache or in memory.

When a buffer that was already aged out of memory is accessed again, the system checks the flash cache. If the buffer is found, it reads it back from the flash cache, which takes only a fraction of the time of reading from the disk. The consistency of flash cache buffers across RAC clusters is maintained in the same way as by Cache Fusion. Because the flash cache is an extended cache and direct path I/O totally bypasses the buffer cache, this feature does not support direct path I/O.

Note that the system does not put dirty buffers in flash cache because it may have to read buffers into memory in order to checkpoint them because writing to flash cache does not count for checkpoint.

Configuring Database Smart Flash Cache

- Two initialization parameters:
 - `DB_FLASH_CACHE_FILE` specifies the OS file/disk path and name for the flash cache.
 - `DB_FLASH_CACHE_SIZE` represents the size of the flash cache.
- Nonstandard block size is not supported with Database Smart Flash Cache.
- All standard block size buffers use the same flash cache.
- Database Smart Flash Cache is not auto-tuned.
- Dynamically changing the size of the flash cache is not supported.

```
db_flash_cache_file = '/dev/foal'
db_flash_cache_file = '/work/flash_cache.dbf'
db_flash_cache_file = '+FLASH_DG/flash_cache'
db_flash_cache_size = 5G
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You need to set two initialization parameters to use the Database Smart Flash Cache feature:

- `DB_FLASH_CACHE_FILE` specifies the OS file/disk path and name for the flash cache. The file specified in the parameter must be a path to a flash disk. If the file does not exist, the database will create a new file with this name during startup, with a size of `DB_FLASH_CACHE_SIZE`. The first example illustrates using a raw device, whereas the second shows using a file system mounted on the device. The third example could be used with ASM. The disk group is created first (`FLASH_DG` in this case) where all physical disks in the disk group are flash drives/cards, and where `FLASH_CACHE` is an alias to an actual ASM file, named automatically. If the file exists, the database server will reuse the same file and validate the size. In this release, the old content on the same file will be discarded.
- The flash cache file/disk must be private to a single instance and cannot be shared between instances or databases. This parameter has to be specified before startup and cannot be modified after the instance has been started up.
- `DB_FLASH_CACHE_SIZE` specifies the size of the flash cache. This parameter must be specified before startup and `DB_FLASH_CACHE_FILE` must be specified. After the database is started, it can be modified to a value of 0 (which disables the flash cache). When flash cache is disabled, it can be reenabled by setting `DB_FLASH_CACHE_SIZE` back to the original value. Dynamic resizing of `DB_FLASH_CACHE_SIZE` or reenabling to a different size is not supported.

Nonstandard block sizes are not supported in flash cache. All standard block size buffers from different buffer pools will use the same flash cache.

Sizing Database Smart Flash Cache

- Size flash cache to be between 2-10 times the size of the buffer cache.
- With Automatic Shared Memory Management, make flash cache between 2-10 times the size of SGA_TARGET.
- Add 100-200 bytes to buffer cache for each block moved to flash cache.
- You can use:
 - ALTER SYSTEM to set DB_FLASH_CACHE_SIZE to zero to disable flash cache
 - ALTER SYSTEM to set flash cache back to its original size to reenable it
- Database Smart Flash Cache is not auto-tuned.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

As a general rule, size flash cache to be between two times and ten times the size of the buffer cache. Any multiplier less than two will not provide any benefit.

If you are using Automatic Shared Memory Management, make flash cache between two times and ten times the size of SGA_TARGET. Using 80 percent of the size of SGA_TARGET instead of the full size would also suffice for this calculation.

For each database block moved from the buffer cache to flash cache, a small amount of metadata about the block is kept in the buffer cache. For a single instance database, the metadata consumes approximately 100 bytes. For a RAC database, it is closer to 200 bytes.

You must take this extra memory requirement into account when adding flash cache:

- If you are managing memory manually, increase the size of the buffer cache by approximately an amount equal to the number of database blocks that fit into flash cache multiplied by 100 (or 200 for Oracle RAC).
- If you are using Automatic Memory Management, increase the size of MEMORY_TARGET using the algorithm described in the previous bullet. You may first have to increase the size of MEMORY_MAX_TARGET.
- If you are using Automatic Shared Memory Management, increase the size of SGA_TARGET.

Note: The tuning between DB_CACHE_SIZE and DB_FLASH_CACHE_SIZE cannot be done automatically because DB_FLASH_CACHE_SIZE is not auto-tuned in the MEMORY_TARGET feature.

Enabling and Disabling Flash Devices

- Set DB_FLASH_CACHE_FILE and DB_FLASH_CACHE_SIZE:

```
db_flash_cache_file = /dev/raw/sda, /dev/raw/sdb,  
                      /dev/raw/sdc  
db_flash_cache_size = 32G, 32G, 64G
```

- Disable /dev/raw/sdb:

```
db_flash_cache_size = 32G, 0, 64G
```

- Enable /dev/raw/sdb:

```
db_flash_cache_size = 32G, 32G, 64G
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You specify the flash devices and their sizes at instance startup in the SPFILE or text parameter file. Even though you cannot change the size of the flash cache dynamically, you can enable and disable individual devices.

You can specify up to 16 flash memory devices with DB_FLASH_CACHE_FILE and specify their sizes with DB_FLASH_CACHE_SIZE.

For example, if there are three flash raw devices, you can specify the devices and the size of each device, as shown in the slide.

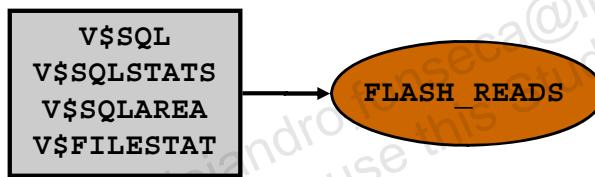
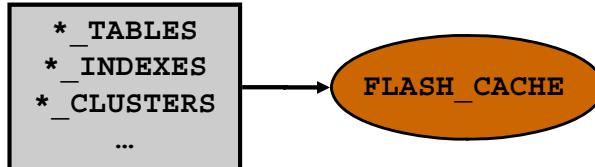
If your flash cache consists of one flash cache device, you can dynamically change this parameter to 0 for that flash cache device (disabling the flash cache) after the database is started. You can then reenable the flash cache by setting this parameter for the device back to the original value when the database was started. Dynamic resizing of DB_FLASH_CACHE_SIZE or reenabling flash cache to a different size is not supported.

If your flash cache includes multiple flash cache devices, you can dynamically change the parameter to 0 for a particular flash cache device (turning it off) after the database is started. You can then re-enable that flash cache device by setting this parameter for the device back to its original value when the database was started (turning it back on).

The example in the slide turns off the /dev/raw/sdb flash cache device and then turns the /dev/raw/sdb flash cache device back on again.

Specifying Database Smart Flash Cache for a Table

```
CREATE TABLE mycache
TABLESPACE tbs_1
STORAGE (flash_cache keep);
```



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The FLASH_CACHE option of the storage clause enables you to specify the following attributes:

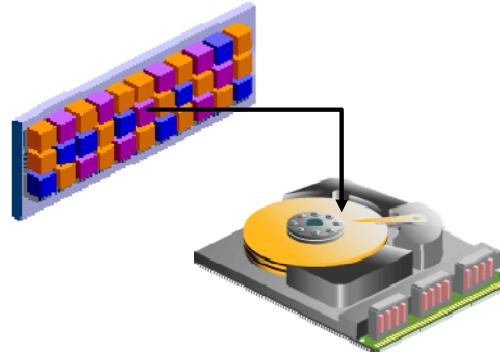
- DEFAULT: In-memory buffers for this object are put on the flash cache after they are aged out of the main memory and become flash buffers. Such flash buffers are kept on a flash LRU list and might be aged out with a standard LRU algorithm.
- KEEP: In-memory buffers for this object are put on the flash cache after they are aged out of the main memory. Flash buffers for this object are kept on a separate LRU list from the other flash buffers, and are not aged out of the flash cache as long as the flash cache size is large enough.
- NONE: In-memory buffers for this object are not put on the flash cache when they are aged out of memory.

The slide shows you a CREATE TABLE example using the FLASH_CACHE storage clause option.

In addition, the slide shows some of the dictionary views that have columns to reflect the flash information.

Note: For consistency, the system counts flash I/Os as physical I/O for all type of statistics. It also provides corresponding flash-specific statistics to differentiate flash I/Os from disk I/Os. AWR reports are updated to reflect these statistics and wait events. Flash-related statistics are also in V\$SESSTAT.

Flushing the Buffer Cache (for Testing Only)



```
ALTER SYSTEM FLUSH BUFFER_CACHE;
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The intent of flushing the database buffer cache is to provide an identical starting point for comparison of rewritten SQL statements. After you execute this command, the buffer is emptied and the next statement has a 100 percent cache miss. You should execute this command only on a test system; you should also execute it only when everyone using the system is aware of the ramifications.

The benefit of this feature is that it allows you to establish a consistent testing environment. You can flush the buffer cache manually between runs of test queries, thereby making it possible to determine the effects of changes in queries or applications.

Quiz

The Database Smart Flash Cache is used to hold database blocks that have been aged out of the buffer cache. These blocks when accessed again are returned to the buffer cache in a fraction of the time required for a disk I/O. The recommended size of the flash cache is between 2 and 10 times the size of the buffer cache.

- a. True
- b. False



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: a

Summary

In this lesson, you should have learned how to:

- Describe the buffer cache architecture
- Size the buffer cache
- Resolve common performance issues related to the buffer cache
- Use common diagnostic indicators to suggest a possible solution

Practice 17: Tuning the Buffer Cache

This practice covers:

- Using the DB Cache Advisor
- Using the Keep Pool

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

18

Tuning PGA and Temporary Space

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Diagnose PGA memory issues
- Size the PGA memory
- Diagnose temporary space issues
- Specify temporary tablespace parameters for efficient operation

SQL Memory Usage

- Memory-intensive SQL operators:
 - Sort-based (sort, group-by, rollup, window, and so on)
 - Hash-join
 - Bitmap operators (merge and inversion)
- Concept of work area: Memory allocated by a memory-intensive operator to process its input data
- Performance impact of memory:
 - **Optimal:** Input data fits into the work area (cache).
 - **One-pass:** Perform one extra pass over input data.
 - **Multi-pass:** Perform several extra passes over input data.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

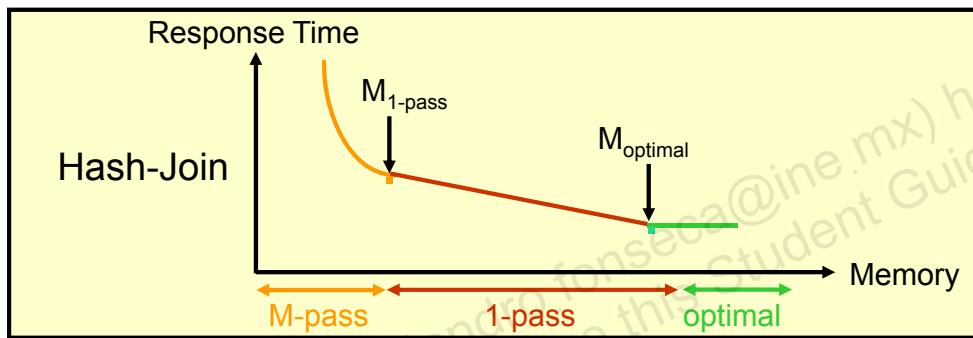
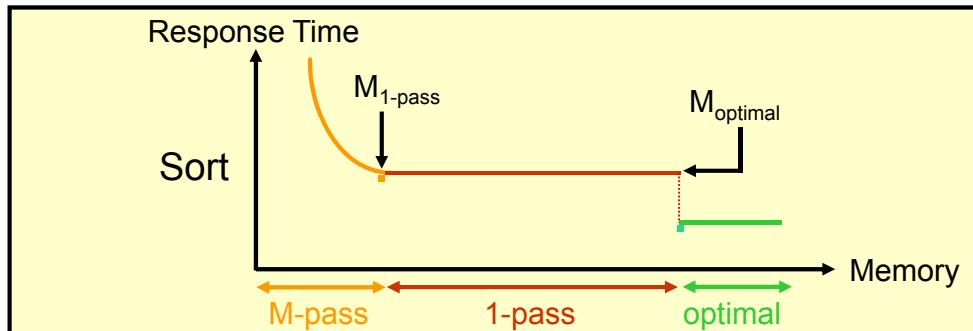
Complex database queries require memory-intensive operators such as sort and hash-join. Those operators need what is called “work area memory” to process their input data. For example, a sort operator uses a work area to perform the in-memory sort of a set of rows. Similarly a hash-join operator uses a work area to build a hash table on one of the tables in the FROM clause.

The amount of memory allocated by these operators greatly affects their performance, and a larger work area can significantly improve the performance of a SQL operator. The optimal size of a work area is big enough to accommodate the input data and auxiliary memory structures.

Because there is only a finite amount of memory in the system shared by all concurrent operators, an operator cannot always allocate its optimal size. When the size of the work area is smaller than its ideal cache size, the response time increases because an extra pass is performed on all or some of the input data. This is referred to as the one-pass size of the work area.

When the work area size is less than the one-pass threshold, multiple passes over the input data are needed, causing dramatic increase in response time. This is referred to as the multi-pass size of the work area. For example, a sort operation that needs to sort 10 GB of data needs a little more than 10 GB of memory to run in cache, and 40 MB to run in one-pass mode. It will run in multiple passes with less than 40 MB.

Performance Impact



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The diagram in the slide shows you the performance characteristics of the sort and hash-join operators with regard to memory usage.

The one-pass point on the curve is the start of the area where the operator runs in one-pass mode, and the optimal point corresponds to the case when the work area size is equal to the optimal size.

The sort curve is flat between those two points because a sort does not benefit from additional memory if it cannot use its optimal size. However, the hash-join benefits from additional memory between the one-pass and optimal points.

In online transaction processing (OLTP) systems, the size of the input data to SQL operators is generally small, thus they can run in optimal mode most of the time. This is not the case in decision support systems (DSS), where the input data is very large. Thus, it is important to size their work area to obtain good performance.

Automatic PGA Memory

- Dynamically adapts the SQL memory allocation based on:
 - PGA memory available
 - SQL operator needs
 - System workload
- Improves manageability:
 - No need to set * _AREA_SIZE parameters
 - DBA sets a memory target: PGA_AGGREGATE_TARGET
- Improves performance:
 - PGA memory is really returned to the OS.
 - Memory is allocated to the operation to maximize throughput.
 - Overall memory utilization is maximized by dynamically adapting memory with workload variation.
 - An operation adapts its memory usage during execution.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

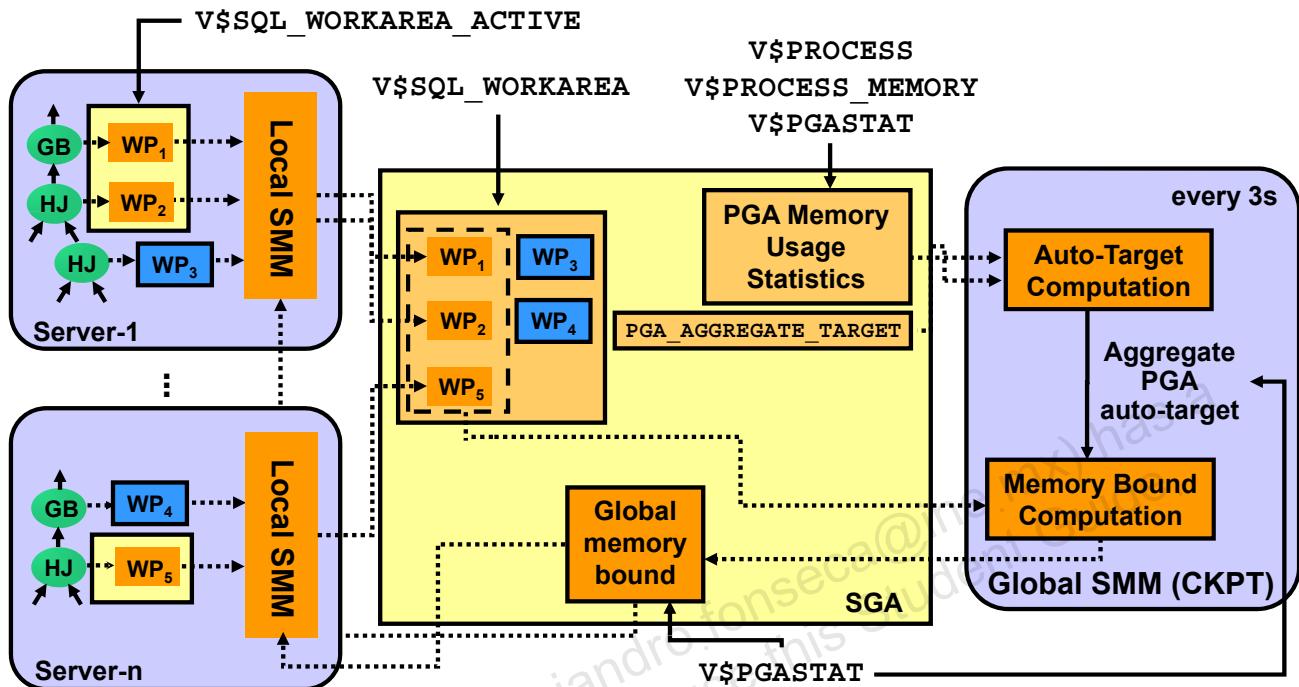
Besides SQL, various components in the database server make use of PGA memory. These other parts are known as the untunable parts of the PGA because they require an allocation of memory that cannot be adjusted. This is the case for:

- The context information of each session
- Each open cursor
- PL/SQL, OLAP, or Java memory

The tunable portion of the PGA represents the memory available to SQL work areas. This portion could represent 90% of the overall PGA memory for decision support systems, whereas it could be less than 10% in pure OLTP systems. With automatic PGA memory management, the system attempts to keep the amount of private memory below the target specified by the PGA_AGGREGATE_TARGET initialization parameter by adapting the size of the work areas to private memory. When increasing the value of this parameter, you indirectly increase the memory allotted to work areas. Consequently, more memory-intensive operations are able to run fully in memory and less will work their way over to disk.

Note: Oracle Corporation does not recommend the use of static SQL memory management. For more information about SORT_AREA_SIZE, HASH_AREA_SIZE, BITMAP_MERGE_AREA_SIZE, and CREATE_BITMAP_AREA_SIZE, refer to the *Oracle Database Reference* guide.

SQL Memory Manager



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

SQL PGA memory management (the tunable portion of the PGA) is mainly based on the feedback loop mechanism depicted in the slide.

The left side of the figure represents active SQL statements. When a SQL operator starts, it registers its work area profile (**WP₁**) using the local SQL memory manager (SMM). A work area profile is a piece of metadata that describes all the characteristics of a work area: its type (sort, hash-join, group-by); its current memory requirement to run with minimum, one-pass, and cache memory; its degree of parallelism; and finally the amount of PGA memory it currently uses. The set of active work area profiles is maintained by the local memory manager in the SGA. These profiles are constantly being updated with the current needs and usage.

The right side of the graphic in the slide represents the global SQL memory manager implemented by the CKPT background process, which runs every three seconds. It has two main components:

- The first component is responsible for computing the aggregate PGA auto-target, which represents the amount of PGA memory the system can use for the SQL work areas running in automatic mode. This amount is dynamically derived from the value of **PGA_AGGREGATE_TARGET** and also accounts for other PGA memory structures used by components such as PL/SQL, Java, and OLAP.

- The second component is responsible for computing a global memory bound. The process takes into account the profiles of active work areas and the aggregate PGA auto-target. This bound is used to constrain the size of each SQL work area; it sets a maximum size of each SQL work area. Therefore, the memory bound is high when the overall memory requirement of all active work areas is low and vice-versa. In simple terms, finding a proper value for the memory bound is finding the maximum value for which the sum of the expected work area size of each operator is less than or equal to the aggregate PGA auto-target.

The feedback loop is closed by the local memory manager. It uses the current value of the memory bound and the current profile of a work area to determine the correct amount of PGA memory, called expected size, which can be allotted to this work area. The expected size is checked periodically by SQL operators, which are then responsible to adapt their work area size to the specified value.

When the local memory manager sees that an operation has finished or reduces the size of its memory allocation, memory is released back to the OS on any operating system that supports this operation.

Configuring Automatic PGA Memory

- **PGA_AGGREGATE_TARGET:**
 - Specifies the target aggregate amount of PGA memory available to the instance
 - Can be dynamically modified at the instance level
 - Examples: 100,000 KB; 2,500 MB; 50 GB
 - Default value: 10 MB or 20% of the size of the SGA, whichever is greater
- **WORKAREA_SIZE_POLICY:**
 - Is optional
 - Can be dynamically modified at the instance or session level
 - Allows you to fall back to static SQL memory management for a particular session



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

PGA_AGGREGATE_TARGET specifies the target of the aggregate of all PGA memory available to all server processes attached to the instance.

The WORKAREA_SIZE_POLICY parameter is automatically set to AUTO when the PGA_AGGREGATE_TARGET parameter is set to a nonzero value. This means that SQL work areas used by memory-intensive SQL operators are automatically sized. The default value for this parameter is 20% of the SGA or 10 MB, whichever is greater. Setting PGA_AGGREGATE_TARGET to 0 automatically sets the WORKAREA_SIZE_POLICY parameter to MANUAL. This means that SQL work areas are sized by using the *_AREA_SIZE parameters. PGA_AGGREGATE_TARGET is not a strict limit. It is used to help the system better manage PGA memory, but the system will exceed this setting if necessary.

WORKAREA_SIZE_POLICY can be altered per database session, allowing manual memory management on a per-session basis, if needed. For example, a session is loading a large import file and a larger SORT_AREA_SIZE is needed. A logon trigger could be used to set the WORKAREA_SIZE_POLICY for the account doing the import.

Note: PGA_AGGREGATE_TARGET controls work areas allocated by both dedicated and shared connections.

Setting PGA_AGGREGATE_TARGET Initially

- Leave 20% of the available memory to other applications.
- Leave 80% of memory to the Oracle instance.
- For OLTP:

```
PGA_AGGREGATE_TARGET = (total_mem * 80%) * 20%
```

- For DSS:

```
PGA_AGGREGATE_TARGET = (total_mem * 80%) * 50%
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Assume that an Oracle instance is configured to run on a system with 4 GB of physical memory. Part of that memory should be left for the operating system and other non-Oracle applications running on the same hardware system. You might decide to dedicate only 80% (3.2 GB) of the available memory to the Oracle instance.

You must then divide the resulting memory between the SGA and the PGA.

For OLTP systems, the PGA memory typically accounts for a small fraction of the total memory available (for example, 20% of the instance memory), leaving 80% for the SGA.

For DSS systems running large, memory-intensive queries, PGA memory can typically use up to 70% of the instance memory (up to 2.2 GB in this example).

Good initial values for the `PGA_AGGREGATE_TARGET` parameter might be:

- **For OLTP:** `PGA_AGGREGATE_TARGET = (total_mem * 80%) * 20%`
- **For DSS:** `PGA_AGGREGATE_TARGET = (total_mem * 80%) * 50%`

where `total_mem` is the total amount of physical memory available on the system.

In this example, with a value of `total_mem` equal to 4 GB, you can initially set `PGA_AGGREGATE_TARGET` to 1600 MB for a DSS system and to 655 MB for an OLTP system.

Note: Setting the PGA memory target greater than available real memory can lead to excessive paging. The PGA advisor may recommend more than the amount of real memory available.

Limiting the Size of the Program Global Area

- PGA memory usage may exceed the value set by `PGA_AGGREGATE_TARGET` due to the following reasons:
 - `PGA_AGGREGATE_TARGET` acts as a target, not a limit.
 - `PGA_AGGREGATE_TARGET` controls only the allocations of tunable memory.
- Use the `PGA_AGGREGATE_LIMIT` initialization parameter to specify a hard limit on PGA memory usage.
- When `PGA_AGGREGATE_LIMIT` is reached:
 - The sessions that are using the most untunable memory will have their calls aborted
 - If the total PGA memory usage is still over the limit, the sessions that are using the most untunable memory will be terminated



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In automatic PGA memory management mode, Oracle Database attempts to adhere to the value specified in `PGA_AGGREGATE_TARGET` by dynamically controlling the amount of PGA memory allotted to work areas. However, at times PGA memory usage may exceed the value specified by `PGA_AGGREGATE_TARGET` for the following reasons:

- `PGA_AGGREGATE_TARGET` acts as a target, not a limit.
- `PGA_AGGREGATE_TARGET` only controls allocations of tunable memory.

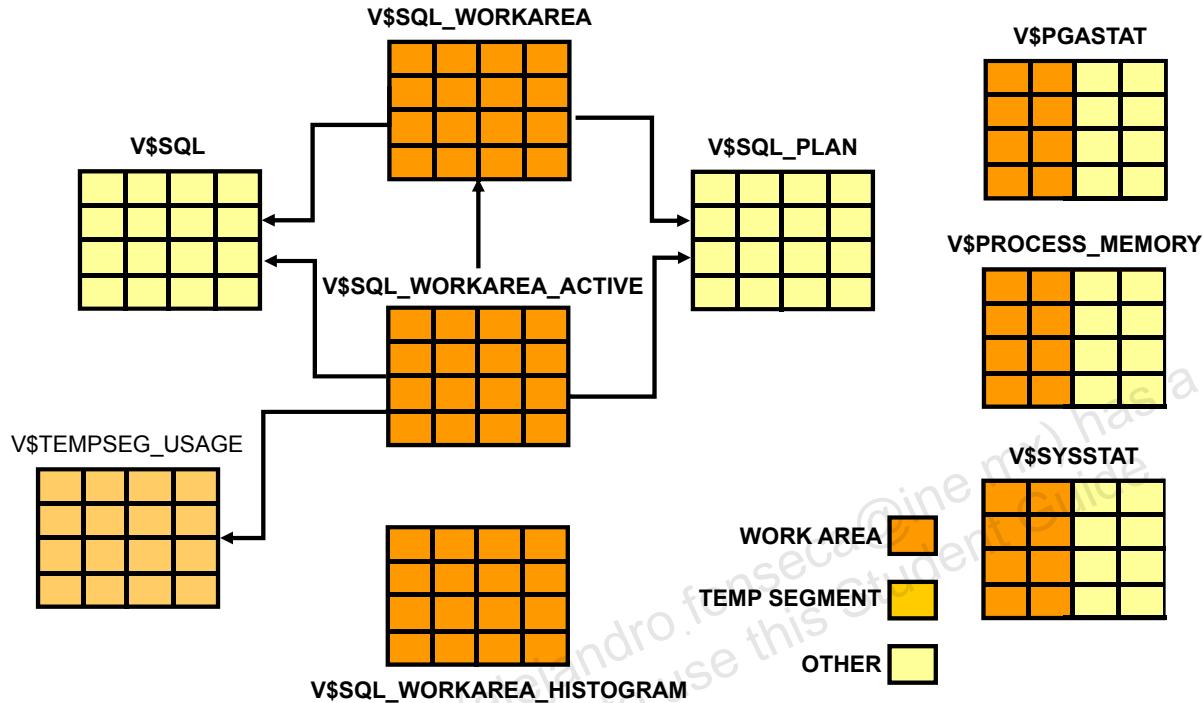
Excessive PGA usage can lead to high rates of swapping. If this occurs, consider using the `PGA_AGGREGATE_LIMIT` initialization parameter to limit overall PGA usage.

`PGA_AGGREGATE_LIMIT` enables you to specify a hard limit on PGA memory usage. If the `PGA_AGGREGATE_LIMIT` value is exceeded, Oracle Database aborts or terminates the sessions or processes that are consuming the most untunable PGA memory. Parallel queries are treated as a single unit.

By default, `PGA_AGGREGATE_LIMIT` is set to the greater of 2 GB, 200% of the `PGA_AGGREGATE_TARGET` value, or 3 MB times the value of the `PROCESSES` parameter. However, it will not exceed 120% of the physical memory size minus the total SGA size. The default value is written to the alert log. A warning message is written in the alert log if the amount of physical memory on the system cannot be determined.

SYS and background sessions are exempted from the limit, but not job queue processes.

Monitoring SQL Memory Usage



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

V\$SQL_WORKAREA maintains cumulative work area statistics for each loaded cursor whose execution plan uses one or more work areas. Every time a work area is deallocated, the **V\$SQL_WORKAREA** table is updated with execution statistics for that work area.

V\$SQL_WORKAREA can be joined with **V\$SQL** to relate a work area to a cursor. It can even be joined to **V\$SQL_PLAN** to precisely determine which operator in the plan uses a work area.

V\$SQL_WORKAREA_ACTIVE can be used to display the work areas that are active (or executing) in the instance. Small active sorts (under 64 KB) are excluded from the view. Use this view to precisely monitor the size of all active work areas and to determine whether these active work areas spill to a temporary segment. If a work area spills to disk, this view contains information for the temporary segment created on behalf of this work area, and can be joined with **V\$TEMPSEG_USAGE** to retrieve more information.

V\$SQL_WORKAREA_HISTOGRAM displays the cumulative statistics for different work area sizes. There are 33 groups of work areas based on their optimal memory requirements increasing in powers of two: work areas whose optimal requirement varies from 0 KB to 1 KB, 1 KB to 2 KB, 2 KB to 4 KB, ... and 2 TB to 4 TB. The **V\$SQL_WORKAREA_HISTOGRAM** view shows for each work area group how many work areas in that group were able to run in optimal mode, how many were able to run in one-pass mode, and finally how many ran in multi-pass mode.

V\$PROCESS_MEMORY displays dynamic PGA memory usage by category, for each process. The categories are: SQL, PL/SQL, OLAP, and Java. Special categories are Freeable and Other. The amount of Freeable memory represents the free PGA memory eligible to be released to the operating system.

V\$PGASTAT provides cumulative PGA memory usage statistics as well as current statistics about the automatic PGA memory manager when it is enabled.

Statistics in the V\$SYSSTAT and V\$SESSTAT views show the total number of work areas executed with optimal memory size, one-pass memory size, and multi-pass memory size. These statistics are cumulative since the instance or the session was started.

The information in the Other category includes cursor and operator information in the V\$SQL and V\$SQL_PLAN views. There is also session and system aggregate statistics in the V\$SESSTAT, V\$SYSSTAT, and V\$PGASTAT views.

Note: For more information about these performance views, refer to the *Oracle Database Reference* guide.

Monitoring SQL Memory Usage: Examples

```
SELECT sql_text,
       sum(onepass_executions) onepass_cnt,
       sum(multipasses_executions) mpass_cnt
  FROM v$sql s, v$sql_workarea wa
 WHERE s.address = wa.address
 GROUP BY sql_text
 HAVING sum(onepass_executions+multipasses_executions)>0;
```

1

```
SELECT TO_NUMBER(DECODE(sid, 65535, NULL, sid)) sid,
       operation_type OPERATION,
       TRUNC(expected_size/1024) ESIZE,
       TRUNC(actual_mem_used/1024) MEM,
       TRUNC(max_mem_used/1024) MAXMEM,
       number_passes PASS,
       TRUNC(tempseg_size/1024) TSIZE
  FROM v$sql_workarea_active
 ORDER BY 1,2;
```

2

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The first example shows you how to find the cursors with one or more work areas that have been executed in one or multiple passes.

The second example shows you how to find information about active work areas. The output of this query might look like the following:

SID	OPERATION	ESIZE	MEM	MAXMEM	PASS	TSIZE
8	GROUP BY (SORT)	315	280	904	0	
8	HASH-JOIN	2995	2377	2430	1	20000
9	GROUP BY (SORT)	34300	22688	22688	0	
11	HASH-JOIN	18044	54482	54482	0	
12	HASH-JOIN	18044	11406	21406	1	120000

This output shows you that session 12 is running a hash-join having its work area running in one-pass mode. This work area is currently using 11,406 KB of memory and has used, in the past, up to 21,406 KB of PGA memory. It has also spilled to a temporary segment of size 120,000 KB. Finally, the ESIZE column indicates the maximum amount of memory that the PGA memory manager expects this hash-join to use. This maximum is dynamically computed by the PGA memory manager according to workload.

Tuning SQL Memory Usage

- Determine the best `PGA_AGGREGATE_TARGET` value by using:
 - `V$PGA_TARGET_ADVICE`
 - `V$PGA_TARGET_ADVICE_HISTOGRAM`
- Monitor AWR reports/Statspack reports for:
 - direct path read temp
 - direct path write temp



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Oracle Database server helps you in the task of tuning the value of `PGA_AGGREGATE_TARGET` by providing two advice performance views: `V$PGA_TARGET_ADVICE` and `V$PGA_TARGET_ADVICE_HISTOGRAM`. By examining these two views, you do not need to use an empirical approach to the tuning process.

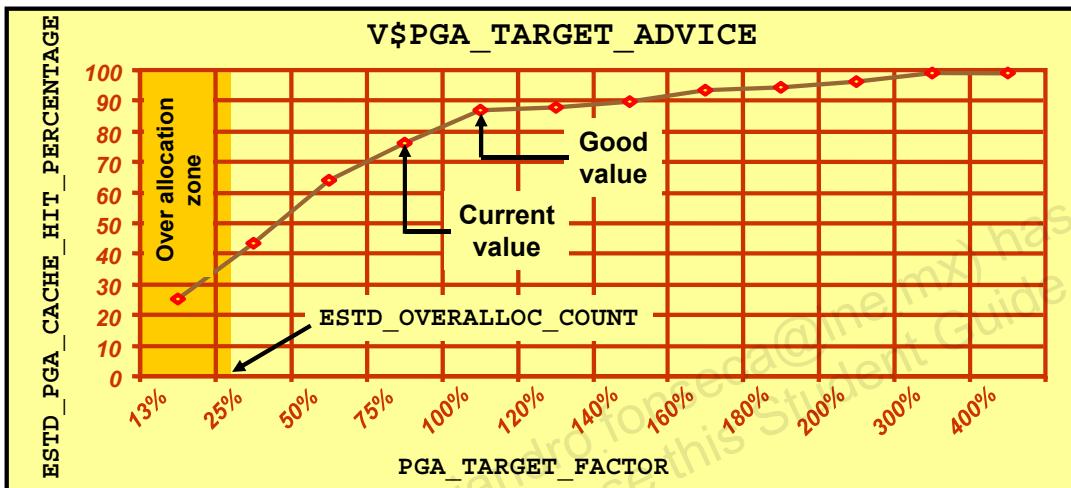
In addition, you can monitor the top wait event sections of your AWR or Statspack report looking for the following two events: `direct path read temp` and `direct path write temp`.

These two events are triggered when a session is reading buffers from or writing buffers to the temporary segments. These reads and writes are directly to or from the PGA. These occur because work areas are too large to fit in memory and are written to disk. These are the biggest waits for large data warehouse sites. However, if the workload is not a DSS workload, then examine why this is happening.

You can do so by looking at the SQL statement currently being run by the session experiencing waits to see what is causing the sorts. Query `V$TEMPSEG_USAGE` to find the SQL statement that is generating the sort. Also, query the statistics from `V$SESSTAT` for the session to determine the size of the sort. See whether it is possible to reduce the use of work areas by tuning the SQL statement. In addition, investigate whether to increase `PGA_AGGREGATE_TARGET`.

PGA Target Advice Statistics

- V\$PGA_TARGET_ADVICE predicts how cache hit percentages shown in V\$PGASTAT evolve.
- STATISTICS_LEVEL must be set to at least TYPICAL.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

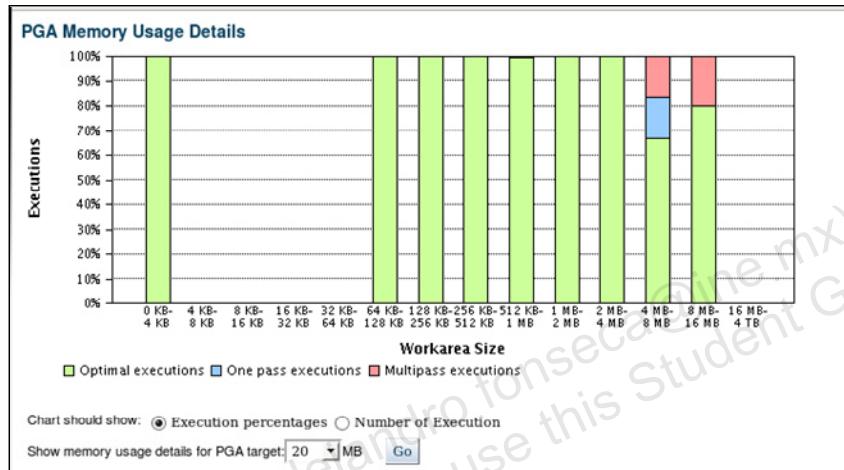
V\$PGA_TARGET_ADVICE predicts how the cache hit percentage displayed by the V\$PGASTAT performance view would be impacted if the value of the PGA_AGGREGATE_TARGET parameter is changed. The prediction is performed for various values of the PGA_AGGREGATE_TARGET parameter, selected around its current value. The advice statistic is generated by simulating the past workload run by the instance.

The graph in the slide shows how the PGA cache hit percentage metric improves as the value of PGA_AGGREGATE_TARGET increases. This cache hit percentage metric reflects the average percentage of SQL work areas that are able to run cache. The overallocation zone indicates that PGA_AGGREGATE_TARGET is insufficient to meet the minimum PGA memory needs, and forces the memory manager to allocate more PGA memory than the limit that you set. The ESTD_OVERALLOC_COUNT column predicts the number of overallocations. The good value for PGA_AGGREGATE_TARGET is reached when the cache hit percentage improves marginally compared to the extra memory need. Ideally, PGA_AGGREGATE_TARGET should be set to the maximum value possible in the region beyond the overallocation zone.

Note: The content of V\$PGA_TARGET_ADVICE is empty if PGA_AGGREGATE_TARGET is not set, or if STATISTICS_LEVEL is set to BASIC. Base statistics for this view are reset at instance startup and when the value of the PGA_AGGREGATE_TARGET initialization parameter is dynamically modified.

PGA Target Advice Histograms

- V\$PGA_TARGET_ADVICE_HISTOGRAM predicts how histograms shown in V\$SQL_WORKAREA_HISTOGRAM evolve.
- STATISTICS_LEVEL must be set to at least TYPICAL.



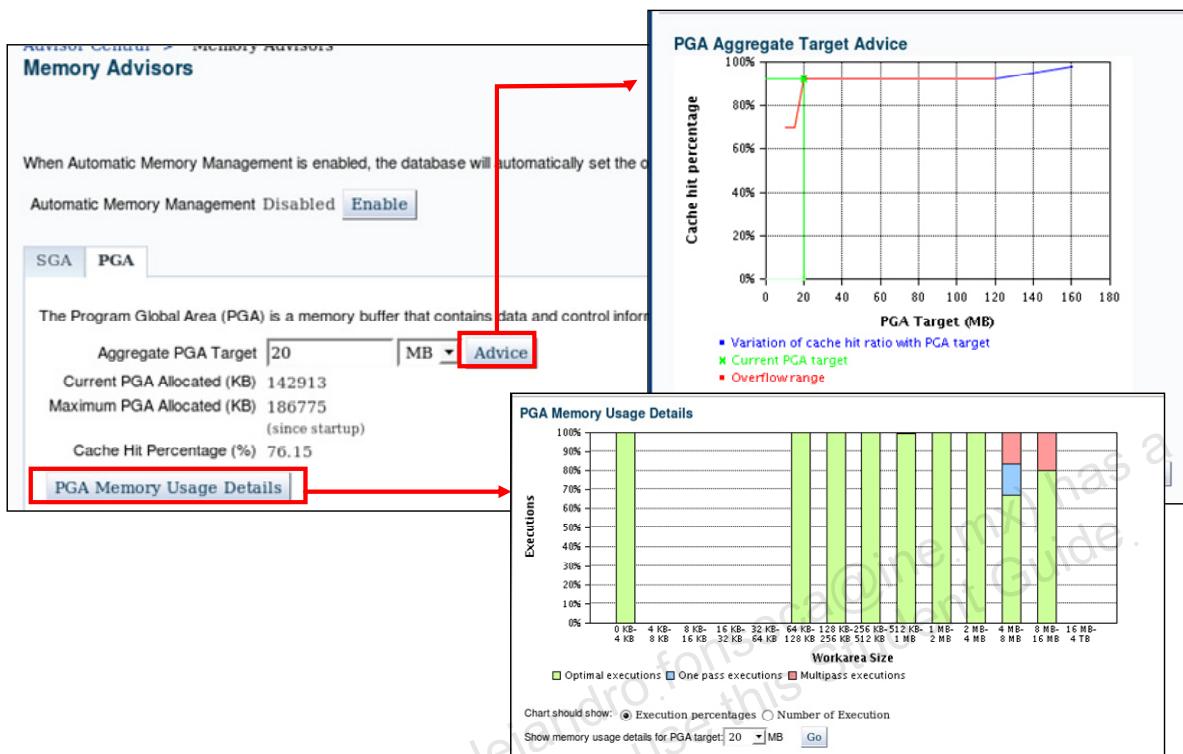
ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

V\$PGA_TARGET_ADVICE_HISTOGRAM predicts how statistics displayed by V\$SQL_WORKAREA_HISTOGRAM would be impacted if the value of the PGA_AGGREGATE_TARGET parameter is changed. This prediction is performed for various values of the PGA_AGGREGATE_TARGET parameter, selected around its current value. The advice statistic is generated by simulating the past workload run by the instance to determine the number of work areas executed with optimal memory size, one-pass memory size, and multi-pass memory size.

The slide shows you a graphical interpretation of the content of V\$PGA_TARGET_ADVICE_HISTOGRAM in Enterprise Manager Cloud Control.

Automatic PGA and Enterprise Manager



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Use the PGA tab on the Memory Advisors page to set the Aggregate PGA Target value. You can choose the unit of measure from the drop-down list in bytes, kilobytes, megabytes, or gigabytes. You can also see the values for the Current PGA Allocated, the Maximum PGA Allocated, and the Cache Hit Percentage.

You can also click Advice to display a chart that shows the cache hit percentage and the PGA target in megabytes. The chart provides advice on the value that you can set for the Aggregate PGA Target value.

You can click PGA Memory Usage Details to display a chart showing the various types of executions in the available work area size depending on different possible PGA targets.

Automatic PGA and AWR Reports

PGA Aggr Target Stats

- B: Begin Snap E: End Snap (rows denoted by B or E contain data which is absolute i.e. not relative)
- Auto PGA Target - actual workarea memory target
- W/A PGA Used - amount of memory used for all Workareas (manual + auto)
- %PGA W/A Mem - percentage of PGA memory allocated to workareas
- %Auto W/A Mem - percentage of workarea memory controlled by Auto Mem Mgmt
- %Man W/A Mem - percentage of workarea memory under manual control

PGA Aggr Target(M)	Auto PGA Target(M)	PGA Mem Alloc(M)	W/A PGA Used(M)	%PGA W/A Mem	%Auto W/A Mem	%Man W/A Mem	Global Mem Bound(K)
B	1,020	171	187.43	0.00	0.00	0.00	59,801
E	1,020	176	179.51	0.00	0.00	0.00	59,801

PGA Aggr Summary

- PGA cache hit % - percentage of W/A (WorkArea) data processed only in-memory

PGA Cache Hit %	W/A MB Processed	Extra W/A MB Read/Written
100.00	1,212	0

PGA Memory Advisory

- When using Auto Memory Mgmt, minimally choose a pga_aggregate_target value where Estd Extra W/A MB Read/ Written is 0

PGA Target Est (MB)	Size Factor	W/A MB Processed	Estd Extra W/A MB Read/ Written
37	0.13	25,098.60	4,21
73	0.25	25,098.60	4,21
146	0.50	25,098.60	1,21
219	0.75	25,098.60	0.00
292	1.00	25,098.60	0.00
350	1.20	25,098.60	0.00
409	1.40	25,098.60	0.00
467	1.60	25,098.60	0.00
526	1.80	25,098.60	0.00
584	2.00	25,098.60	0.00
876	3.00	25,098.60	0.00
1,168	4.00	25,098.60	0.00
1,752	6.00	25,098.60	0.00
2,336	8.00	25,098.60	0.00

PGA Aggr Target Histogram

- Optimal Executions are purely in-memory operations

Low Optimal	High Optimal	Total Execs	Optimal Execs	1-Pass Execs	M-Pass Execs
2K	4K	79,937	79,937	0	0
64K	128K	82	82	0	0
256K	512K	9	9	0	0
512K	1024K	621	621	0	0
1M	2M	378	378	0	0

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can find the main information (shown in the slide) relating to PGA statistics in the Advisory Statistics section of the AWR report.

In the PGA statistics, in the Process Memory Summary section of the AWR report, you can also find the following statistics: maximum PGA allocation size at snapshot time and the historical maximum allocation for still-connected processes.

In the Instance Activity Statistics section of the AWR report, you can also find the following total, per second, and per transaction statistics: session PGA memory and session PGA maximum memory.

Temporary Tablespace Management: Overview

- Temporary data generated by a database includes:
 - Bitmap merges
 - Hash-join
 - Bitmap index creation
 - Sort
 - Temporary LOBs
 - Global temporary tables
- Data persists for the duration of a transaction or session.
- High concurrency of the space management operation is critical.
- Media and instance recovery is not required.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A temporary tablespace is used to store transient data generated explicitly by the user and implicitly by the system. The data stored in a temporary tablespace is predominantly from hash-join, sort, bitmap merges, bitmap index creation operations, as well as temporary LOBs and global temporary tables.

In DSS and OLAP environments, the efficient management of transient data is central to the execution of queries; therefore, the performance of temporary tablespace is critical.

Temporary objects do not require recovery. Therefore, temporary tablespaces never need to be backed up, they can easily be re-created after a database recovery. For the same reason, the redo generated on temp files is automatically discarded after modifying the block. It is not written to the disk. This allows for faster writes to temp files.

Temporary objects must have undo for transactional consistency, and traditional undo must be recoverable with redo generated for each undo record. Setting `TEMP_UNDO_ENABLED = TRUE`, will create undo for these objects in temporary tablespaces, so no redo is generated. This reduces the amount of redo, LGWR I/O, and ARCH I/O in systems using temporary objects. The `TEMP_UNDO_ENABLED` initialization parameter can be set at the instance or session level. The `COMPATIBLE` parameter must be set to 12.1.0 or higher.

Temporary Tablespace: Locally Managed

Using locally managed temporary tablespaces:

- Allows high-concurrency space management
 - At steady state, all space metadata is cached in SGA.
 - Operations are serialized by the SGA latch.
- Allows faster writes to temp files. Redo generated on temporary blocks is not written to disk.
- Makes READ ONLY databases possible



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Tablespaces are classified as temporary and permanent. The space management in temporary tablespaces differs from that in permanent tablespaces in the following ways:

- The temporary tablespaces consist of temp files rather than data files that constitute the permanent tablespace.
- A temporary tablespace allows high-concurrency space management. In a steady state, all the space metadata is cached in the SGA. This allows for completely in-memory space management operations. The space management operations in a locally managed temporary tablespace are serialized by instance-specific SGA latches held for a very short duration.
- Read-only database: All the metadata required by the temporary tablespace is stored in the tablespace itself. No files outside the temporary files need to be modified, thereby making read-only databases possible.

Oracle Corporation recommends the use of locally managed temporary tablespaces (which use temp files), due to the inherent performance benefits.

Configuring Temporary Tablespace

- Locally managed temporary tablespaces are uniform-extent tablespaces.
- 1 MB to 10 MB extent size:
 - For DSS, OLAP applications involving huge work areas
 - Large temporary LOBs are predominant.
- 64 KB or multiple less than 1 MB:
 - Small global temporary tables are predominant.
 - This is suitable for OLTP.
- Use 1 MB extents until workload patterns are known.
- Use V\$TEMPSEG_USAGE to monitor space usage and workload distribution.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

While creating a temporary space for the database, the tablespace extent size is important to consider, because of its impact on performance. The extent size can vary anywhere from two data blocks to over 2 GB with 1 MB being the default. Choosing the correct extent size involves the usual space versus time trade-off. Very small extent sizes affect read/write performance, and increase the number of space management operations on large segments. Very large extent sizes can cause space to be wasted in the last extent allocated to the segment, with no improvement in performance. It is important to understand the types of workloads that use the temporary tablespace:

- **Temporary LOBs:** Temporary LOBs are created explicitly by the user as well as implicitly by the database to store transient unstructured data. If the temporary space usage is dominated by large temporary LOBs, larger extent sizes should be used. Oracle Corporation recommends using 1 MB to 10 MB for the extent size.
- **DSS:** The DSS workload is typically dominated by complex queries performing intensive sort and hash-join operations. Because the data in the work area is written to the temporary segments in multiples of 64 KB, it is advisable to choose an extent size that is a multiple of 64 KB. The general rule for good performance and efficient space usage is to set an extent size of 1 MB.

- **Global temporary tables:** These are transient tables created by the user as well as by the system in the temporary tablespace. Each global temporary table requires at least one extent to be allocated. If the volume of data loaded into the temporary tables is small, then choosing smaller multiples of 64 KB for the extent size should avoid space wastage. This is also suitable for OLTP.

- **General:** 1 MB extent size is a good initial choice until workload patterns are established

The distribution of workload can be monitored through the V\$TEMPSEG_USAGE dictionary view:

```
SQL> SELECT session_num, username, segtype, blocks, tablespace
```

```
2 FROM v$tempseg_usage;
```

```
SESSION_NUM USERNAME SEGTYPE BLOCKS TABLESPACE
```

SESSION_NUM	USERNAME	SEGTYPE	BLOCKS	TABLESPACE
101	SCOTT	SORT	128	TEMP
102	SCOTT	LOB_DATA	128	TEMP
103	SYS	SORT	256	TEMP
104	BLAKE	LOB_DATA	128	TEMP

The SEGTYPE column indicates what type of segment is using the space: SORT, HASH, INDEX, LOB_DATA, or DATA. On the basis of the distribution of segment sizes, an appropriate extent size can be chosen.

Temporary Tablespace Group: Overview

- Groups multiple temporary tablespaces
- Characteristics:
 - At least one temporary tablespace
 - Same namespace as tablespaces
 - Created implicitly on first assignment
 - No explicit deletion



ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A temporary tablespace group can be thought of as a shortcut for a list of temporary tablespaces. A temporary tablespace group consists of only temporary tablespaces.

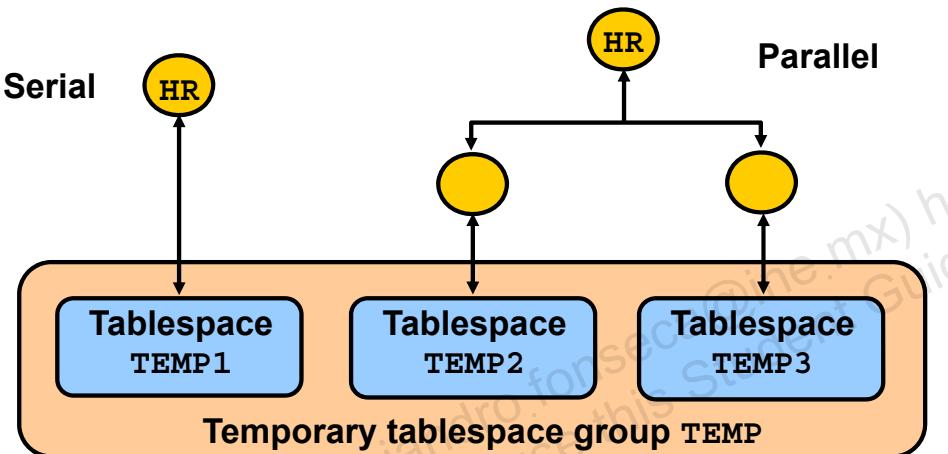
A temporary tablespace group has the following properties:

- It contains at least one temporary tablespace. There is no explicit limit on how many tablespaces are contained in a group.
- It has the same namespace as tablespaces. It is not possible for a tablespace and a temporary tablespace group to have the same name.
- A temporary tablespace group name can appear wherever a temporary tablespace name appears (for example, while assigning a default temporary tablespace for the database, or assigning a temporary tablespace for a user).
- It is not explicitly created. It is created implicitly when the first temporary tablespace is assigned to it, and it is deleted when the last temporary tablespace is removed from the group.

Temporary Tablespace Group: Benefits

Enables a user to use multiple temporary tablespaces:

- Same user in multiple sessions
- One particular parallel operation



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

This feature has the following benefits:

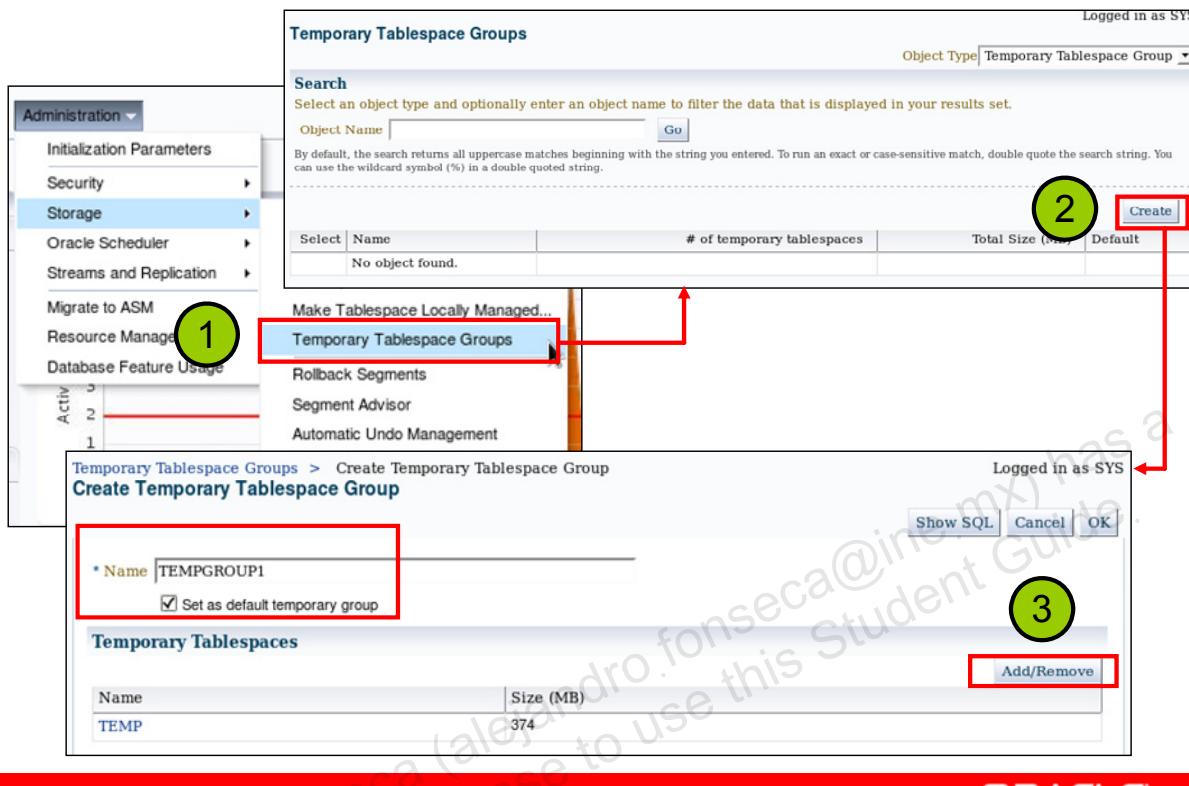
- Enables one particular user to use multiple temporary tablespaces in different sessions at the same time
- Enables the slave processes in a single parallel operation to use multiple temporary tablespaces
- Enables multiple default temporary tablespaces to be specified at the database level. Sort operations are assigned to these tablespaces in a round-robin fashion, spreading I/O across tablespaces and possibly across devices.

You can define more than one default temporary tablespace and a single parallel SQL operation can use more than one temporary tablespace for sorting. This prevents large tablespace operations from running out of temporary space. This is especially relevant with bigfile tablespaces.

The primary purpose of a temporary tablespace group is to increase the addressability of the temporary space. A single temporary tablespace can have a maximum of four billion blocks. This is 8 TB for a block size of 2 KB and 64 TB for a block size of 16 KB. With temporary tablespace groups, the addressability is increased to several petabytes.

Note: A single, non-parallel SQL statement uses only one temporary tablespace; a single temporary segment does not span multiple tablespaces.

Creating Temporary Tablespace Groups



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

ORACLE

You can create and maintain temporary tablespace groups by using Enterprise Manager Cloud Control.

1. Expand the Administration menu, expand the Storage menu, and click Temporary Tablespace Groups.
2. The Temporary Tablespace Groups page, where you can see a list of existing tablespace groups. On the Temporary Tablespace Groups page, you can view and edit existing tablespace groups. When you click the Create button, the Create Temporary Tablespace Group page appears.
3. Enter the name of the new group in the Name field, and specify whether or not you want this new group to be set as the default temporary group. You can do this by selecting the "Set as default temporary group" option. After selecting this option, you must add existing temporary tablespaces to the group. Click the Add/Remove button, and select the temporary tablespaces that belong to the group. Then click the OK button to create the new temporary tablespace group.

Maintaining Temporary Tablespace Groups

The screenshot shows two pages from the Oracle Database Enterprise Manager Cloud Control interface:

- Temporary Tablespace Groups Page (Top):** Shows a list of temporary tablespace groups. A row for 'TEMPGROUP1' is selected. A red box highlights the 'Edit' button in the Actions menu. A green circle with the number '1' is placed over the 'Edit' button.
- Edit Temporary Tablespace Group Page (Bottom):** Shows the details for 'TEMPGROUP1'. It includes fields for 'Name' (set to 'TEMPGROUP1') and 'Set as default temporary group' (unchecked). Below this is a 'Temporary Tablespaces' section with a table:

Name	Size (MB)
TEMP	374
TEMP02	100

A green circle with the number '2' is placed over the 'Add/Remove' button in the 'Temporary Tablespaces' section.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can maintain the temporary tablespace groups by using Database Enterprise Manager Cloud Control. Expand the Administration menu, expand the Storage menu, and click Temporary Tablespace Groups. On the Temporary Tablespace Groups page, you can see the list of existing tablespace groups. To maintain a temporary tablespace group:

1. Select the tablespace group that you want to maintain, and click the Edit button.
2. On the Edit Temporary Tablespace Group page, you can add and remove temporary tablespaces by clicking the Add/Remove button. After doing so, click the Apply button for your changes to take effect.

Note: If you remove all temporary tablespaces from a temporary tablespace group, the group is also removed implicitly.

Viewing Tablespace Groups

Temporary Tablespace Groups > View Temporary Tablespace Group: TEMPGROUP1
Logged in as SYS
View Temporary Tablespace Group: TEMPGROUP1

Name TEMPGROUP1
Set as default temporary group No

Temporary Tablespaces

Name	Size (MB)
TEMP02	100
TEMP	374

Actions Create Like Go Edit Return

```
SQL> SELECT group_name, tablespace_name
  2  FROM dba_tablespace_groups;
```

GROUP_NAME	TABLESPACE_NAME
-----	-----
TEMPGROUP1	TEMP
TEMPGROUP1	TEMP02

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can view temporary tablespace group information in Enterprise Manager Cloud Control. You can also use the DBA_TABLESPACE_GROUPS view, which lists all tablespaces contained inside each temporary tablespace group.

Note: The three views ALL_USERS, USER_USERS, and DBA_USERS have a column named TEMPORARY_TABLESPACE. This column contains either the name of a tablespace or the name of a temporary tablespace group.

Monitoring Temporary Tablespace

- Use V\$TEMPSEG_USAGE to monitor space usage and workload distribution:

```
SELECT session_num, username, segtype, blocks, tablespace
FROM   v$tempseg_usage;
```

- Use V\$SORT_SEGMENT to determine space usage percentage:

```
SELECT (s.tot_used_blocks/f.total_blocks)*100 as pctused
FROM (SELECT SUM(used_blocks) tot_used_blocks
      FROM v$sort_segment
      WHERE tablespace_name='TEMP') s,
     (SELECT SUM(blocks) total_blocks
      FROM dba_temp_files
      WHERE tablespace_name='TEMP') f;
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The distribution of workload can be monitored through the V\$TEMPSEG_USAGE dictionary view. The SEGTYPE column in the first example indicates what type of segment is using the space. It is one of SORT, HASH, INDEX, LOB_DATA, LOB_INDEX, and DATA. The BLOCKS column gives you the number of blocks currently used by the operation on disk. On the basis of the distribution of segment sizes, an appropriate extent size can be chosen.

The query in the second example can be used to show the percentage of space used in a locally managed temporary tablespace.

Note: The temporary segment of a given temporary tablespace is created at the time of the first operation that has to write to disk to free up space in memory. Multiple transactions that need space on disk can share the same segment; however, they cannot share the same extent. The segment expands by allocating new extents. The extents are not deallocated while the instance is running, but are marked as free and can be reused as required. Therefore, the segment grows to a certain steady state or until it fills the tablespace.

Shrinking a Temporary Tablespace

- Sort segment extents are managed in memory after being physically allocated.
- This method can be an issue after big sorts are done.
- To release physical space from your disks, you can shrink temporary tablespaces:
 - Locally managed temporary tablespaces
 - Online operation

```
CREATE TEMPORARY TABLESPACE temp
TEMPFILE 'tbs_temp.dbf' SIZE 600m REUSE AUTOEXTEND ON MAXSIZE
UNLIMITED
EXTENT MANAGEMENT LOCAL UNIFORM SIZE 1m;

ALTER TABLESPACE temp SHRINK SPACE [KEEP 200m];

ALTER TABLESPACE temp SHRINK TEMPFILE 'tbs_temp.dbf';
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Huge sorting operations can cause a temporary tablespace to grow a lot. For performance reasons, after a sort extent is physically allocated, it is managed in memory to avoid physical deallocation later. As a result, you can end up with a huge temp file that stays on disk until it is dropped. One possible workaround is to create a new temporary tablespace with smaller size, set this new tablespace as the default temporary tablespace for users, and then drop the old tablespace. However, there is a disadvantage in that the procedure requires no active sort operations at the time the old temporary tablespace is dropped.

You can use the `ALTER TABLESPACE SHRINK SPACE` command to shrink a temporary tablespace, or you can use the `ALTER TABLESPACE SHRINK TEMPFILE` command to shrink one temp file. For both commands, you can specify the optional `KEEP` clause that defines the lower bound that the tablespace/temp file can be shrunk to. If you omit the `KEEP` clause, the database attempts to shrink the tablespace/temp file as much as possible (total space of all currently used extents) as long as other storage attributes are satisfied. This operation is done online. However, if some currently used extents are allocated above the shrink estimation, the system waits until they are released to finish the shrink operation.

Note: The `ALTER DATABASE TEMPFILE RESIZE` command generally fails with ORA-03297 because the temp file contains used data beyond requested `RESIZE` value. As opposed to `ALTER TABLESPACE SHRINK`, the `ALTER DATABASE` command does not try to deallocate sort extents after they are allocated.

Using the Tablespace Option when Creating a Temporary Table

- Specify which temporary tablespace to use for your global temporary tables.
- Decide a proper temporary extent size.

```
CREATE TEMPORARY TABLESPACE temp
TEMPFILE 'tbs_temp.dbf' SIZE 600m REUSE AUTOEXTEND ON MAXSIZE
UNLIMITED
EXTENT MANAGEMENT LOCAL UNIFORM SIZE 1m;
```

```
CREATE GLOBAL TEMPORARY TABLE temp_table (c varchar2(10))
ON COMMIT DELETE ROWS TABLESPACE temp;
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can specify a TABLESPACE clause when you create a global temporary table. If no tablespace is specified, the global temporary table is created in your default temporary tablespace. In addition, indexes created on the temporary table are also created in the same temporary tablespace as the temporary table.

This allows you to decide a proper extent size that reflects your sort-specific usage, especially when you have several types of temporary space usage.

Note: You can find which tablespace is used to store your global temporary tables by querying DBA_TABLES.

Quiz

You can think of a row source used by the optimizer as a table in memory. It can be the projection of a table with columns limited by the select list and rows by the WHERE clause or the results of join and sort operations. Some of these row sources must be complete (all rows retrieved) before going to the next operation. Identify the place where these intermediate row sources are stored first.

- a. Sort area memory
- b. SQL cache
- c. Global temporary tables
- d. Temporary tablespaces
- e. Data files



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: a

Summary

In this lesson, you should have learned how to:

- Diagnose PGA memory issues
- Size the PGA memory
- Diagnose temporary space issues
- Specify temporary tablespace parameters for efficient operation

Practice 18: Tuning PGA Memory

This practice covers tuning `PGA_AGGREGATE_TARGET` to improve sort performance



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2018, Oracle and/or its affiliates.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

Using Automatic Memory Management

19

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

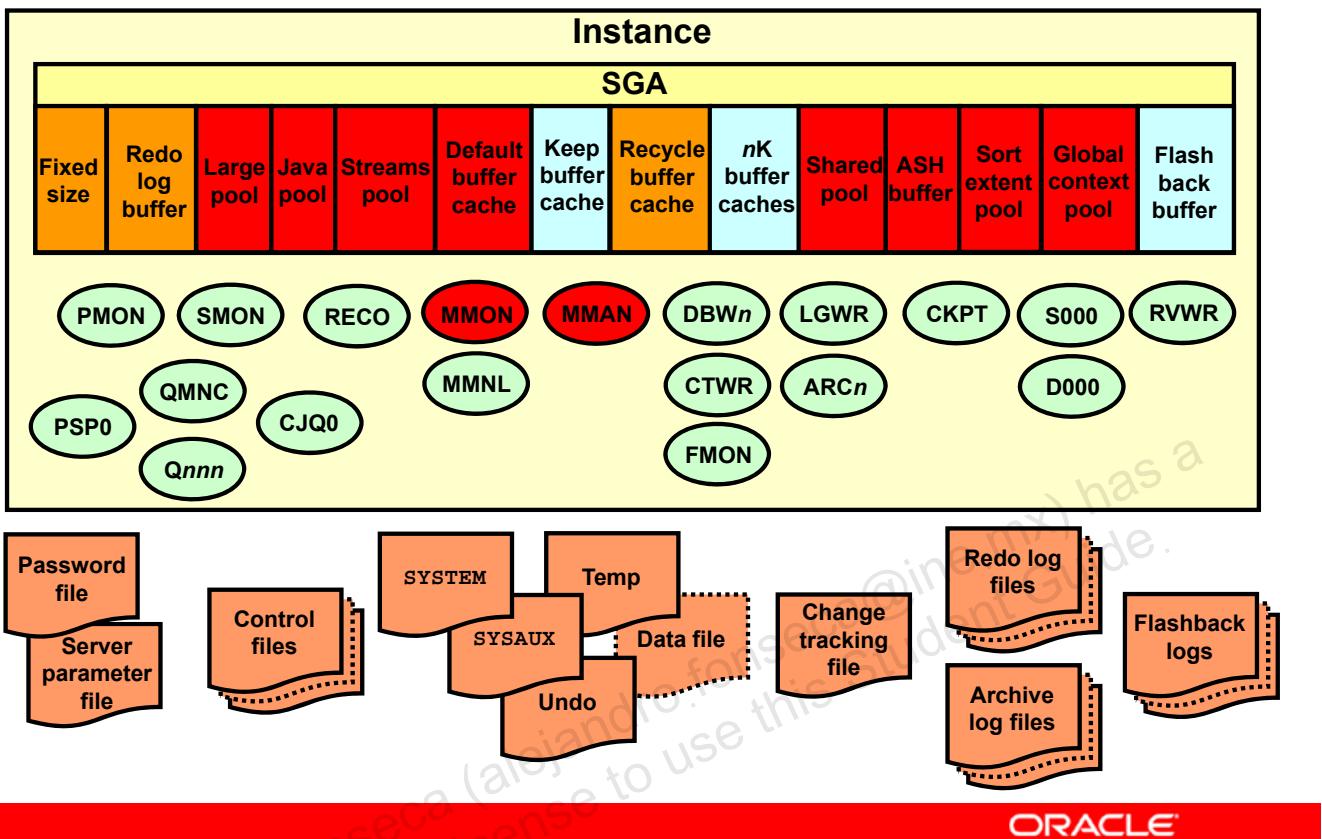
After completing this lesson, you should be able to do the following:

- Use memory advisors to size dynamic memory areas
- Enable Automatic Shared Memory Manager
- Configure memory parameters by using Enterprise Manager
- Set the minimum size for auto-tuned SGA components
- Use the SGA Advisor to set `SGA_TARGET`
- Enable Automatic Memory Management
- Use the Memory Advisor to set overall memory parameters



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Oracle Database Architecture



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The slide shows the key SGA components and processes that are involved in Automatic Shared Memory Management (ASMM).

The SGA components shared pool, large pool, Java pool, Streams pool, and the default buffer cache participate in ASMM. The monitoring and dynamic resizing are handled by two background processes: the memory manager (MMAN) and the manageability monitor (MMON).

The ASMM infrastructure is dependent on the statistics collected by the manageability monitor, so the `STATISTICS_LEVEL` parameter must be set to `TYPICAL` or `ALL` to use ASMM.

Note: The ASH buffer, sort extent pool, and the global context pool are all included in the shared pool.

Dynamic SGA

- Implements an infrastructure to allow the server to change its SGA configuration without shutting down the instance
- SGA size is limited by `SGA_MAX_SIZE`, which:
 - Reserves virtual memory address space at instance startup
 - Cannot be changed dynamically
- Allows for certain SGA components to be dynamically resized

```
SELECT bytes
  FROM v$sgainfo
 WHERE name = 'Free SGA Memory Available';
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The System Global Area (SGA) is internally divided into memory components. A component represents a pool of memory used to satisfy a particular class of memory allocation requests. The most commonly configured memory components include the database buffer cache, shared pool, large pool, Java pool, and Streams pool.

The dynamic SGA infrastructure allows for the resizing of certain components of the SGA without having to shut down the instance, modify the initialization parameter file, and restart the instance.

In addition, the dynamic SGA infrastructure allows limits to be set at run time on how much memory is used for the entire SGA. The parameter that limits the memory is `SGA_MAX_SIZE`. This is the amount of memory allocated at startup of the instance, regardless of whether the individual components utilize the entire amount of memory.

Note: During startup of the instance, if the sum of all the memory used by the SGA components exceeds the amount allotted by `SGA_MAX_SIZE`, the value of `SGA_MAX_SIZE` is adjusted to meet the memory requirement.

Granules

- SGA memory is allocated in units of contiguous memory chunks called “granules.”
- The size of a granule depends on the estimated total SGA and the version of the database.

```
SELECT bytes
  FROM v$sgainfo
 WHERE name = 'Granule Size';
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The SGA memory is allocated in granules. The number of granules used by each SGA component is recorded in dynamic performance views.

Granule size is determined by total SGA size. The size of a granule can vary with the version of the database and the platform. See My Oracle Support Note 947152.1 *How to determine granule size*.

The granule size that is currently being used for the SGA for each component can be viewed in the V\$SGAINFO view.

Memory Advisories

- Buffer Cache Advice:
 - V\$DB_CACHE_ADVICE
 - Predicts physical reads for different cache sizes
- Shared Pool Advice:
 - V\$SHARED_POOL_ADVICE
 - Predicts parse time savings from having different sizes of the shared pool
- Java Pool Advice:
 - V\$JAVA_POOL_ADVICE
 - Predicts Java class load time savings from having different sizes of Java pool
- Streams Pool Advice
 - V\$STREAMS_POOL_ADVICE
 - Predicts spill and unspill activity for various sizes



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Several advisories help you to size the most important SGA components:

- V\$DB_CACHE_ADVICE contains rows that predict the number of physical reads for the cache size corresponding to each row.
- V\$SHARED_POOL_ADVICE displays information about the estimated parse time in the shared pool for different pool sizes.
- V\$JAVA_POOL_ADVICE displays information about the estimated parse time in the Java pool for different pool sizes.
- V\$STREAMS_POOL_ADVICE displays information about the estimated count of spilled or unspilled messages and the associated time spent in the spill or unspill activity for different Streams pool sizes.

Note: For more information about these views, refer to the *Oracle Database Reference* guide.

Manually Adding Granules to Components

- Use the `ALTER SYSTEM` command to dynamically increase memory allocation to a component.
- Increasing the memory use of a component succeeds only if there are enough free granules to satisfy the request.
- Memory granules are not freed automatically from another component to satisfy the increase.
- Decreasing the size of a component is possible, but only if the granules being released are unused by the component.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can increase the size of the shared pool or any buffer cache component by issuing an `ALTER SYSTEM` command. The new size is rounded up to the nearest multiple of the granule size.

Increasing the memory use of a component with an `ALTER SYSTEM` command succeeds only if there are enough free granules (`SGA_MAX_SIZE` minus current size of all the SGA components) to satisfy the request. The server does not free another component's granules to enable an SGA component to grow. You must ensure that the instance has enough free granules to satisfy the increase of a component's granule use. If there are free granules available, then the server can allocate more granules until the SGA size reaches `SGA_MAX_SIZE`.

Note: When resizing SGA components, remember that there is a portion of the SGA that is fixed. This fixed memory will have a minimum of one granule.

Increasing the Size of an SGA Component

```
SQL> show parameter

NAME          TYPE        VALUE
-----
sga_max_size  big integer 200M
shared_pool_size  big integer 84M
db_cache_size   big integer 92M

SQL> alter system set shared_pool_size=120M;
alter system set shared_pool_size=120M
*
ERROR at line 1:
ORA-02097: parameter cannot be modified ...
ORA-04033: Insufficient memory to grow pool

SQL> alter system set db_cache_size=50M;
System altered.

SQL> alter system set shared_pool_size=120M;
System altered.
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

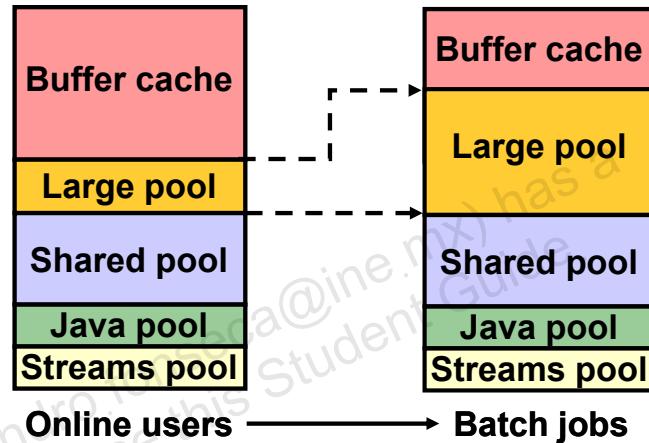
For the example in the slide, the starting point for the memory allocations is shown in this table:

NAME	TYPE	VALUE
sga_max_size	big integer	200M
db_cache_size	big integer	92M
java_pool_size	big integer	4M
large_pool_size	big integer	4M
shared_pool_size	big integer	84M
streams_pool_size	big integer	8M

In the example, memory is made available for the shared pool by first shrinking the buffer cache.

Automatic Shared Memory Management: Overview

- Uses dynamic SGA and memory advisors to automatically adapt to workload changes
- Maximizes memory utilization
- Helps to eliminate out-of-memory errors
- Avoids relearning when using SPFILE



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

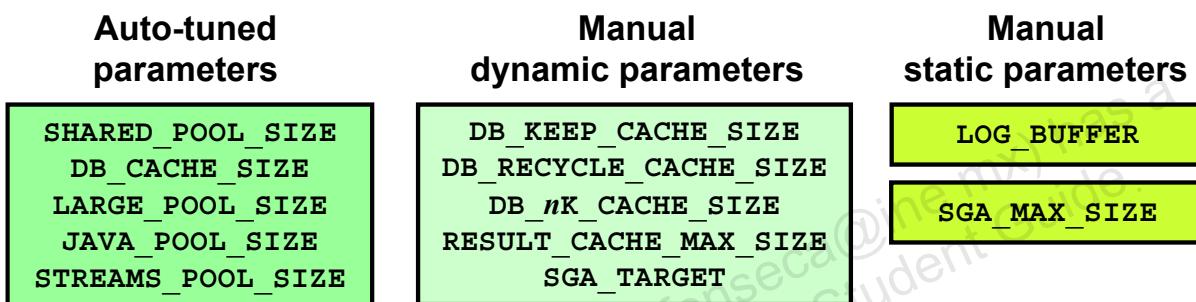
Automatic Shared Memory Management (ASMM) simplifies the configuration of the System Global Area (SGA). ASMM uses memory advisor data to evaluate the best memory configuration, and then resizes the amount of memory to be allocated for the database buffer cache, shared pool, Java pool, large pool, and Streams pool by using the dynamic SGA feature. ASMM makes more effective use of available memory and thereby reduces the cost incurred for acquiring additional hardware memory resources, and significantly simplifies Oracle database administration with a more dynamic, flexible, and adaptive memory management scheme.

ASMM enables you to specify a total memory amount to be used for all SGA components. The Oracle Database server periodically redistributes memory between the components given in the slide according to workload requirements. For example, in a system that runs high-concurrency OLTP workload during the day, which requires a large buffer cache, you would have to configure both the buffer cache and the large pool to accommodate your peak requirements. With ASMM, when the OLTP workload runs, the buffer cache is given the required memory to optimize buffer cache access. When the decision support system (DSS) batch job starts up later, the memory is automatically migrated to the large pool so that it can be used by parallel query operations without producing memory overflow errors.

Note: With ASMM, component sizes are saved across a shutdown if an SPFILE is used. The sizes are resurrected from before the last shutdown to avoid relearning.

SGA Sizing Parameters: Overview

- With ASMM, five important SGA components can be automatically sized.
- Nondefault buffer pools are not auto-tuned.
- Log buffer is not a dynamic component, but has a good default.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You have exact control over the total size of the SGA because memory that is allocated for the fixed SGA and for other internal metadata allocations is included in the total size of the user-specified SGA parameters.

Setting the `SGA_TARGET` parameter to a nonzero value enables ASMM. The `SGA_TARGET` initialization parameter includes all memory in the SGA, including the automatically sized components, the manually sized components, and any internal allocations during startup. If `SGA_MAX_SIZE` is set, it must be equal to or larger than `SGA_TARGET`.

As shown in the slide, the five most important pools are automatically tuned when ASMM is activated. The parameters for these pools are referred to as “auto-tuned parameters.”

“Manual dynamic parameters” are parameters that can be manually resized without having to shut down the instance, but are not automatically tuned by the system.

“Manual static parameters” are parameters that are fixed in size, and cannot be resized without first shutting down the instance.

Dynamic SGA Transfer Modes

- ASMM IMMEDIATE transfer mode:
 - Out-of-memory (ORA-04031) errors
 - Partial granules can be used.
- ASMM DEFERRED transfer mode:
 - Transparently executed in the background
 - Partial granules can be used.
- MANUAL transfer mode:
 - Used with ALTER SYSTEM commands
 - Resize must use full granules.

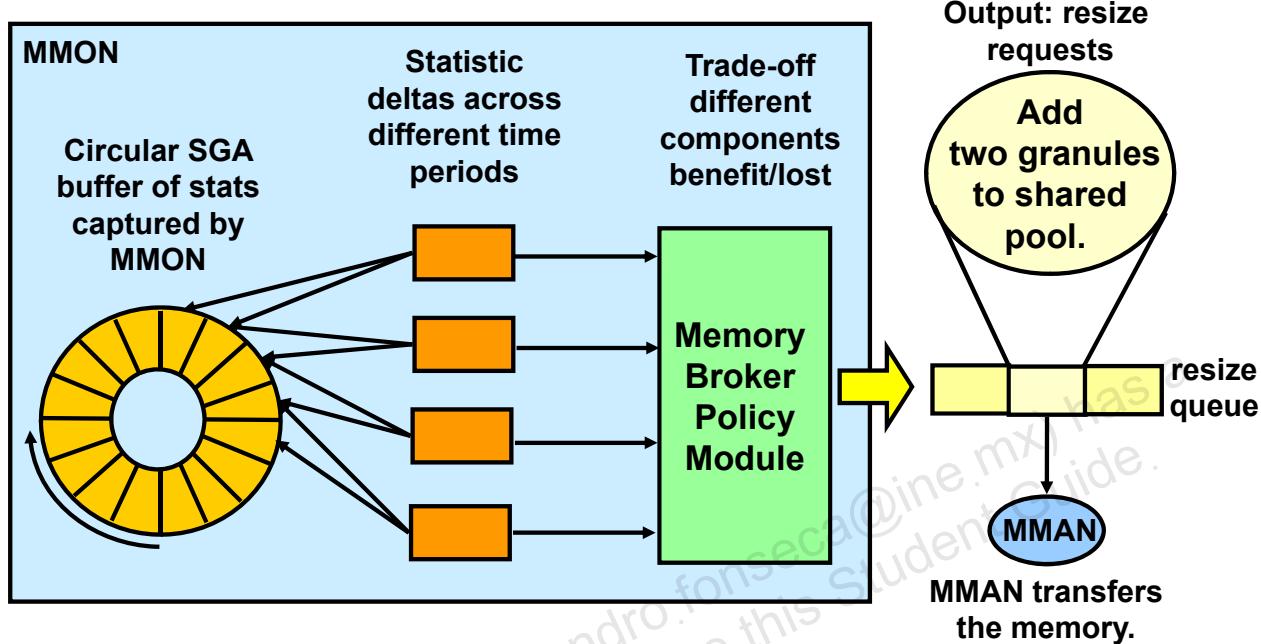


Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

There are three types of granule transfer mechanisms between dynamic SGA components:

- IMMEDIATE: This mode is used when ASMM is activated and one of the auto-tuned components is about to get an out-of-memory error. To prevent the error from happening, the system tries to transfer a granule from another component. This transfer can be a partial transfer if there are no entirely empty granules available. If that is the case, the system starts cleaning out granules from other components to satisfy the memory request, and partially transfer a granule to the component requesting for memory.
- DEFERRED: This mode is used by the system to transfer memory between components when it determines there is a more efficient memory distribution. Advisory-related data is used to determine the optimal memory distribution.
- MANUAL: This mode is used when you are asking the system to resize components. This mode can use only empty granules for the resize operation. If there are no empty granules, an ORA-4033 error is returned for a grow request and an ORA-4034 error is returned for a shrink request.

Memory Broker Architecture



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When ASMM is enabled, the memory broker periodically performs the activities in this slide. These operations are in DEFERRED mode. Its goal is to automate the sizing of auto-tuned components to adapt to workload changes by distributing the memory to where it is most needed. The transfer is done when there is an overall benefit to do so.

During this background action, statistics and memory advisory data are periodically captured in a circular buffer by MMON.

Deltas between different buffer entries represent statistics for different time periods. These deltas are computed by MMON.

MMON uses the Memory Broker policy to analyze the deltas and examines both long-term and short-term trends.

MMON generates resize decisions based on this analysis by posting requests in the resize request system queue. MMAN scans the request system queue periodically to execute the corresponding memory transfers.

Manually Resizing Dynamic SGA Parameters

- For auto-tuned parameters, manual resizing:
 - Results in immediate component resize if the new value is greater than the current size
 - Changes the minimum size if the new value is smaller than the current size
- Manually tuned parameter resizing affects the tunable portion of the SGA.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When an auto-tuned parameter is resized and `SGA_TARGET` is set, the resize results in an immediate change to the size of the component only if the new value is larger than the current size of the component. For example, if you set `SGA_TARGET` to 8 GB and set `SHARED_POOL_SIZE` to 2 GB, then you ensure that the shared pool has at least 2 GB at all times to accommodate the necessary memory allocations. After this, adjusting the value of `SHARED_POOL_SIZE` to 1 GB has no immediate effect on the size of the shared pool. It simply allows the automatic memory-tuning algorithm to later reduce the shared pool size to 1 GB if it needs to. However, if the size of the shared pool is 1 GB to begin with, then adjusting the value of `SHARED_POOL_SIZE` to 2 GB results in the shared pool component growing to a size of 2 GB. The memory used in this resize operation is taken away from one or more auto-tuned components, and the sizes of the manual components are not affected.

Parameters for manually sized components can be dynamically altered as well, but the difference is that the value of the parameter specifies the precise size of that component immediately. Therefore, if the size of a manual component is increased, extra memory is taken away from one or more automatically sized components. If the size of a manual component is decreased, the memory that is released is given to the automatically sized components.

Behavior of Auto-Tuned SGA Parameters

- When SGA_TARGET is not set or is set to zero:
 - Auto-tuned parameters are explicitly set
Note: SHARED_POOL_SIZE includes internal startup overhead.
- When SGA_TARGET is set to a nonzero value and auto-tuned parameters are:
 - Not set, the auto-tuned parameters default to zero
 - Set, a nonzero value is a lower bound
- Current values in megabytes are shown by:

```
SELECT component, current_size/1024/1024
FROM v$sga_dynamic_components;
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When SGA_TARGET is not set or is equal to zero, auto-tuned SGA parameters must be explicitly set. SHARED_POOL_SIZE is an exception. Internal overhead allocations for metadata (data structures for processes, sessions, and so on) are included as part of the value of the SHARED_POOL_SIZE parameter.

When SGA_TARGET is set to a nonzero value, the auto-tuned SGA parameters have default values of zero. These components are automatically sized by the ASMM algorithm. However, if they are set to nonzero values, the specified values are used as a lower bound. For example, if SGA_TARGET is set to 8 GB and SHARED_POOL_SIZE is set to 1 GB, the shared pool should not shrink below 1 GB, but it may grow to larger values. You can use the query to determine the actual size of the auto-tuned components in the SGA.

Behavior of Manually Tuned SGA Parameters

- Manually tuned components are:
 - KEEP and RECYCLE buffer caches
 - Nondefault block size caches
 - LOG_BUFFER
- Manually tuned components are user specified.
- Manually tuned components are included in SGA_TARGET to precisely control the SGA size.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The manual SGA size parameters are:

- DB_KEEP_CACHE_SIZE
- DB_RECYCLE_CACHE_SIZE
- DB_nK_CACHE_SIZE ($n = 2, 4, 8, 16, 32$)
- LOG_BUFFER

Manual SGA parameters are specified by the user, and the given sizes precisely control the sizes of their corresponding components.

When SGA_TARGET is set, the total size of manual SGA size parameters is subtracted from the SGA_TARGET value, and the balance is given to the auto-tuned SGA components.

For example, if SGA_TARGET is set to 8 GB and DB_KEEP_CACHE_SIZE is set to 1 GB, the total size of the five auto-tuned components (shared pool, Java pool, default buffer cache, large pool, and Streams pool) is limited to 7 GB. The 7 GB size includes the fixed SGA and log buffer. Only after those have been allocated is the rest of the memory divided between the auto-tuned components. The size of the keep cache is 1 GB, as specified by the parameter.

Using the V\$PARAMETER View

```
SGA_TARGET = 8G
```

```
SELECT name, value, isdefault
FROM v$parameter
WHERE name LIKE '%size';
```



NAME	VALUE	ISDEFAULT
shared_pool_size	0	TRUE
large_pool_size	0	TRUE
java_pool_size	0	TRUE
streams_pool_size	0	TRUE
db_cache_size	0	TRUE

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When you specify a nonzero value for SGA_TARGET and do not specify a value for an auto-tuned SGA parameter, the value of the auto-tuned SGA parameters in the V\$PARAMETER view is 0, and the value of the ISDEFAULT column is TRUE.

If you have specified a value for any of the auto-tuned SGA parameters, the value displayed when you query V\$PARAMETER is the value that you specified for the parameter.

Resizing SGA_TARGET

- The SGA_TARGET initialization parameter:
 - Is dynamic
 - Can be increased up to SGA_MAX_SIZE
 - Can be reduced until all components reach minimum size
 - Changes affect only automatically sized components
- SGA_TARGET includes everything in the SGA:
 - Fixed SGA and other internal allocations
 - Automatically sized SGA components
 - Manually sized SGA components
- SGA_TARGET allows precise sizing of the total shared memory allocation by the Oracle server.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

SGA_TARGET is a dynamic parameter and can be changed through Enterprise Manager or with the ALTER SYSTEM command.

SGA_TARGET can be increased up to the value of SGA_MAX_SIZE. It can be reduced until any one of the auto-tuned components reaches its minimum size: either a user-specified minimum or an internally determined minimum.

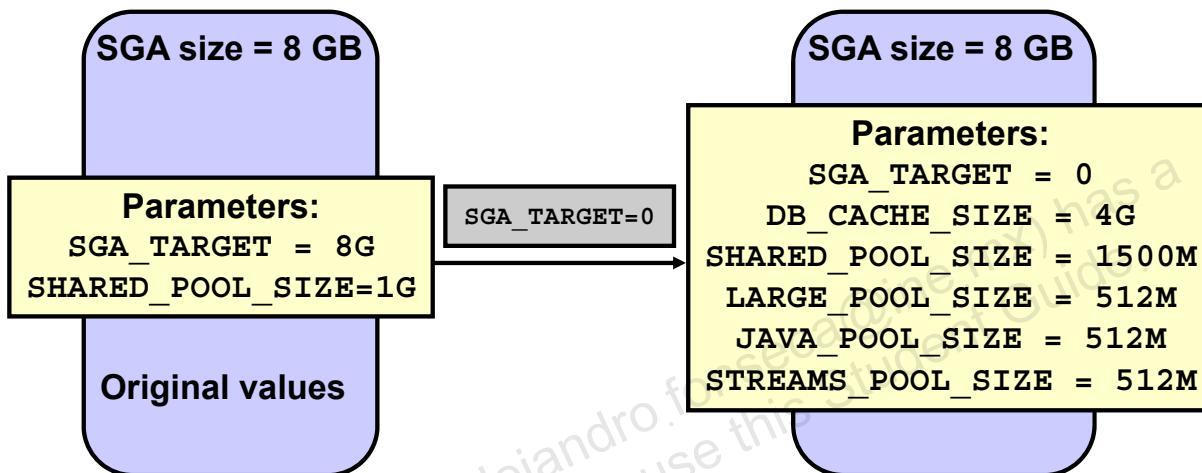
If you increase the value of SGA_TARGET, the additional memory is distributed according to the auto-tuning policy across the auto-tuned components.

If you reduce the value of SGA_TARGET, the memory is taken away by the auto-tuning policy from one or more of the auto-tuned components. Therefore, any change in the value of SGA_TARGET affects only the sizes of the auto-tuned components.

For example, SGA_MAX_SIZE is set to 10 GB and SGA_TARGET is set to 8 GB. If DB_KEEP_CACHE_SIZE is set to 1 GB and you increase SGA_TARGET to 9 GB, the additional 1 GB is distributed only among the components controlled by SGA_TARGET. The value of DB_KEEP_CACHE_SIZE is not affected. Likewise, if SGA_TARGET is reduced to 7 GB, the 1 GB is taken from only those components controlled by SGA_TARGET. This decrease does not change the minimum settings of parameters you explicitly set such as DB_KEEP_CACHE_SIZE.

Disabling Automatic Shared Memory Management

- Setting SGA_TARGET to zero disables auto-tuning.
- Auto-tuned parameters are set to their current sizes.
- SGA size as a whole is unaffected.



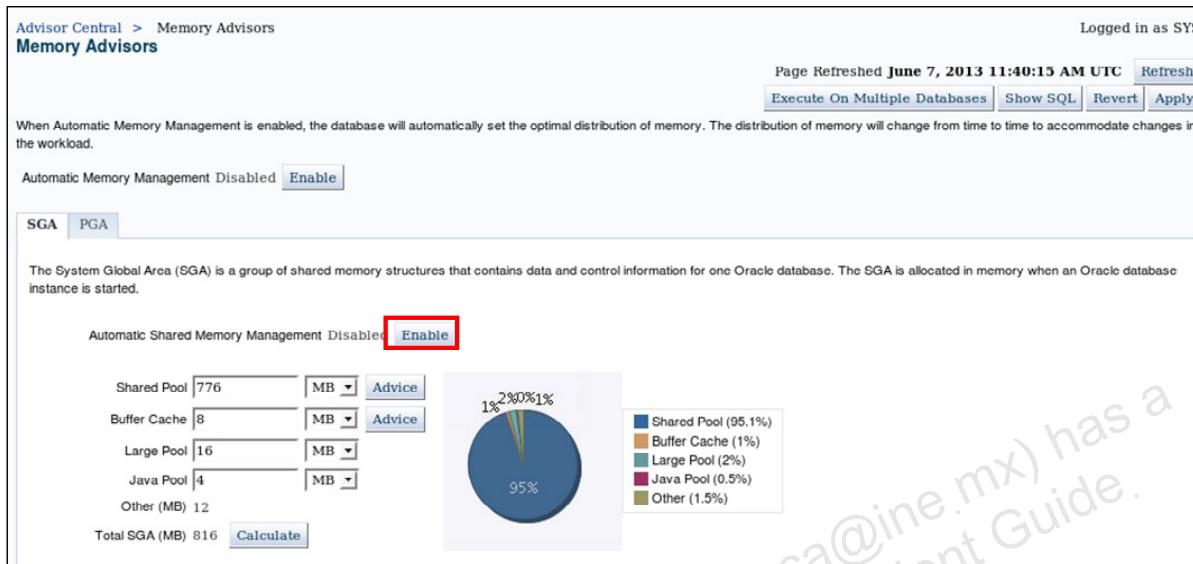
ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can dynamically disable Automatic Shared Memory Management by setting SGA_TARGET to zero. In this case, the values of all the auto-tuned parameters are set to the current sizes of the corresponding components, even if you had earlier specified a different nonzero value for an auto-tuned parameter.

In the example in the slide, the value of SGA_TARGET is 8 GB and the value of SHARED_POOL_SIZE is 1 GB. If the system has internally adjusted the size of the shared pool component to 1500 MB, setting SGA_TARGET to zero results in SHARED_POOL_SIZE being set to 1500 MB, thereby overriding the original user-specified value.

Enabling ASMM



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To use Enterprise Manager Cloud Control to enable Automatic Shared Memory Management:

1. Expand the Performance menu.
2. Click Memory Advisors.
3. Click the Enable button for Automatic Shared Memory Management.
4. On the Enable Automatic Shared Memory Management page, enter the total SGA size (in MB). This is the value that `SGA_TARGET` is set to.
5. Click OK.

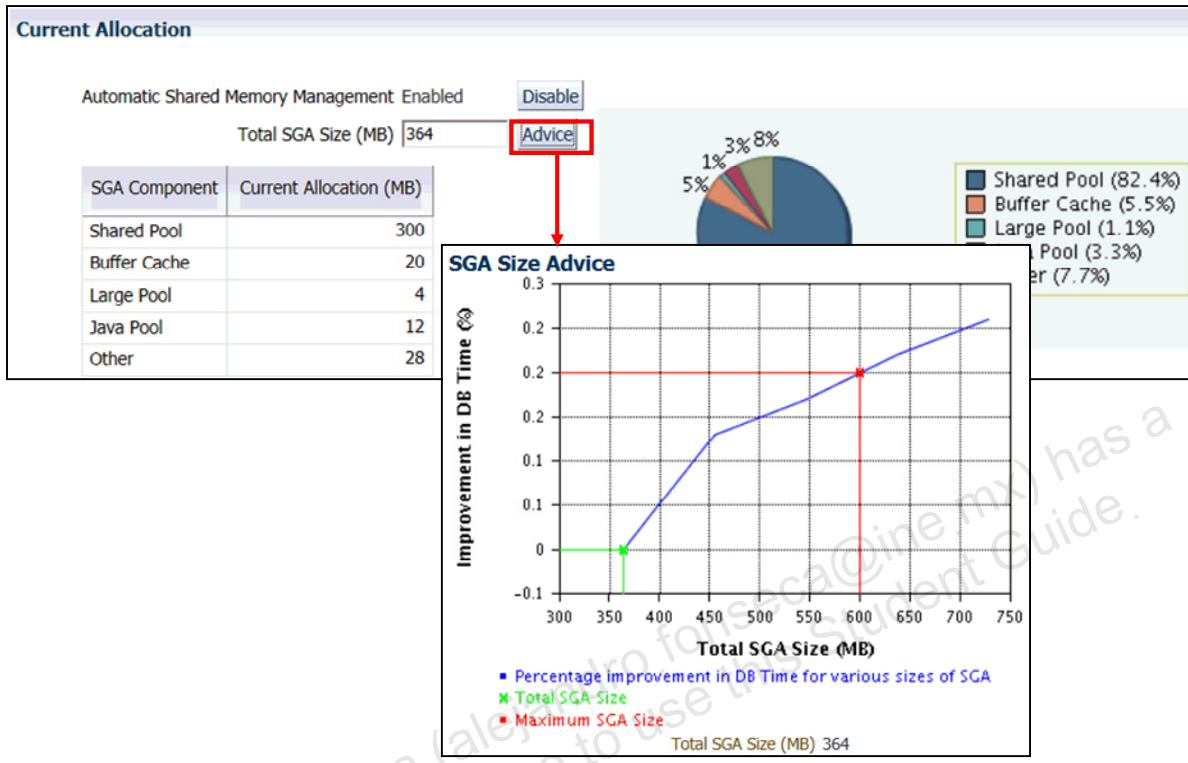
To disable Automatic Shared Memory Management:

1. On the Memory Advisors page, click Disable.
2. On the Disable Automatic Shared Memory Management page, set the desired component values or allow them to default to the current values.
3. Click OK.

In Oracle Database, the default for Automatic Shared Memory Management is Enabled. On the Memory Advisors page, you see a new chart that displays the history of the SGA allocation.

Note: The Streams pool size is not exposed through the Enterprise Manager interface.

Using the SGA Advisor



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The SGA Advisor shows the improvement in DB time that can be achieved if a total SGA size is specified. This advisor enables you to reduce trial and error efforts in setting the SGA size.

The advisor data is stored in the `v$sga_target_advice` table and includes the following:

- `SGA_SIZE`: Size of the SGA
- `SGA_SIZE_FACTOR`: Ratio between the `SGA_SIZE` and the current size of the SGA
- `ESTD_DB_TIME`: Estimated `DB_TIME` for this `SGA_SIZE`
- `ESTD_DB_TIME_FACTOR`: Ratio between `ESTD_DB_TIME` and `DB_TIME` for the current size of the SGA
- `ESTD_PHYSICAL_READS`: Estimated number of physical reads

Note: DB time is the same as the database time discussed in the lesson titled “Using Automatic Workload Repository.” Database time includes all the waits to perform an operation; in this case, the difference in the reads and writes required with a different `SGA_TARGET` size.

Monitoring ASMM

Monitor Automatic Shared Memory Management and examine the resize decisions it makes with the following views:

- V\$SGA_CURRENT_RESIZE_OPS: Information about resize SGA operation in progress
- V\$SGA_RESIZE_OPS: Circular history buffer of the last 800 SGA resize requests
- V\$SGA_DYNAMIC_COMPONENTS: Information about the dynamic SGA components
- V\$SGA_DYNAMIC_FREE_MEMORY: Information about SGA memory available for future resize operations



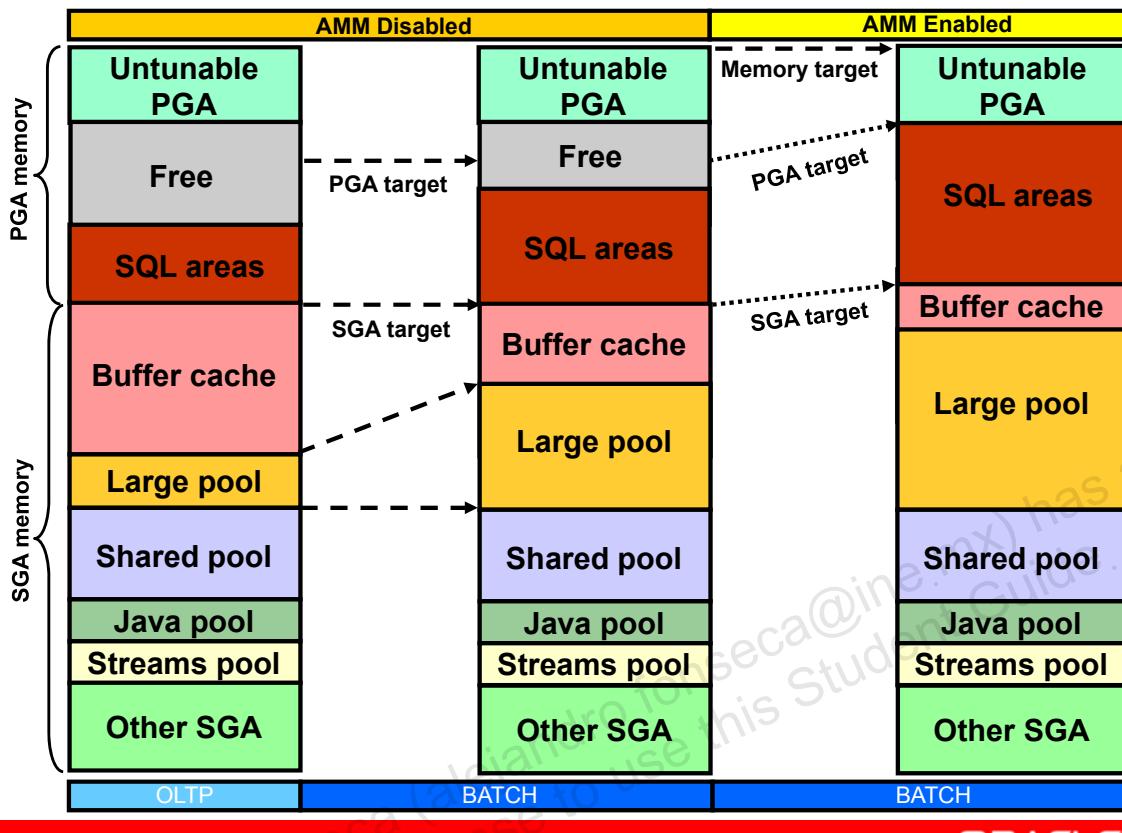
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The following views provide information about dynamic SGA resize operations:

- V\$SGA_CURRENT_RESIZE_OPS: Information about SGA resize operations that are currently in progress. An operation can be a grow or a shrink of a dynamic SGA component.
- V\$SGA_RESIZE_OPS: Information about the last 800 completed SGA resize operations. This does not include any operations currently in progress.
- V\$SGA_DYNAMIC_COMPONENTS: Information about the dynamic components in SGA. This view summarizes information based on all completed SGA resize operations since startup.
- V\$SGA_DYNAMIC_FREE_MEMORY: Information about the amount of SGA memory available for future dynamic SGA resize operations

Note: For more information about these views, refer to the *Oracle Database Reference* guide.

Automatic Memory Management: Overview



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

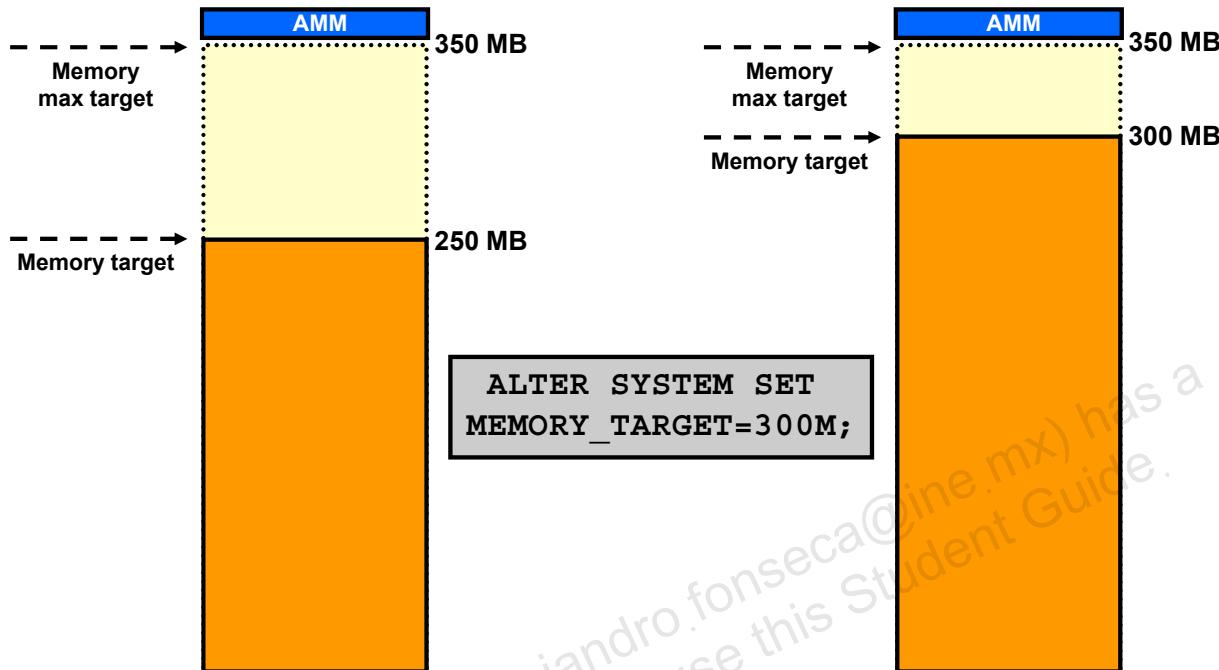
With Automatic Memory Management (AMM), the system causes an indirect transfer of memory from SGA to PGA (and vice versa). It automates the sizing of PGA and SGA according to your workload.

This indirect memory transfer relies on the OS mechanism of freeing shared memory. After memory is released to the OS, the other components can allocate memory by requesting memory from the OS.

Currently, this is implemented on Linux, Solaris, HP-UX, AIX, and Windows. Set your memory target for the database instance and the system then tunes to the target memory size, redistributing memory as needed between the system global area (SGA) and the aggregate program global area (PGA).

The slide displays the differences between the Automatic Shared Memory Management mechanism and the Automatic Memory Management.

Automatic Memory Management: Overview

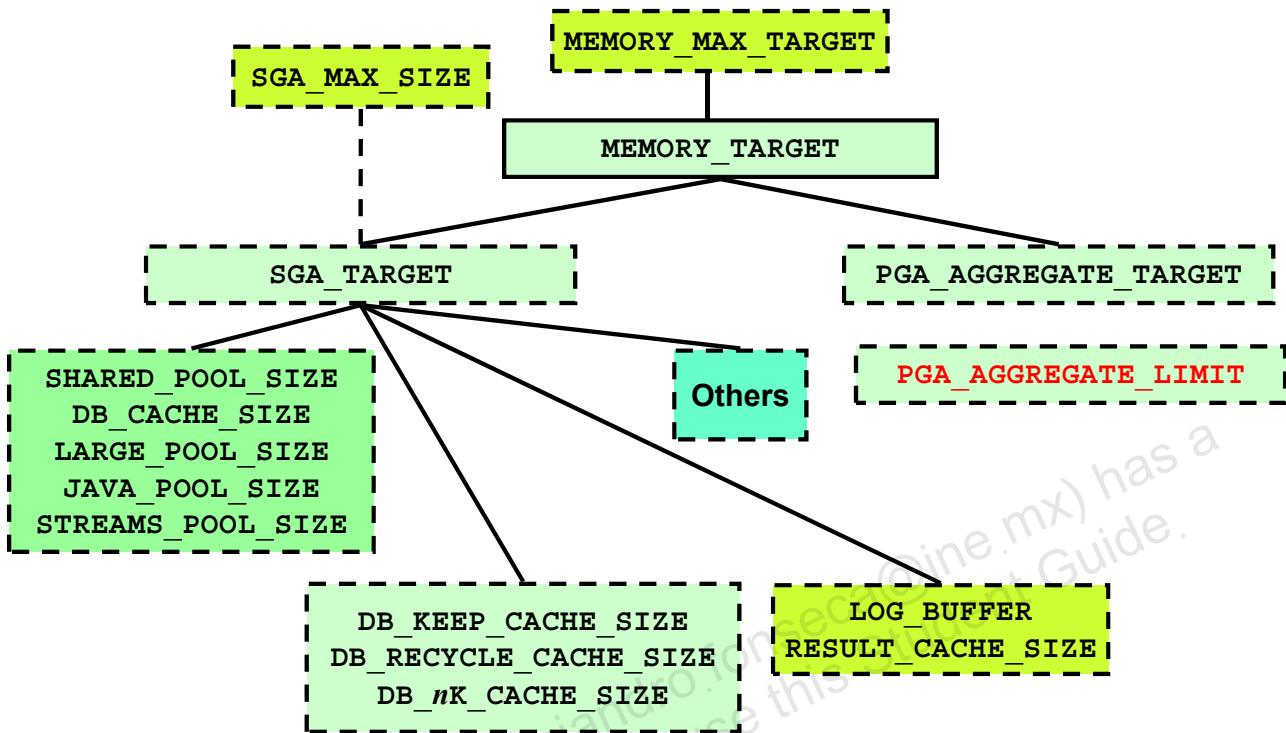


ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The simplest way to manage memory is to allow the database server to automatically manage and tune it for you. To do so (on most platforms), you have to set only a target memory size initialization parameter (`MEMORY_TARGET`) and a maximum memory size initialization parameter (`MEMORY_MAX_TARGET`). Because the target memory initialization parameter is dynamic, you can change the target memory size at any time without restarting the database. The maximum memory size serves as an upper limit so that you do not accidentally set the target memory size too high. Because certain SGA components either cannot easily shrink or must remain at a minimum size, the database also prevents you from setting the target memory size too low.

Oracle Database Memory Parameters



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The slide displays the memory initialization parameters hierarchy. Although you have to set only **MEMORY_TARGET** to trigger Automatic Memory Management, you still have the ability to set lower-bound values for various caches. So if the child parameters are set by the user, they will be the minimum values below which that component is not auto-tuned.

The SGA is made up of several components, some are auto-tuned when ASMM is enabled, some are dynamic, and some are static. When Automatic Memory management is enabled Automatic Shared Memory Management is also enabled.

PGA_AGGREGATE_LIMIT is a new parameter in Oracle Database 12c that sets an upper limit on PGA usage. See Lesson “Tuning PGA and Temporary Space for more detail.

Enabling Automatic Memory Management

The screenshot shows two consecutive pages from Oracle Enterprise Manager Cloud Control. The top page is titled 'Memory Advisors' under 'Advisor Central > Memory Advisors'. It contains a note about Automatic Memory Management and a button labeled 'Enable' which is highlighted with a red box. A red arrow points down to the second page. The second page is titled 'Enable Automatic Memory Management' under 'Advisor Central > Memory Advisors > Enable Automatic Memory Management'. It includes a note about memory distribution changing over time, a 'Maximum Memory Size' field set to 1024 MB, and a 'Total Memory Size for Automatic Memory Management' field set to 840 MB. At the bottom right are 'Cancel' and 'OK' buttons, with 'OK' also highlighted with a red box.

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

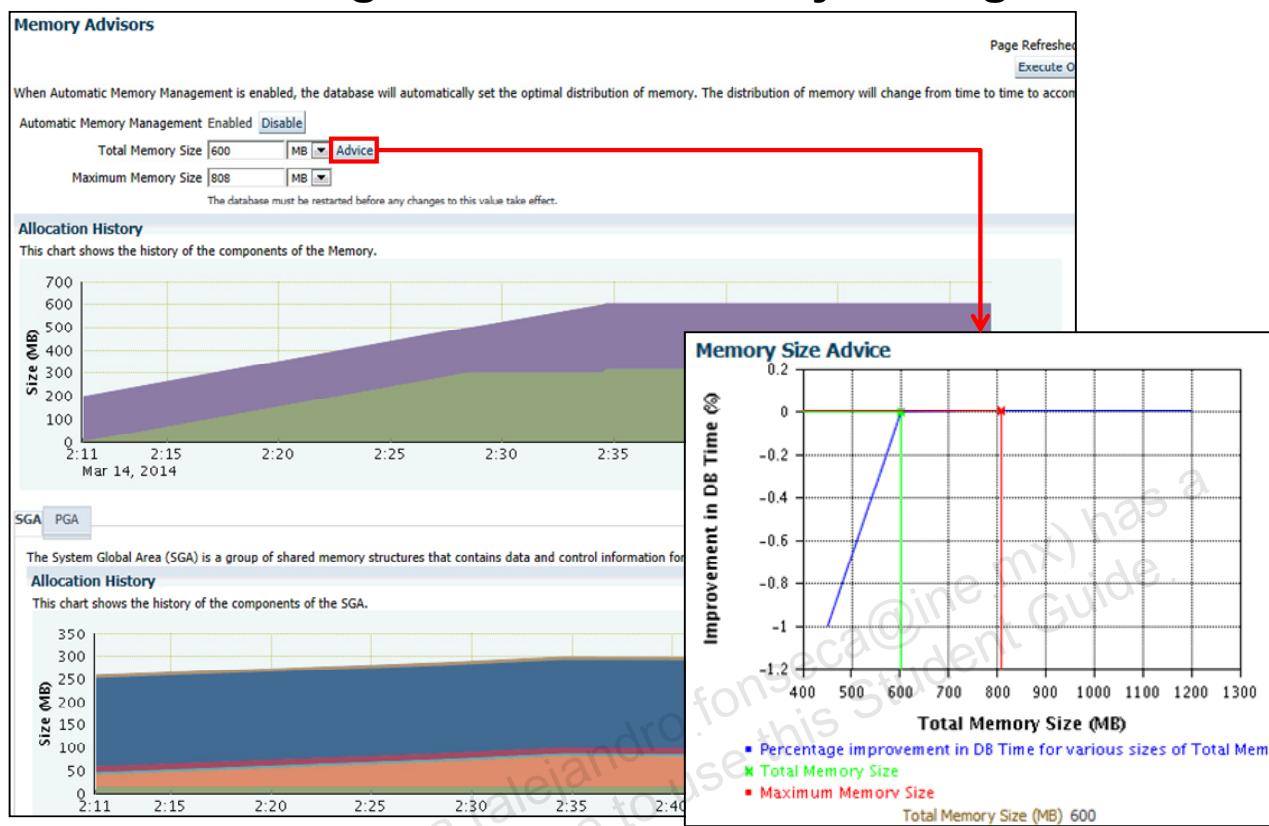
You can enable Automatic Memory Management by using Enterprise Manager Cloud Control as shown in the slide.

Expand the Performance menu and select Memory Advisors. On this page, you can click the Enable button to enable Automatic Memory Management.

The value in the “Total Memory Size for Automatic Memory Management” field is set by default to the current SGA + PGA size. You can set it to anything more than this but less than the value in Maximum Memory Size.

Note: On the Memory Advisors page, you can also specify the Maximum Memory Size. If you change this field, the database must be restarted for your change to take effect.

Monitoring Automatic Memory Management



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When Automatic Memory Management is enabled, you see a graphical representation of the history of your memory size components in the Allocation History section of the Memory Parameters page. The purple part in the first graphic represents the PGA and the green part is all of the SGA.

The change in the graph displays the possible repartition of memory after the execution of the various demanding queries or changes in the Total Memory Size (MEMORY_TARGET). Both SGA and PGA might grow or shrink. Note that when SGA shrinks, its subcomponents also shrink at the same time.

On this page, you can also access the Memory Size Advisor by clicking the Advice button. This advisor gives you the possible DB time improvement for various total memory sizes.

Note: V\$MEMORY_TARGET_ADVICE displays the tuning advice for the MEMORY_TARGET initialization parameter.

Monitoring Automatic Memory Management

To monitor the decisions made by Automatic Memory Management, use the following views:

- V\$MEMORY_DYNAMIC_COMPONENTS: Displays the current status of all memory components
- V\$MEMORY_RESIZE_OPS: Displays a circular history buffer of the last 800 completed memory resize requests
- V\$MEMORY_CURRENT_RESIZE_OPS: Displays current memory resize operations

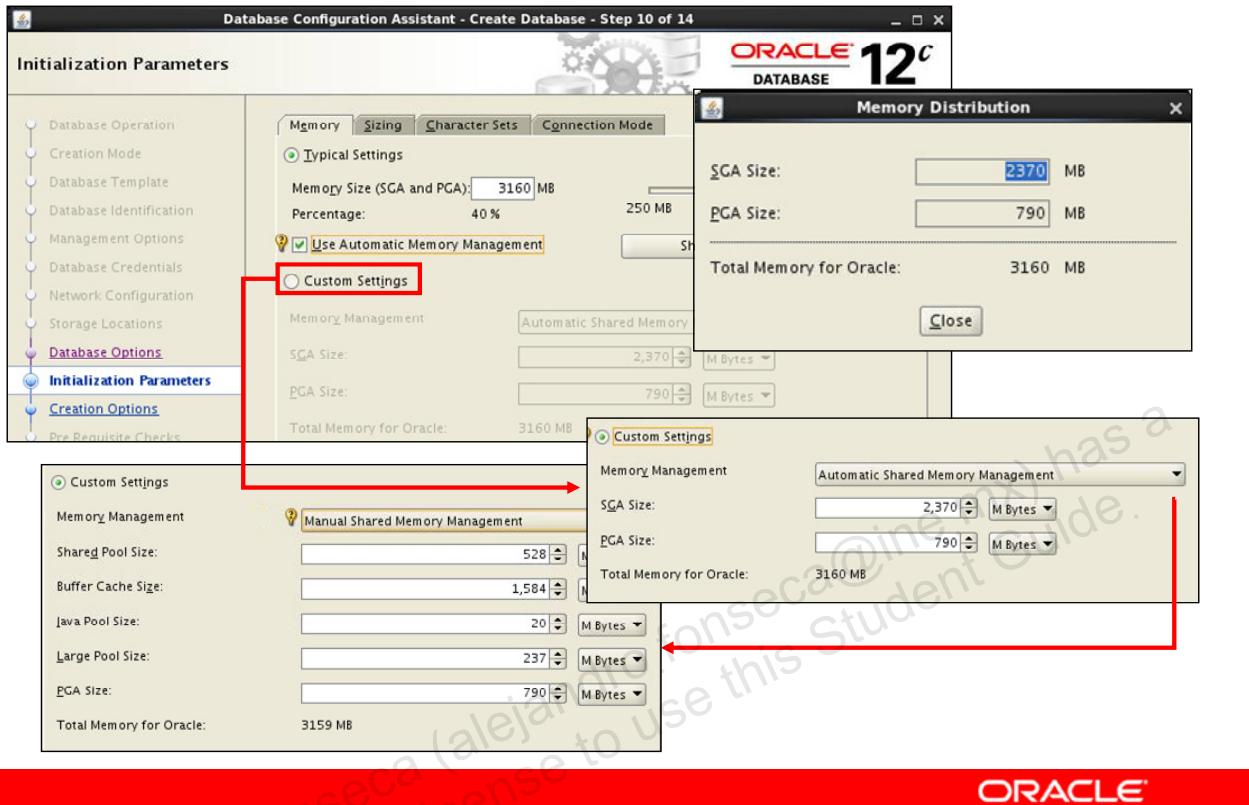


Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The following views provide information about dynamic resize operations:

- V\$MEMORY_DYNAMIC_COMPONENTS displays information about the current sizes of all dynamically tuned memory components, including the total sizes of the SGA and PGA.
- V\$MEMORY_RESIZE_OPS displays information about the last 800 completed memory resize operations (both automatic and manual). This does not include in-progress operations.
- V\$MEMORY_CURRENT_RESIZE_OPS displays information about the memory resize operations (both automatic and manual) that are currently in progress.
- V\$SGA_CURRENT_RESIZE_OPS displays information about SGA resize operations that are currently in progress. An operation can be a grow or a shrink of a dynamic SGA component.
- V\$SGA_RESIZE_OPS displays information about the last 800 completed SGA resize operations. This does not include operations currently in progress.
- V\$SGA_DYNAMIC_COMPONENTS displays information about the dynamic components in SGA. This view summarizes information based on all completed SGA resize operations since startup.
- V\$SGA_DYNAMIC_FREE_MEMORY displays information about the amount of SGA memory that is available for future dynamic SGA resize operations.

DBCA and Automatic Memory Management



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

ORACLE

DBCA has options to accommodate Automatic Memory Management (AMM). Use the Memory tab of the Initialization Parameters page to set the initialization parameters that control how the database manages its memory usage. You can choose from three basic approaches to memory management: Automatic Shared Memory Management (ASMM), Automatic Memory Management, or Manual.

- **Default:** The default database template configures the instance to use ASMM.

Advanced Mode Memory tab Options:

- **Typical:** Requires very little configuration and allows the database server to manage how it uses a percentage of your overall system memory. Select Typical to create a database with minimal configuration or user input. This option is sufficient for most environments and for DBAs who are inexperienced with advanced database creation procedures. Enter a value in megabytes in the Memory Size field. To use AMM, select the corresponding option in the Typical section of the page. Click Show Memory Distribution to see how much memory the DBCA assigns to both SGA and PGA when you do not select the AMM option.
- **Custom (uses ASMM or Manual):** Requires more configuration but provides you with more control over how the database server uses available system memory. To allocate specific amounts of memory to the SGA and PGA, select Automatic Shared Memory Management. To customize how the SGA memory is distributed among the SGA memory structures (buffer cache, shared pool, and so on), select Manual Shared Memory Management and enter specific values for each SGA subcomponent. Review and modify these initialization parameters later in DBCA.

Note: When you use manual DB creation, the MEMORY_TARGET parameter defaults to 0.

Quiz

When Automatic Memory Management is enabled:

- a. Individual pool parameters, such as DB_BUFFER_CACHE, set the maximum size for the pool
- b. All the pools are resized automatically including the keep and recycle caches
- c. PGA and auto-tuned components of SGA are resized as needed
- d. The redo log buffers are tuned as needed
- e. The SGA_MAX_SIZE parameter can be changed while the database instance is active

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: c

PGA and auto-tuned components of SGA are resized as needed when AMM is enabled.

The individual pool parameters set the minimum sizes.

Only the auto-tuned components of the SGA are resized automatically. The non-default buffer caches are not auto-tuned.

The redo log buffer is a static parameter that is set at startup.

The SGA_MAX_SIZE parameter is static and can be changed only by restarting the instance.

Summary

In this lesson, you should have learned how to:

- Use memory advisors to size dynamic memory areas
- Enable Automatic Shared Memory Management
- Configure memory parameters by using Enterprise Manager
- Set minimum size of auto-tuned SGA components
- Use the SGA Advisor to set `SGA_TARGET`
- Enable Automatic Memory Management
- Use the Memory Advisor to set overall memory parameters

Practice 19: Automatic Memory Tuning

This practice covers:

- Enabling Automatic Shared Memory Management
- Adjusting Memory as Workloads Change

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2018, Oracle and/or its affiliates.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

20

Performance Tuning Summary

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Diagnose and implement solutions for common wait events using the Performance Tuning Methodology
- Describe the Oracle Database Architecture related to common wait events
- Describe the Oracle Database Architecture related to common performance-related errors

Methodology

- Discover a symptom.
- Determine possible causes.
- Develop a solution.
- Implement the solution.
- Test the solution.
- Decision – Did the solution solve the problem?



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To summarize the performance tuning methodology as applied generally to any performance issue:

1. When a performance issue is suspected, gather the information necessary to determine the symptoms. This can include ADDM, AWR, alert logs, trace files, and other diagnostics.
2. Based on the diagnostic information, make a list of possible causes.
3. Develop a trial solution. This can be as simple as following an ADDM recommendation, or may require research from My Oracle Support or other sources.
4. Implement the solution. Some solutions should be implemented in a test environment to reduce the risk of worse performance.
5. Test the solution with the same workload, or with a workload as similar as possible. A reproducible test case is very helpful or captured workload using Database Replay.
6. Did the solution solve the problem?
 - No. Go back to step 3 and repeat.
 - Yes. Stop tuning.

Top 10 Mistakes Found in Customer Systems

1. Bad connection management
2. Bad use of cursors and shared pool
3. Bad SQL
4. Use of nonstandard initialization parameters
5. Getting database I/O wrong
6. Redo log setup problems
7. Serialization of data blocks in the buffer cache
8. Long full-table scans
9. High amount of recursive SQL
10. Deployment and migration errors



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

This slide lists the most common mistakes found in customer systems. The first three are application design or coding problems:

1. The application connects and disconnects for each database interaction. This problem is common with stateless middleware in application servers. It has more than two orders of magnitude impact on performance, and is not scalable.
2. Not using shared cursors results in repeated parses. If bind variables are not used, then there is hard parsing of all SQL statements. This has an order of magnitude impact in performance, and it is not scalable. Use cursors with bind variables that open the cursor and execute it many times. Be suspicious of applications generating dynamic SQL.
3. Bad SQL is SQL that uses more resources than appropriate for the application requirement. This can be a decision support system (DSS) query that runs for more than 24 hours or a query from an online application that takes more than a minute. Any SQL that consumes significant system resources should be investigated for potential improvement. ADDM identifies high-load SQL and the SQL Tuning Advisor can be used to provide recommendations for improvement.

4. These might have been implemented on the basis of poor advice or incorrect assumptions. Most systems give acceptable performance using only the set of basic parameters. In particular, parameters associated with undocumented optimizer features can cause a great deal of problems that can require considerable investigation. Likewise, optimizer parameters set in the initialization parameter file can override proven optimal execution plans. For these reasons, schemas, schema statistics, and optimizer settings should be managed together as a group to ensure consistency of performance.
5. Many sites lay out their databases poorly over the available disks. Other sites specify the number of disks incorrectly, because they configure the I/O subsystem by disk space and not I/O bandwidth.
6. Many sites run with too few redo logs that are too small. Small redo logs cause system checkpoints to continuously put a high load on the buffer cache and I/O system. If there are too few redo logs, the archive cannot keep up, and the database waits for the archive process to catch up.
7. Serialization of data blocks in the buffer cache due to lack of free lists, free list groups, transaction slots (`INITTRANS`), or shortage of rollback segments. This is particularly common on insert-heavy applications, in applications that have raised the block size above 8 KB, or in applications with large numbers of active users and few rollback segments. Using Automatic Segment Space Management (ASSM) and Automatic Undo Management solves many of these problems.
8. Long full-table scans for high-volume or interactive online operations could indicate poor transaction design, missing indexes, or poor SQL optimization. Long table scans, by nature, are I/O intensive and not scalable.
9. Large amounts of recursive SQL executed by `SYS` could indicate space management activities, such as extent allocations, taking place. This is not scalable and impacts user response time. Use locally managed tablespaces to reduce recursive SQL due to extent allocation. Recursive SQL executed under another user ID is probably SQL and PL/SQL, and this is not a problem.
10. In many cases, an application uses too many resources because the schema owning the tables has not been successfully migrated from the development environment or from an older implementation. Examples of this are missing indexes or incorrect statistics. These errors can lead to suboptimal execution plans and poor interactive user performance. When migrating applications of known performance, export the schema statistics to maintain plan stability by using the `DBMS_STATS` package. SQL Plan Baselines provide a method for migrating the execution plans and preventing plan regression.

Common Symptoms

Issues:

- Slow response times
- Intermittent “hangs”
- High CPU use
- High Disk I/O
- Instance failure

Classes of symptoms:

- Errors
- Wait events



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The slide lists several performance issues. These issues do not provide any clear direction for solving them. The next step is to find relevant diagnostics to direct the tuning efforts.

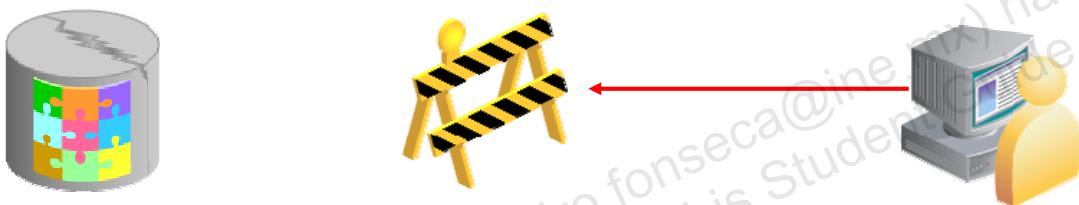
The top foreground wait events are typically the first diagnostics that we consider in the tuning methodology, unless there are errors. While there are a large number of errors, only a few are directly related to performance, but any error can be considered a performance problem if the instance does not start.

A few common errors are considered in this section with their causes and solutions. For a full listing of error messages, refer to *Oracle Database Error Messages 12c Release 1*.

There are over 1500 wait events in 12 categories listed in *Oracle Database Reference 12c Release 1*. Only a few are ever referenced in the Top 10 Foreground Wait Events section of the AWR report. Only the most common will be considered in this lesson.

Errors

- Parameter File Errors:
 - Parameter syntax
 - Parameter conflicts
- Exceeding instance or session limits:
 - Allocated resources
 - Memory allocation



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Some errors are catastrophic and terminate processes, or the instance. These must be handled immediately. Several errors included in this category are configuration errors that will prevent the instance from starting.

Parameter file errors that appear at startup are usually syntax errors in the parameter file (LRM prefix) or a conflict between related parameters. A conflict example is ORA-00849 “SGA_TARGET string cannot be set to more than MEMORY_MAX_TARGET string”. Another more subtle conflict is the dependency between STATISTICS_LEVEL and automatic memory management as illustrated by the error “ORA-00830: cannot set statistics_level to BASIC with auto-tune SGA enabled”.

Instance and session limit error appear in user sessions that exceed the limits, causing the session to terminate, or fail to start (for example, “ORA-00020: maximum number of processes (string) exceeded”). For this error, the PROCESSES parameter must be increased or the number of processes limited per instance. A failure to allocate memory will cause statement failures, session failures and even instance failures in extreme cases (for example, “ORA-04031: unable to allocate string bytes of shared memory...”).

Diagnosing Errors

1. Check the Errors manual.



2. Research My Oracle Support.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When confronted with an error, first check *Oracle Database Error Messages 12c Release 1*. The cause and action sections for the error may give sufficient information to solve the problem. Continue, if necessary, with searching the Knowledge Base at My Oracle Support <https://support.oracle.com>. Use the full text of the error message; often the parameters supplied in the error message are crucial to retrieving the correct information.

Common Errors and Actions

- Exceeds maximum number of sessions, processes, or transactions
- Snapshot too old
- Max # extents ... reached for <object type> <object name>
- Unable to allocate x bytes of shared memory ...
- PGA memory used by the instance exceeds PGA_AGGREGATE_LIMIT.

There are many other less common errors.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

This slide discusses the errors that occur while the instance is active.

Sessions will fail to start when either “ORA-00018 maximum number of sessions exceeded”, or “ORA-00020: maximum number of processes (string) exceeded” occur. Solutions: Increase the parameter in question, decrease the number of processes or sessions, or reconfigure the instance to use shared servers.

Transactions fail when “ORA-01574: maximum number of concurrent transactions exceeded” occurs. Solution: Change the TRANSACTIONS parameter and restart the instance.

Transactions fail and must be rolled back when the rollback records needed by a reader for consistent read are overwritten by other writers, with ORA-01555 snapshot too old. Solution: Use automatic undo, and the Undo Adviser to properly size the UNDO tablespace. See the topic “Undo Tablespace Best Practices” later in this lesson.

There are several object type errors that will cause a transaction failure when the object can not be extended. Solution: Use the error message to find the specific object and make the necessary adjustments to max extents, autoextend properties, and possibly to the tablespace.

The error, "ORA-04031: unable to allocate 56 bytes of shared memory ("streams pool", "unknown object", "streams pool", "fixed allocation callback")" is an example of a memory allocation error. Notice that the parameters give specific direction as to which memory area is affected. Solution: Increase the value of the parameter for this area, or use Automatic Memory Tuning. See previous lessons on tuning the various memory components.

When the PGA allocations exceed the `PGA_AGGREGATE_LIMIT`, the sessions using the most untunable memory will have their calls aborted. Parallel queries will be treated as a unit. First, the sessions that are using the most untunable memory will have their calls aborted. Then, if the total PGA memory usage is still over the limit, the sessions that are using the most untunable memory will be terminated. Solution: Increase the `PGA_AGGREGATE_LIMIT` initialization parameter or reduce memory usage.

There are many other less common errors that could potentially appear. These illustrate the methods of finding the cause and preventing future errors.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

Initialization Parameters with Performance Impact

Parameter	Description
COMPATIBLE	To take advantage of the latest improvements of a new release
DB_BLOCK_SIZE	8192 for OLTP and higher for DSS
MEMORY_TARGET	Automatically sized memory components
SGA_TARGET	Automatic SGA component management
PGA_AGGREGATE_TARGET	Automatic PGA management
PGA_AGGREGATE_LIMIT	Upper Limit on PGA allocated
PROCESSES	Maximum number of processes that can be started by the instance
SESSIONS	To be used with shared server



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A running Oracle instance is configured using initialization parameters, which are set in the initialization parameter file. These parameters influence the behavior of the running instance, including influencing performance. In general, a very simple initialization file with a few relevant settings covers most situations. The initialization file should not be the first place you expect to do performance tuning. However, improperly set initialization parameters can have drastic effects.

The table in the slide includes the most important parameters to set with performance implications:

- COMPATIBLE: Specifies the release with which the Oracle server must maintain compatibility. It lets you take advantage of the maintenance improvements of a new release immediately in your production systems without testing the new functionality in your environment. If your application is designed for a specific release, and you are actually installing a later release, then you might want to set this parameter to the version of the previous release. This parameter does not guarantee that the behavior of the new release will be identical to the release specified by COMPATIBLE. COMPATIBLE allows you to use a new release, while at the same time guaranteeing backward compatibility with an earlier release. This is helpful if it becomes necessary to revert to the earlier release. Reversion is not available between major releases if the COMPATIBLE parameter has been set to the new release. Some features of the release may be restricted, including patches, when the COMPATIBLE parameter has been set to an older release.

- DB_BLOCK_SIZE: Sets the size of the database blocks stored in the database files and cached in the SGA. The range of values depends on the operating system, but it is typically 8192 for transaction processing systems and higher values for data warehouse systems. 8192 is the default block size.
- MEMORY_TARGET: Specifies the total memory allocated to the instance, SGA and PGA. This parameter is incompatible with Linux huge pages.
- SGA_TARGET: Specifies the total size of all SGA components, and enables automatic tuning of certain memory pools.
- PGA_AGGREGATE_TARGET: Specifies the target aggregate PGA tunable memory available to all server processes attached to the instance; may be exceeded if untunable memory requirements exceed the target setting
- PGA_AGGREGATE_LIMIT: Specifies an upper limit to the amount of PGA that may be used. If the PGA_AGGREGATE_LIMIT value is exceeded, Oracle Database aborts or terminates the sessions or processes that are consuming the most untunable PGA memory. Parallel queries are treated as a single unit.
- PROCESSES: Sets the maximum number of processes that can be started by the instance. This is the most important primary parameter to set, because many other parameter values are derived from this.
- SESSIONS: This is derived by default from the value of PROCESSES. However, if you are using the shared server, the derived value is likely to be insufficient.

Initialization Parameters with Performance Impact

Parameter	Description
STATISTICS_LEVEL	Set to TYPICAL; required for reasonable Statspack and AWR reporting
UNDO_MANAGEMENT	AUTO mode recommended
UNDO_TABLESPACE	Undo tablespace to be used by instance
TEMP_UNDO_ENABLED	Use temporary tablespace for undo. Do not generate retained undo or redo.
LOG_BUFFER	Set a minimum size for redo log strands.
FAST_START_MTTR_TARGET	Set the number of seconds the database takes to perform crash recovery of a single instance.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

- STATISTICS_LEVEL can have a large impact on the performance of your database. Timed statistics are turned off when STATISTICS_LEVEL is set to BASIC, removing the effective information from the Statspack and AWR reports. STATISTICS_LEVEL set to TYPICAL has a small performance overhead. The entire AWR suite of tools has no more than a 5% performance overhead.
- UNDO_MANAGEMENT: the default and recommended value is AUTO. In manual mode rollback segments must be managed.
- UNDO_TABLESPACE specifies the undo tablespace to be used when an instance starts. You can replace an undo tablespace with another undo tablespace while the instance is running.
- TEMP_UNDO_ENABLED: If this is set to TRUE, the database can provide separate storage and retention model for temporary objects. This results in overall reduction in the size of undo log and redo log in the database.
- LOG_BUFFER: This specifies the amount of memory (in bytes) that Oracle uses when buffering redo entries to a redo log file. The log buffer size depends on the number of redo strands in the system. One redo strand is allocated for every 16 CPUs and has a minimum size of 2 MB. A minimum of 2 redo strands per instance are allocated. Any remaining memory in the redo granules is given to the log buffer.
- FAST_START_MTTR_TARGET: See the topic "Automatic Checkpoint Tuning" later in this lesson for more details.

Wait Events



Top 10 Foreground Events by Total Wait Time

Event	Waits	Total Wait Time (sec)	Wait Avg(ms)	% DB time	Wait Class
DB CPU		402.5		43.2	
log file sync	5,133	311.4	61	33.5	Commit
enq: HW - contention	95	36.9	389	4.0	Configuration
buffer busy waits	48,513	15	0	1.6	Concurrency
control file parallel write	364	11.6	32	1.2	System I/O
db file sequential read	83,300	10.9	0	1.2	User I/O
library cache: mutex X	1,846	6.2	3	.7	Concurrency
Disk file operations I/O	2,549	3.9	2	.4	User I/O
latch: cache buffers chains	7,586	3.2	0	.3	Concurrency
buffer deadlock	1,510	2.3	2	.3	Other

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Top 10 Foreground Wait Events is the first stop for collecting diagnostic information about the performance issues. Even well-performing databases will have a set of top wait events. It is a good practice to note on your database what these characteristic wait events are when it is performing acceptably. Frequently, the normal wait events are DB CPU, User I/O, and System I/O: datafile file reads and writes, log file sync, log file single write, and control file parallel write. The example shows a tuned simulated OLTP workload where performance is acceptable. These waits show a workload with a very high rate of commits, and a growing data set. The workload was designed to create some contention, so several of the waits are expected. In a production environment the high commit rate would indicate an opportunity to tune the application to reduce the commits, possibly by combining transactions.

Knowing the set of waits when the performance is acceptable, allows you to identify the unusual waits when the performance varies.

Diagnosing CPU waits

- DB Time
- DB CPU
- CPU per session

	Snap Id	Snap Time
Begin Snap:	75	04-Mar-14 14:30:59
End Snap:	76	04-Mar-14 14:40:09
Elapsed:		9.18 (mins)
DB Time:		27.66 (mins)

Top 10 Foreground Events by Total Wait Time						
Event	Waits	Total Wait Time (sec)	Wait Avg(ms)	% DB time	Wait Class	
log file sync	4,505	501.7	111	30.2	Commit	
DB CPU		340.2		20.5		
enq: HW - contention	364	327.9	901	19.8	Configuration	
buffer busy waits	56,547	68.2	1	4.1	Concurrency	

Host CPU									
CPUs	Cores	Sockets	Load Average Begin	Load Average End	%User	%System	%WIO	%Idle	
2	2	1	5.21	4.00	35.5	7.7	16.3	56.5	

Instance CPU		
%Total CPU	%Busy CPU	%DB time waiting for CPU (Resource Manager)
32.5	74.6	0.0

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

As mentioned in the lesson titled “Basic Tuning Diagnostics,” DB Time is the sum of the time sessions spend in user calls. DB Time includes actual CPU time, waits for CPU, I/O waits, or other waits. DB Time does not include waits by background processes. For example on a busy system, DB Time could be many times the elapsed time, because many sessions spend most of the time waiting.

The wait event DB CPU is almost always listed in the Top Foreground events, but does necessarily mean there is a problem. DB CPU time is the time the DB sessions are actually using the CPU. The important measurement is DB CPU compared to maximum amount of CPU Time available.

The maximum CPU time available is the number of CPUs times the elapsed time. Total DB CPU divided by the maximum CPU time should be close to the %Total in the Instance CPU section of the AWR report. Whether this is a problem to be tuned, depends on perception and CPU load.

If the perception is that the response time is acceptable, stop tuning. If the CPU load in the Host CPU Section of the AWR report shows that the %User + %System is close to 90% or greater the CPU is a bottleneck. This will also be reflected in the load average. When the load average is greater than two times the number of CPUs, the CPU is becoming a bottleneck.

CPU used by this session appears in the Other Instance Activity Stats section. This is the sum of all CPU used by foreground sessions in centiseconds, and should be very close to DB CPU time.

Tuning CPU Waits



CPU Intensive Tasks:

- Parsing – Cursor Sharing
- SQL Execution – SQL by CPU Time
- Others – Missing CPU time

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When DB CPU wait indicates a problem, examine known CPU-intensive tasks.

First check for parsing and hard parsing. The ADDM findings may show Hard Parse. The Load profile may show significant Parses (SQL) and Hard parses (SQL), Instance Efficiency Percentages section will show a low (< 80%) Execute to Parse% and a low % Non-Parse CPU. Latch: shared pool in Foreground Waits is often high as well, because parsing requires allocating cursor areas. When the parse time is high, generally the SQL statements are not using shared cursors so the CPU time is spread across hundreds or thousands of SQL statements making the SQL Statistics Sections unreliable. Parsing may also be caused by insufficient shared pool size causing reloads or invalidations. For solutions, see the lesson titled “Tuning the Shared Pool” for details on cursor sharing and sizing the shared pool.

SQL statement execution is by definition a major task for the CPU on any database system. If this is the problem, the solution is to find and tune the SQL statements that will yield the most benefit. The statements listed in SQL ordered by CPU Time and SQL ordered by Gets sections should be the first SQL statements examined and tuned. These sections include statements that are resource hogs where each execution uses large amounts of CPU, and SQL statements that are executed very frequently where a small boost in efficiency will be multiplied by the number of executions.

Other possible issues can include areas where there are holes in the database instrumentation. This is indicated when DB CPU shows a much lower time than expected based on the Busy% in the Instance CPU, or Host CPU sections. In this case further examination is required to determine the source of the waits.. Other wait events that are in the top foreground wait list are candidates. Research in My Oracle Support or create a performance service request with Oracle Support to determine likely candidates and solutions.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

Redo Path Wait Events



- log buffer space
- log file synch
- log file switch <various>
- log archive I/O
- log file read and write

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The redo path starts with a transaction that changes something in the database. This change creates undo and redo records that are written to the SGA (buffer cache and log buffers, respectively) and data changes that are written to the buffer cache. As time progresses, DBWR writes the oldest buffers to database files, and LGWR writes the oldest log records to the redo log files. When a Commit occurs, LGWR notifies DBWR that all the buffers associated with the transaction plus any buffers that are covered by the redo records older than the commit, must be written to datafiles. When DBWR completes this operation, LGWR writes all the redo records older than the commit to redo log files. The rule is that DBWR must not write any data buffers unless the associated redo records have been written to a log file. This ensures that the data is always recoverable. When a log file fills, a log file switch closes one log file and starts filling the next. When archiving is enabled, the ARCh process copies the redo log to one or more archivelogs. The log file is not available to be overwritten until the archiver has made a copy. When archiving is not enabled, a log file is not available until all the changed data blocks covered by that redo log have been written to disk (checkpoint completed).

The wait events that indicate waits in the redo path:

- **log file space** – The log buffer is full waiting on LGWR to make space by writing some of the buffer to log files, primarily because log buffer is too small.

- **log file synch** – The log buffer is full waiting on LGWR to make space by writing some of the buffer to log files, primarily because log files are filling too rapidly. Increase the size and number of log file groups. Consider moving the log files to faster disks such as striped disks.
 - **log file switch (archiving needed)** – LGWR is writing redo faster than ARCh is copying redo log files to archive log files. Check that archiving has not stopped, then check the speed of the archive log I/O, and possibly add ARCh processes. Create more and larger redo log groups, if this is an intermittent problem due to brief high volume operations.
 - **log file switch (checkpoint incomplete)** – Waiting for a log switch because the session cannot wrap into the next log. Wrapping cannot be performed because the checkpoint for that log has not completed. Consider that DBWR may be lagging the LGWR and increase the speed of DBWR with additional DBWR processes. Increase the size and number of log file groups to allow DBWR more time to complete checkpoints without causing waits.
 - **log file switch (private strand flush incomplete)** – User sessions trying to generate redo wait on this event when LGWR waits for DBWR to complete flushing redo from in memory undo (IMU) buffers into the log buffer. When DBWR is complete, LGWR can then finish writing the current log, and then switch log files. This is considered normal if the time between this message and the “Thread ... advanced” message is not too long (a few seconds). The in memory undo feature introduced in Oracle Database 10g, moves undo segment management into a memory structure, reducing the amount of redo generated. This information is kept in a private strand and must be dumped to redo logs on commit, and it is done in bulk. If this is a problem, increase the size and number of redo log files.
 - **log file switch *** – For all of the log file switch wait events. **Diagnosis:**
 1. Look for log file parallel writes, check the average write time, and fix I/O speed if needed and if possible.
 2. Increase the size of the log file to reduce the number of switches.
 3. If there are alert log messages “log file switch not complete”, check the time between log switches. If it is less than 15 minutes, increase the size and number of log files. The larger size increases the time between log switches, the number of log files gives archiver more time to copy the log file before the log file is overwritten. This solution is indicated when concurrent with log file switch (archiving needed) or log file switch (checkpoint incomplete)
 - **log archive I/O** – Local archiving of online redo logs (for a production database) or standby redo logs (for a standby database) is being used. When the archiving process exhausts its I/O buffers because all of them are being used for on-going I/O's, the wait for an available I/O buffer is captured in this system wait event. **Diagnosis:** Determine why buffers are not being cleared; possibilities include slow I/O and problem with the disk I/O
 - **log file read and write** – Includes log file parallel write, and log file single write, and log file sequential read. These events are part of normal database operation it is when the wait times are excessive ~ 10 msec average the investigation is called for.
- Note:** A well-tuned DB with NO performance problem will likely show these some of these events in the top 10 wait events.

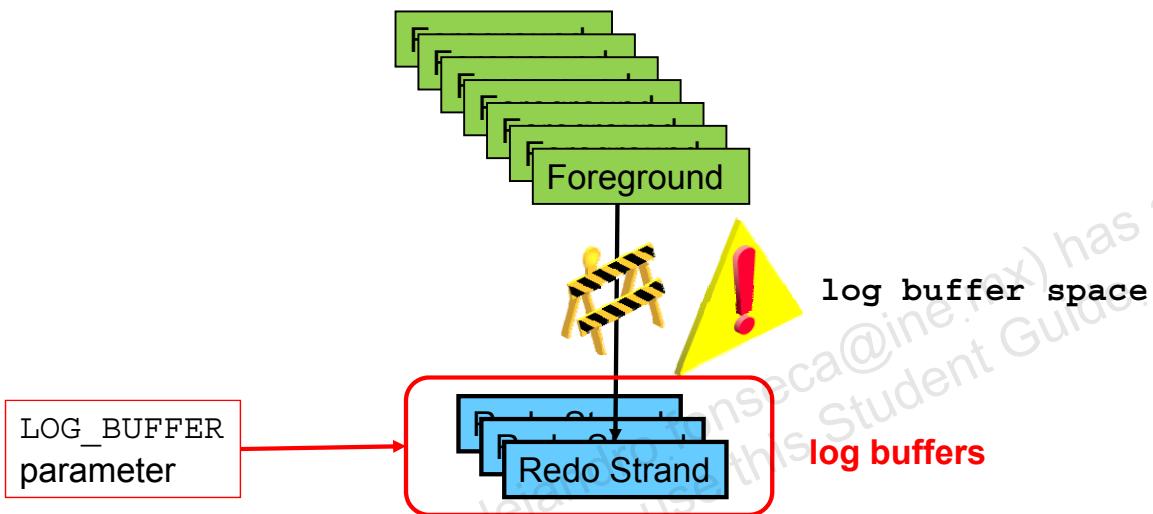
Diagnosis

- Compare the database I/O statistics with the system I/O Statistics, check for large differences.
- Check I/O for particular files, check for hot volumes, and hot files.
- Check that disk devices are of comparable speed in disk arrays.
- Reduce redo generation if possible with nologging for bulk loads.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

Redo Generation

- Redo buffer is divided into multiple strands.
- Foreground processes write to redo strands.
- Strands are activated and deactivated based on load.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In the redo generation phase, since Oracle Database 9*i* Release 2, foreground (server) processes write redo records to redo strands. The redo buffer is divided into multiple redo strands at startup based on the number of CPUs in the server. The number of strands allocated is one per 16 CPUs with a minimum of two strands.

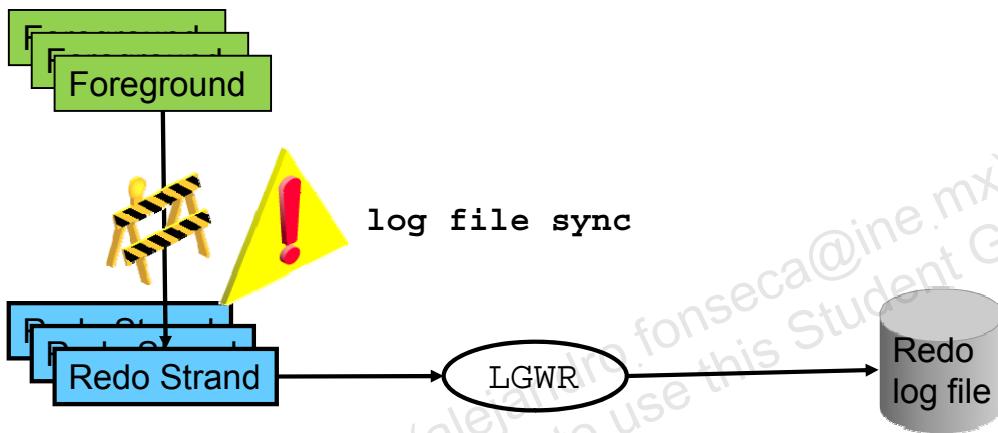
The number of strands that are active at any time is dynamic. Each foreground process will write to one of the active redo strands. Strands will be activated when more strands are required to prevent the foregrounds from waiting. The strands are automatically deactivated to reduce the amount of work that is needed to collect the redo records across all the redo strands.

Having multiple strands allows each foreground process to write redo independently. Many foreground processes may use a few active redo strands. As the load increases, more redo strands will be activated.

The space in the redo strands are used in a circular manner. When the foreground process must wait for space in a redo strand to become available, a log buffer space wait event is logged. The strands can become full if the LGWR process and helpers are not writing to the redo log file quickly enough, or if not enough space is allocated to the redo strand. This space is specified with the LOG_BUFFER initialization parameter.

Redo Write

Foreground processes wait on LGWR to complete a write on Commit and special circumstances.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When a foreground process issues a redo write request associated with a COMMIT, it has to wait until LGWR and helpers complete the request. If all LGWR and helper processes are busy with other requests, then all foreground processes waiting on a commit would wait. In this case, foreground processes wait on a log file sync event, which includes LGWR waiting on I/O, and posting to the foreground. Log file sync and log buffer space are seldom seen together, but both point to a situation where LGWR and helper processes are not writing fast enough.

Oracle Database 12c Redo Write

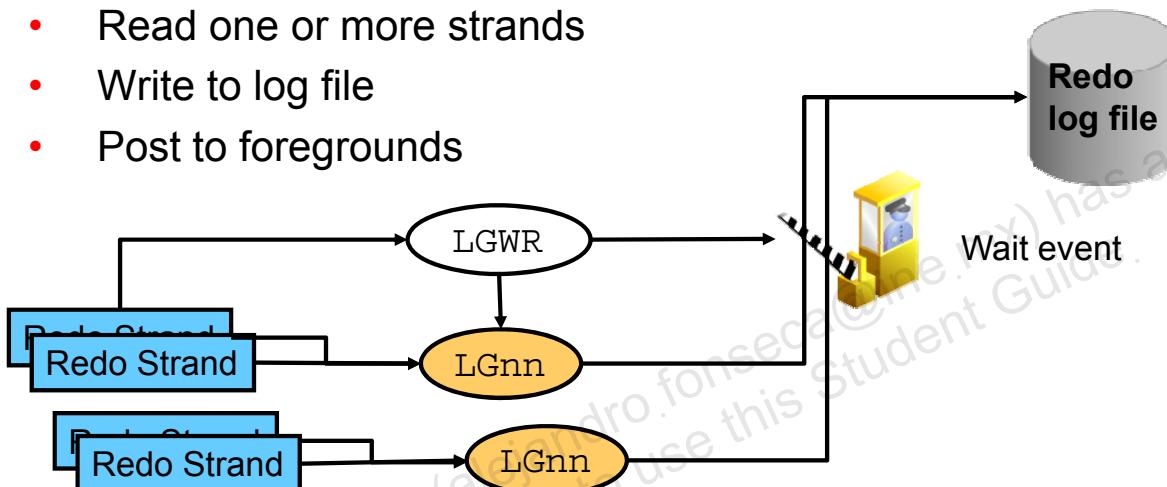
LGWR:

No Configuration Needed

- Is a coordinator of LGnn “helper” processes
- Ensures *correct* order of work for operations that must be ordered

Log Writer Groups (LGnn) work concurrently:

- Read one or more strands
- Write to log file
- Post to foregrounds



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In Oracle Database 12c, LGWR starts a helper process named LGnn. LGWR handles the operations that are either very fast, or must be coordinated. It then delegates to the LGnn operations that could benefit from concurrent operations, primarily writing the redo from the log buffer to the redo log file, and posting the completed write to the foreground process that is waiting.

Because LGnn processes work concurrently and certain operations must be performed in order, LGWR forces an ordering, so that even if the writes complete out of order, the posting of the foregrounds will be in the correct order. If the current log file is not available for writing, a wait event describing the reason is activated.

Automatic Checkpoint Tuning

- Is the best-effort checkpointing, without much overhead
- Reduces average recovery time by making use of unused bandwidth
- Is enabled when `FAST_START_MTTR_TARGET` is not explicitly set to zero



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Waits on the redo chain and the buffer cache can be affected by aggressive checkpoint parameter settings.

Automatic checkpoint tuning greatly reduces the need to set any of the checkpoint parameters. These parameters can still be used to set an upper bound on recovery time. In earlier versions, if you had large log files and large buffer caches so that very few buffers were written for aging purposes, you would set parameters such as `FAST_START_MTTR_TARGET` to reduce crash recovery times.

By default, Oracle Database supports automatic checkpoint tuning by making the best effort to write out dirty buffers without an adverse impact on the throughput. Frequent checkpoints will enable faster recovery, but can cause performance degradation. A checkpoint might be a costly operation when the number of files is huge since it has to freeze the datafile headers during the process. There is a performance trade-off regarding frequency of checkpoints.

More frequent checkpoints enable faster database recovery after a crash. This is why some customer sites, which have a very low tolerance for unscheduled system down time, will often choose this option. However, the performance degradation of frequent checkpoints may not justify this philosophy in many cases.

Assume that the database is up and running 95% of the time, and unavailable 5% of the time from frequent instance crashes or hardware failures requiring database recovery. For most customer sites, it makes more sense to tune for the 95% case rather than the rare 5% down time.

A checkpoint occurs at every log switch. Therefore, frequent log switches will start the checkpoints and may degrade performance. If a previous checkpoint is already in progress, the checkpoint forced by the log switch will override the current checkpoint. This necessitates well-sized redo logs to avoid unnecessary checkpoints as a result of frequent log switches. A good rule of thumb is to switch logs at most every twenty minutes. Having your log files too small can increase checkpoint activity and reduce performance. Oracle recommends the user to set all online log files to be the same size, and have at least two log groups per thread.

Set the value of `FAST_START_MTTR_TARGET` to 3600. This enables Fast-Start checkpointing and the Fast-Start Fault Recovery feature, but minimizes its effect on run-time performance while avoiding the need for performance tuning of `FAST_START_MTTR_TARGET`.

Sizing the Redo Log Buffer

- The size of the redo log buffer is determined by the:
 - `LOG_BUFFER` parameter
 - Remaining space in the fixed area granule
- Default value: Can range from 5 MB to 32 MB
- The log buffer is written at:
 - Commit
 - One third full
 - DBWR request



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

If transactions are long or numerous, then larger values of `LOG_BUFFER` reduce the frequency of log file I/O. The smaller the buffer is, the more often it will get to be one-third full before a `COMMIT` operation clears the buffer.

The minimum value of `LOG_BUFFER` is 2 MB. But the actual allocation will include the remainder of the granule allocated to the fixed area of the SGA. The log buffer size may be different than the `LOG_BUFFER` parameter value. The `V$SGASTAT` view displays the actual log buffer size.

Example:

```
SELECT 'v$parameter' "View name", name,
       to_number (value,'999999999') "Value"
  FROM v$parameter WHERE name = 'log_buffer'
UNION
SELECT 'v$sgastat' "View name", name, bytes
  FROM v$sgastat WHERE name = 'log_buffer';
```

View name	NAME	Value
v\$parameter	log_buffer	5201920
v\$sgastat	log_buffer	5472256

Sizing Redo Log Files

- The size of redo log files can influence performance.
- Larger redo log files provide better performance.
- Generally, redo log files should range between 100 MB and a few gigabytes.
- Switch redo log files at most once every twenty minutes.
- Use the Redo Log file Size Advisor to correctly size your redo logs.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The size of the redo log files can influence performance, because the behavior of the database writer and archiver processes depend on the redo log sizes. Generally, larger redo log files provide better performance. Undersized log files increase checkpoint activity and reduce performance.

If the `FAST_START_MTTR_TARGET` parameter is set to limit the instance recovery time, the system automatically tries to checkpoint as frequently as necessary. Under this condition, the size of the log files should be large enough to avoid additional checkpointing. The optimal size can be obtained by querying the `OPTIMAL_LOGFILE_SIZE` column from the `V$INSTANCE_RECOVERY` view. You can also obtain sizing advice on the Redo Log Groups page of Oracle Enterprise Manager Cloud Control.

It is not possible to provide a specific size recommendation for redo log files, but redo log files in the range of a hundred megabytes to a few gigabytes are considered reasonable. Size your online redo log files according to the amount of redo your system generates. A rough guide is to switch logs at most once every twenty minutes.

Increasing the Performance of Archiving

- Share the archiving work during a temporary increase in workload:

```
ALTER SYSTEM ARCHIVE LOG ALL  
TO <log_archive_dest>
```

- Increase the number of archiver processes with `LOG_ARCHIVE_MAX_PROCESSES`.
- Multiplex the redo log files, and add more members.
- Change the number of archive destinations:
 - `LOG_ARCHIVE_DEST_n`



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Share the Archiving Workload Temporarily

If you anticipate a heavy workload for archiving, you can add another process to share the work by executing the `ALTER SYSTEM ARCHIVE LOG ALL TO 'directory_name'` command. This command causes the server process to behave as an archiver until all redo log files have been archived. These processes are not affected by `LOG_ARCHIVE_MAX_PROCESSES`.

Increase the Number of Archiver Processes

Occasionally, on busy databases, a single `ARCn` process cannot keep up with the volume of information written to the redo logs. You can define multiple archiver processes by using the `LOG_ARCHIVE_MAX_PROCESSES` parameter. This parameter sets the maximum number of `ARCn` processes that can be started.

The `LGWR` process starts a new `ARCn` process whenever the current number of `ARCn` processes is insufficient to handle the workload.

Multiplex Redo Log Files

If you are archiving, it is even more important to have more than two redo log groups with multiple members. When logging switches to another group, the DBW n process must perform checkpointing as usual, and one file must be archived. You must allow time for both of these operations before the LGWR process needs to overwrite the file again by having sufficient groups. Take advantage of the ARC n behavior of reading a block from each member of the redo log group to spread the I/O across multiple disks by adding more members to the redo log groups.

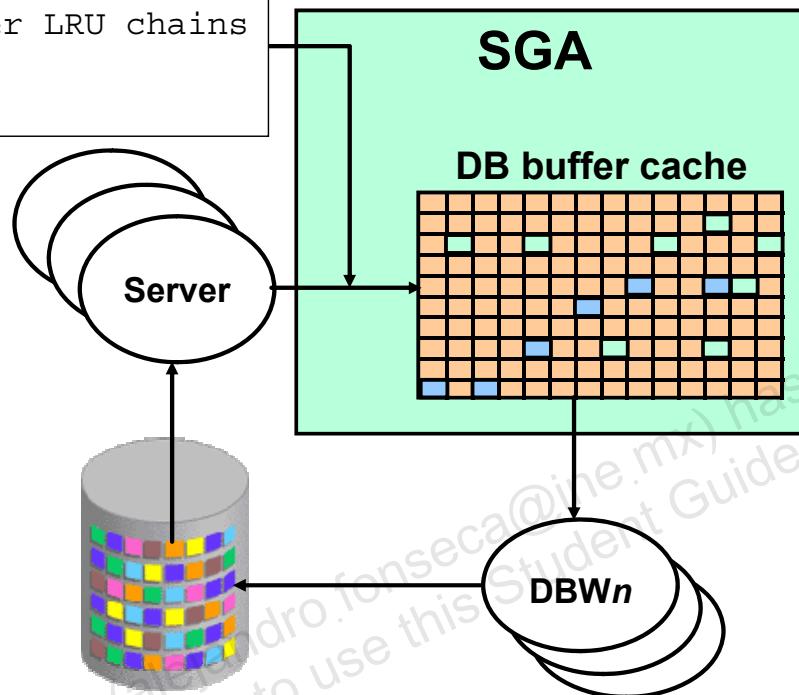
Decrease the Number of Archive Destinations

This is not often a possible choice. More archive destinations increase the overhead of archiving. An alternative to several archive destinations could be to use a fetch archive log server to make additional archive log file copies on another machine.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

Buffer Cache Waits

```
latch: cache buffers chains  
latch:cache buffer LRU chains  
free buffer waits  
buffer busy waits
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The server (foreground) processes will see various waits when accessing the database buffer cache.

The **cache buffer LRU chains latch** is acquired in order to introduce a new block into the buffer cache and when writing a buffer back to disk, specifically when trying to scan the LRU (least recently used) chain containing all the dirty blocks in the buffer cache. The solution is to reduce the number of new blocks being introduced to the buffer cache. Look for SQL statements that have a high number of physical reads. Review and tune to reduce the number of gets per execution.

The **cache buffer chains latch** is acquired whenever a block in the buffer cache is accessed (pinned). Reducing contention for the cache buffer chains latch will usually require reducing logical I/O rates by tuning and minimizing the I/O requirements of the SQL involved. High I/O rates could be a sign of a hot block. Examine the statements that perform the most buffer gets and then look at their access paths to determine whether these are performing as efficiently as you would like. Checking the SQL Ordered by Gets, and the Segments by Logical Reads can narrow your search for the SQL statements and segments that are involved.

A **free buffer wait** occurs, when a server process reads a block into the buffer cache, and searches the LRU list for a free buffer. After looking at a set number of buffers on the cold end of the LRU and not finding a free buffer, the server process posts the DBWR to make free buffers by writing dirty buffer to disk. The server process enters a **free buffer wait**. The solution may be the DB buffer cache is not large enough, or that DBWR is writing fast enough. See the lesson titled “Tuning the Buffer Cache” for more details.

Buffer busy waits are covered in the following slide.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

Buffer Busy Waits

Diagnosing:

- Determine the block type.
- Check the number of waits and total time waited.

Buffer Wait Statistics

- ordered by wait time desc, waits desc

Class	Waits	Total Wait Time (s)	Avg Time (ms)
data block	20,442	8	0
undo block	34,361	4	0
file header block	8	1	149
undo header	3,187	1	0
1st level bmb	1	0	0

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Buffer busy wait can occur for a variety of reasons. In general, a buffer busy wait occurs when one session requests to access a buffer that another session is using. This access can be either read or write.

The actions needed to reduce the block contention vary based on the type of block being accessed. Prioritize the action by the total wait time. The priority for the example shown is to reduce the contention for data blocks.

The actions needed for each type of block:

Data block – This is very common. The next step is to determine which segments are experiencing the contention, then find the SQL statements that are causing the contention.

Eliminate hot blocks by spreading the workload over more blocks.

- Table – Is the contention due to reads or DML?
 - In either case, reduce the time that the session holds the block.
 - For reads, change the SQL to access fewer blocks, and cache the segment in the keep cache. If the Buffer Hit % is low and there are a high number of physical reads for the SQL accessing this segment, then consider increasing the buffer cache size.

- For writes (DML), change the application to use fewer sessions. Spread the inserts across multiple blocks with Automatic Segment Space Management (ASSM) instead of manual space management. For large tables with high insert rates, consider hash partitioning.
- Indexes – Is the contention due to reads or writes?
 - Reads – Change the SQL to use fewer indexes, or more selective indexes. Use bitmapped indexes (more rows in fewer blocks could increase contention). Rebuild or coalesce indexes to reduce the number of levels (reduces the number of branch blocks that must be accessed)
 - Writes – Reduce the number of indexes (use SQL Access Advisor). If indexed columns are always increasing, the new column values are always being added to the right, and in the same block. Consider reverse key index or a scalable index key to spread the insert activity across multiple blocks.

Undo block – Undo is used to protect transactions. This will primarily be seen in OLTP applications with high rates of DML. Use Automatic Undo, then SMON determines the number of undo segments to bring online, based on `maxconcurrency` in `V$UNDOSTAT` in the past week.

Undo header - Use automatic undo management or add more rollback segments. The waits can be amplified greatly when physical reads are slow due to poor I/O subsystem performance.

File Header Block – Seldom a problem, but when there are many sessions attempting to find extents, then the extent map in the file header can show buffer busy waits. Solution is to add more datafiles. Note that only one file is possible with bigfile tablespaces.

Data Header block – The best solution is to use ASSM since it is sometimes tricky to arrive at a correct freelist or freelist group setting. Freelist solutions are seldom used except in RAC environments.

Shared Pool Waits

- latch: shared pool
- latch: row cache objects
- Mutex Waits
 - cursor: mutex S
 - cursor: mutex X
 - cursor: pin S
 - cursor: pin S wait on X
 - cursor: pin X
 - library cache: mutex S
 - library cache: mutex X



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The `shared pool latch` is used to protect critical operations when allocating and freeing memory in the shared pool. The library cache latches protect operations within the library cache itself. All of these latches are potential points of contention. The number of latch gets occurring is influenced directly by the amount activity in the shared pool, especially parse operations. Anything that can minimize the number of latch gets and indeed the amount of activity in the shared pool is helpful to both performance and scalability.

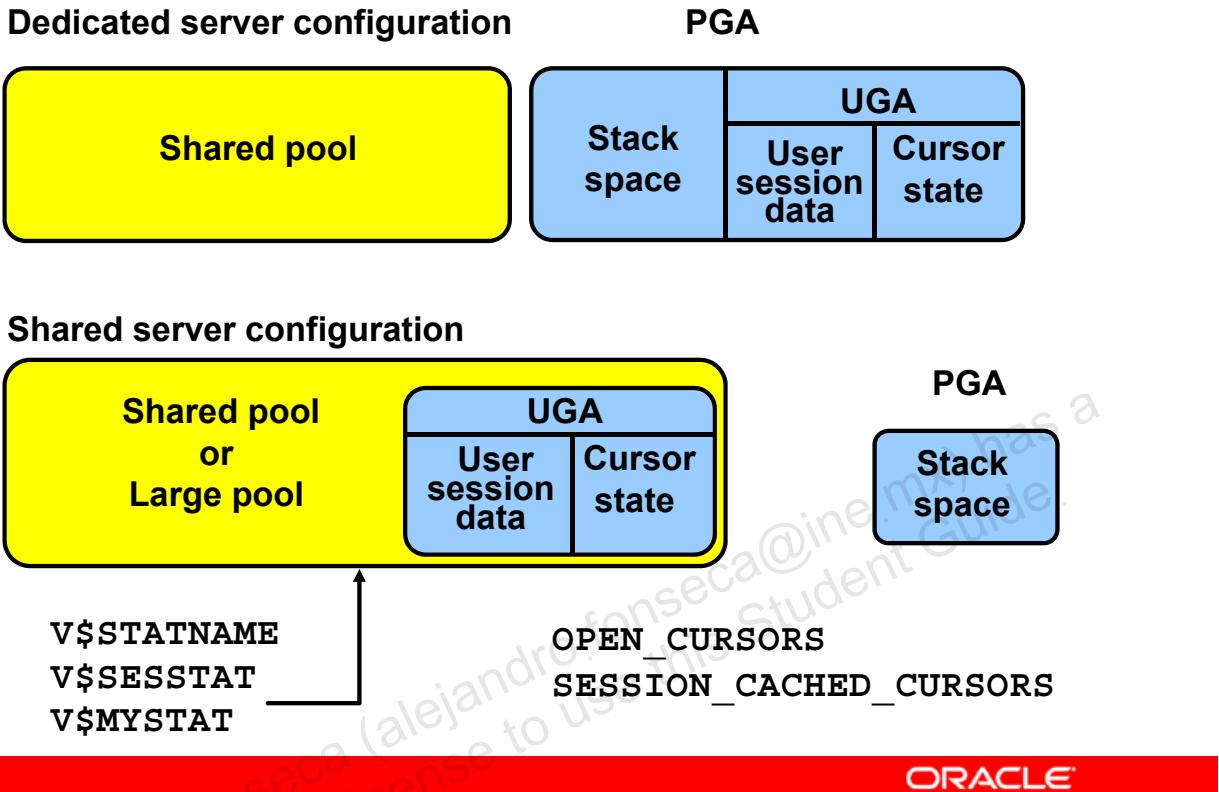
Waits for cursors are parse time waits and involve the loading of cursors into the shared pool or searching for those cursors. Most problems occur because of:

- Excessive number of versions of cursors in the shared pool. Check the SQL ordered by Version Count to verify. Version counts greater than 20 may need cursor sharing techniques applied.
- Excessive hard/soft parsing
- Excessive Invalidations/reloads
- Enormous objects are loaded
- Inappropriately sized shared pool

Mutex Waits – See MOS document “FAQ: 'cursor: mutex ..' / 'cursor: pin ..' / 'library cache: mutex ..' Type Wait Events (Doc ID 1356828.1)” for detailed diagnostic directions.

See lesson “Tuning the Shared Pool” for details

UGA and Oracle Shared Server



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Shared pool waits can occur due to inappropriate use of the shared pool by shared servers.

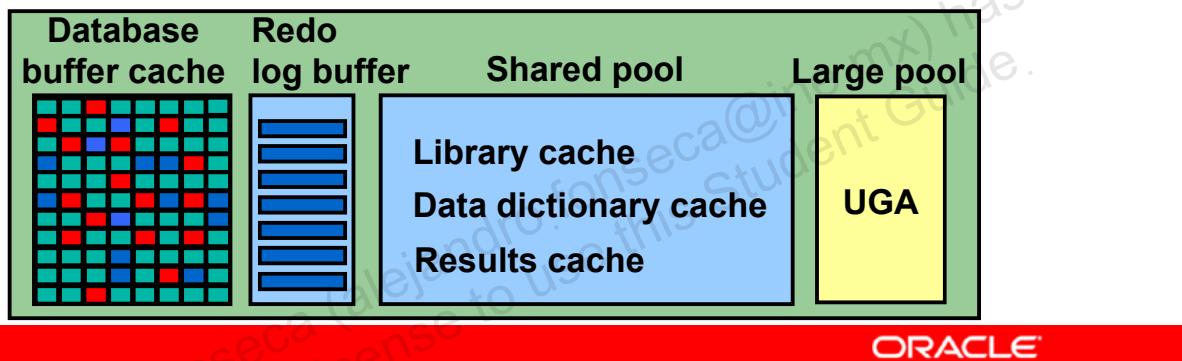
In a dedicated server configuration, the user global area (UGA) resides in PGA and does not use any memory within the SGA. If you use Oracle Shared Server, the UGA (which includes the user session and cursor state information) is stored in the SGA instead of in a private user memory. Sort areas and private SQL areas are included in the session information. This is because shared servers work on a per-call basis, so any server may need access to any user session information.

The memory requirement for the UGA with Oracle Shared Server is no larger than if you use dedicated servers, but it is allocated from the SGA instead of process memory. You may need to increase the SGA size, but your private user memory is lower. The main advantage of shared servers is that the number of servers and the required memory is reduced.

In a shared server environment, configuring the large pool is recommended. If a large pool has not been configured, the UGA is stored in the shared pool. If you configure the large pool, the UGA for shared servers is allocated from the large pool. By allocating session memory from the large pool, the shared pool is used primarily for caching shared SQL and avoids additional contention due to the reduced space because of space used by the UGA. Allocate approximately 250 KB for each concurrent session as a starting point.

Large Pool

- Can be configured as a separate memory area in the SGA, used for memory with:
 - I/O server processes: DBWR_IO_SLAVES
 - Backup and restore operations
 - Session memory for the shared servers
 - Parallel execution messaging
- Is used to avoid performance overhead caused by shrinking the shared SQL cache
- Is sized by the LARGE_POOL_SIZE parameter



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

There are no common wait events that directly reference the large pool. Although ORA-4031 Cannot allocate <x Bytes> of memory from <component>... can occur, configuring the large pool, will reduce pressure on the shared pool, and shared pool-related waits and errors. Oracle Support recommends that you configure the large pool any time you configure shared servers, parallel query server, DBWR_IO_SLAVES, or ASM. The large pool participates in Automatic Shared Memory Management (ASMM). ASMM is the recommended method for managing the memory requirements of the main portions of the SGA.

Existence of the Large Pool

The memory of the large pool is part of the SGA, and adds to the amount of shared memory the Oracle server needs for an instance at startup.

Advantages of the Large Pool

The large pool is better able to satisfy requests than the shared pool for large allocations (a few hundred kilobytes) of session memory for these additional requests:

- I/O server processes
- Backup and restore operations (RMAN Slave Buffers)
- Block change tracking buffers

Tuning the Large Pool

- The large pool has one parameter: `LARGE_POOL_SIZE`.
- `V$SGASTAT` shows used and free memory.

```
SELECT * FROM v$sgastat
WHERE pool = 'large pool';
```

POOL	NAME	BYTES	CON_ID
-----	-----	-----	-----
large pool	free memory	2814112	0
large pool	session heap	1380192	0

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

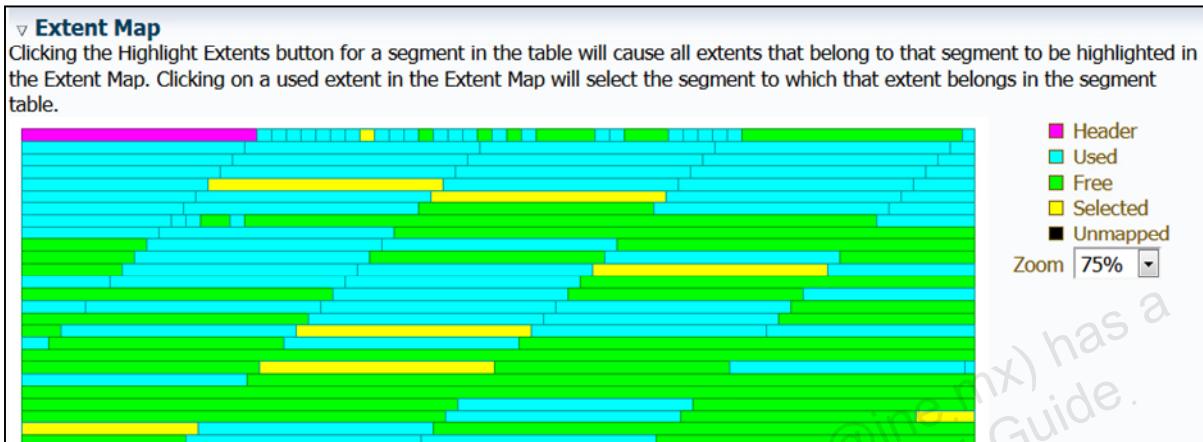
The large pool is not governed by a least recently used algorithm. Memory allocations are released by the processes that allocated them or by PMON as part of a cleanup operation.

The only tuning that is available is to monitor the free memory and adjust the `LARGE_POOL_SIZE` parameter as needed.

The large pool area shown as `session heap` in the slide is the User Global Area used by Oracle Shared Servers.

Space Management Waits

enq: HW contention



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Every time a segment is extended, freespace in the correct tablespace must be found, allocated to the segment and marked as being used. The high-water mark for the segment and the tablespace must be adjusted. If there is insufficient space in the tablespace, and the datafiles are specified to autoextend, the datafile must be extended, the freespace updated, and then the segment can be extended. If there is not enough space, and the datafile is not set to autoextend, an error is raised. Included in this process is a check that the segment owner has sufficient quota on the tablespace. The HW enqueue is held until the segment extension is completed.

enq: HW contention – The High Water (HW) enqueue is used to control the boundary between the used and unused space in a segment. The time that the enqueue is held is longer if a new extent must be allocated from free space in the tablespace, and longer still if the datafile must be extended to provide free space.

The UNDO tablespace can grow like any other tablespace, but undo segments have some specialized activity. The segments are brought online and taken off line dynamically. Extents can be “stolen” from one segment and given to another. The extent map shown in the slide is for an undo tablespace.

Temporary tablespaces also have specialized actions; temporary segments are allocated and deallocated.

General Tablespace: Best Practices

- Use locally managed tablespaces with auto-allocate extents policy.
- Use Automatic Segment Space Management (ASSM).
- Use online segment shrink to eliminate internal fragmentation.
- Periodically review the results of the Automatic Segment Advisor.
- Monitor tablespace space usage using server-generated alerts.
- Size of extents matters more than the number of extents in the segments.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Use locally managed tablespaces with auto-allocate to avoid fragmentation of tablespace.

Note: The largest auto-allocate extent size possible is 64 MB. This should not impact segments that are less than a terabyte in size.

Use ASSM for better segment space management performance.

The Automatic Segment Advisor examines segments residing in tablespaces with outstanding tablespace-full alerts and those with heavy user access for signs of fragmentation. If the segments are fragmented, a recommendation is made to rectify.

Use server-generated alerts to monitor tablespaces for impending out-of-space conditions.

Use properly sized extents, when assigning the extent value manually. If the extents are too small, then multiblock reads are prevented. If the extents are too large, space is wasted in the last extent. There are only a few cases where the number of extents should be a cause for concern:

- Tablespace is dictionary managed and DDL commands such TRUNCATE and DROP are common.
- Parallel queries are common and extents are noncontiguous in the file.

Undo Tablespace: Best Practices

- Use Automatic Undo Management.
- UNDO_RETENTION:
 - Is automatically adjusted when the UNDO tablespace has AUTOEXTEND enabled
- Undo tablespace size:
 - Initial size: Small with AUTOEXTEND enabled
 - Steady state: Fix size using the Undo Advisor and add a 20% safety margin.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Automatic Undo Management (AUM) is the preferred option because it greatly simplifies undo management and also provides an easy way to avoid errors, such as “ORA-1555 : Snapshot too old...”.

UNDO_RETENTION: This parameter sets the minimum time that undo will be retained. The main purpose of this parameter is to support flashback requirements because the read-consistency aspect of undo retention no longer needs manual configuration. Undo retention is automatically tuned to the maximum allowable by the tablespace when the tablespace is a fixed size, even if it is greater than the longest-running query in the system. This gives the best possible retention and in effect makes the setting of the UNDO_RETENTION parameter unnecessary. Therefore, the only consideration in undo management is now the proper sizing of the undo tablespace.

The most accurate and recommended way of sizing the undo tablespace is by using the Undo Advisor. However, a newly created database does not have the kind of historical workload information that is needed by the advisor to make sound recommendations. For such new systems, the best way to size the undo tablespace is to start with a small size, such as 100 MB, but to set the AUTOEXTEND data file attribute to ON. This allows the tablespace to grow automatically as needed. After the database has reached a steady state, it is recommended to adjust the size of the tablespace. To determine the appropriate size of the undo tablespace, you should use the Undo Advisor. Add 20 percent to the size provided by the Undo Advisor for safety purposes and disable the AUTOEXTEND attribute.

SQL Execution Related Waits

Transaction waits

- enq: TX - < reason >

I/O waits

- db file sequential read
- db file scattered read
- db file parallel read



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Every SQL must be parsed and executed. The parsing related waits are considered in the “Shared Pool Waits” topic.

Every SQL statement does some work:

- SELECT statements require that database blocks are read, and row information extracted, formatted, and sent to the user.
- DML statements read the blocks, lock the rows being changed (both table and indexes), write redo and undo, change the rows and indexes, and then report completion to the user.
- Multiple wait events are possible along this path: I/O related for reads and writes, memory management to allocate buffers for blocks from the datafiles, and contention when multiple sessions attempt to access the same block or change the same row.

There are several **enq: TX** waits that can be listed. Some of the more common ones are:

- **Row lock contention** – Cause: One session has changed a row, and another session is waiting to change the same row. The first session must COMMIT or ROLLBACK to allow the second session to continue.

Solution: Keep application transactions small and quick.

- **Index contention – Causes:**

1. A session is updating a bitmapped index segment, and another session is attempting to update a row covered by the same bitmap segment. (Recall that a bitmap segment covers a large number of rows.)

Solution: Commit frequently, use short transactions, re-evaluate the need to use bitmapped indexes.

2. Index block splits while inserting a new row into the index. The transactions will have to wait for TX lock, until the session that is doing the block splits completes the operations. A session will initiate an index block split, when it cannot find space in an index block where it needs to insert a new row. Before starting the split, it would clean out all the keys in the block to check whether there is enough sufficient space in the block. Splitter has to do the following activities: Allocate a new block, copy a percentage of rows to the new buffer, add the new buffer to the index structure, and commit the operation. Waits for block splits are caused by:

- Indexes on the tables, which are being accessed heavily from the application with DML
- Indexes on table columns, which have monotonically increasing values inserted.

Solutions:

- Find the hot indexes from “Segments by Row Lock Waits.”
- Rebuild the indexes listed in the 'Segments by Row Lock Waits' of the AWR reports as reverse key indexes or hash partition the indexes.

I/O Waits for SQL

Diagnosis:

- Average wait time
- Number of waits

Solutions:

- Tune SQL to reduce reads
- Tune the Buffer Cache
- Tune the schema object
- Tune the I/O subsystem



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When a SQL statement needs to access a database block, the foreground session (server) reads the block into memory. If the average wait time is not excessive, that is less than 20 msec, the issue is the number of waits and not the underlying I/O subsystem.

Waits for the event '`db file sequential read`' indicate that a session is waiting for single blocks to be delivered from disk to satisfy a query. The solution for a high number of waits is tuning the SQL statements that issue these logical reads. A SQL statement that appears in both SQL Ordered by Gets and SQL Ordered by Physical Reads is a candidate for tuning to reduce this wait event.

When the segment being accessed is an index, the index may be fragmented due to a significant number of deletes. Consider rebuilding or coalescing the index. Reducing the size of the index can reduce the number of blocks accessed.

The solutions should be used in order. Tune the SQL first, then the Buffer cache, and finally the I/O subsystem. Tuning the SQL can result in system-wide improvements, possibly orders of magnitude, whereas tuning the buffer cache or I/O will yield percentage improvements. I/O tuning includes checking for excessive I/O to specific disks. This can cause disk contention and uneven performance.

Note: Tuning the I/O will help only if the average wait time is excessive.

Waits for the event '**db file scattered read**' indicate that a session is waiting for a multiblock I/O to complete. This typically occurs during full table scans or index fast full scans. Solutions: Reduce the number of full table scans by finding candidate SQL in the SQL Ordered by Physical Reads and tuning. Because a full table scan reads all the blocks below the HWM, tables that have significant (> 10%) empty blocks, or high PCTFREE increase the number of I/Os required. Reduce fragmentation in the tables by using SHRINK to remove empty and partially filled blocks, or set the PCTFREE to a lower value and move the table. Increasing the size of the buffer cache does not help because full table scans do not use the buffer cache, but CACHING frequently accessed smaller segments in the KEEP POOL, and verifying that the OS I/O buffer cache is being used by the disk where the segments reside.

Waits for event '**db file parallel read**' indicate that a session is waiting for parallel reads from multiple datafiles to non-contiguous buffers in memory (PGA or buffer cache). This is done during recovery operations or when buffer prefetching is being used as an optimization, that is, instead of performing multiple single-block reads. Follow the same suggestions as for "db file sequential reads."

Internal Fragmentation Considerations

- Watch for:
 - Bad settings for PCTFREE and PCTUSED for heap segments
 - Bad settings for PCTVERSION and RETENTION for LOB segments
 - Low density of data in segment
 - Direct loads followed by deletes (no inserts)
 - Indexes on tables with random updates and deletes with no further inserts
- Remedy:
 - Online segment shrink
 - Online redefinition
 - MOVE operations



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Segment fragmentation is a serious issue for two main reasons:

- Poor space utilization, because the free space under the HWM of one segment cannot be reused by other segments without performing a segment shrink first
- Poor performance, certain data access patterns cause more I/O and more I/O waits than is necessary to read the data. For example, a full table scan reads all the blocks below the HWM, whether or not there are rows in those blocks.

Note: In the case of LOBs, data in such segments is always accessed through an index. Therefore, there is no impact in access performance.

Internal fragmentation can impact heap, index, as well as LOB segments. The slide shows you the main reasons leading to internal fragmentation.

Fragmentation is an issue that database administrators (DBAs) have been battling for a long time. Use of locally managed tablespaces in conjunction with Automatic Segment Space Management (ASSM) offers the best way to minimize internal fragmentation.

However, after internal fragmentation has occurred, the traditional methods that you can use to solve the problem include the following:

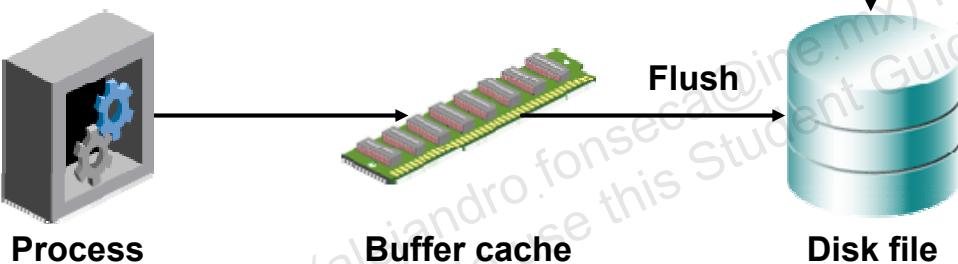
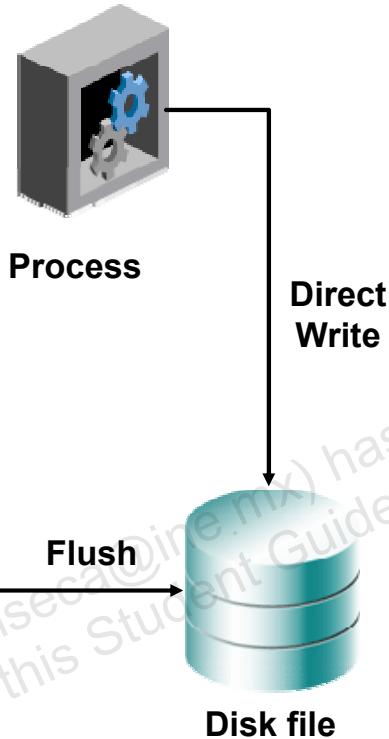
- ALTER TABLE MOVE, CREATE TABLE AS SELECT (CTAS), Import/Export
- Reorganize the object using online redefinition set of procedures.

These techniques are effective in removing fragmentation, but require additional disk space and involve considerable manual labor. The online segment shrink feature is more efficient.

I/O Modes

I/O can be written to disk in several ways by using different system calls:

- Standard I/O
- Synchronous I/O
- Asynchronous I/O
- Direct I/O



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In any OS, there are multiple methods of writing to disk.

The standard method writes the data to the operating system buffer cache, and then the buffer is written to disk by a kernel process at a later time. If the machine crashes or loses power before the buffer is written to disk, the data is lost. This method is fast, but not acceptable for Oracle database file I/O because of the potential for data loss.

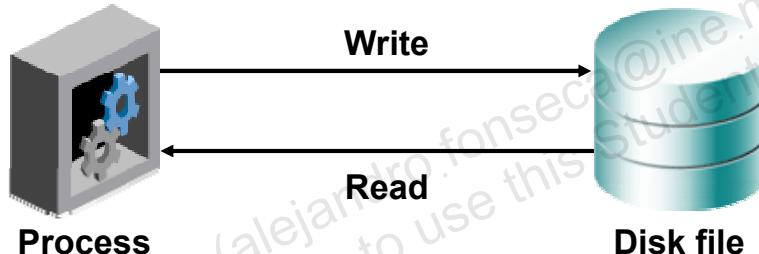
Synchronous I/O, also known as write-thru-cache, writes the data to the buffer but also forces an immediate write to the disk. The process that makes the write request is suspended until the write is completed. The completion is signaled by the disk controller when the buffer is written to disk. On some disk subsystems with large caches, the controller signals a completion when the buffer is written into the subsystem cache. Although synchronous writes are slower, they are the default method for DBWn and LGWR processes to perform their writes because this I/O method is reliable and supported on all operating systems.

Asynchronous writes allow instance background processes to make multiple write requests without being suspended after each request. The completion must still be signaled back to the process. This permits a much larger throughput because the background process can continue work without waiting for the disk I/O to complete. The ability to perform asynchronous writes varies by operating system.

Direct I/O

Direct I/O is considered to be the high-performance solution.

- Direct reads and writes do not use the OS buffer cache.
- Direct reads and writes can move larger buffers than file system I/Os.
- Direct I/O can be synchronous or asynchronous.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The main benefit of direct I/O is the lack of caching. The Oracle Database server caches its own database blocks. For normal reads in an online transaction processing (OLTP) environment, the simple read-ahead that is done by the file system is not effective. By avoiding the OS cache, the direct reads and writes are not copied in memory, and the application performing the reads and writes can choose much larger buffers for the data transfers than are allowed by the file system.

This also frees OS memory used for caching I/O that can be applied to the SGA, where the database server can make more effective use of the space.

Oracle recommends the use of direct I/O on any OS file system that supports it.

Oracle Cluster File System (OCFS and OCFS2), ASM, NFS, various Storage Area Networks, and Clustered File Systems are used for Oracle Real Application Clusters.

I/O can be set to be direct and asynchronous or direct and synchronous. Direct and asynchronous is the recommended setting for high performance systems.

Direct Path Read and Write Waits

Direct path wait events

- Direct path read
- Direct path read temp
- Direct path write
- Direct path write temp
- Direct path write (lob)



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Waits on direct path I/O (reads and writes) can be misleading. If asynchronous IO is supported (and in use), then Oracle can submit I/O requests and continue processing. It can then pick up the results of the I/O request later and will wait on `direct path read`, `write`, or `sync` waits until the required I/O completes, so total number of waits does not reflect the number of I/O requests and total time spent in "`direct path <type>`" does not always reflect the true wait time. In Oracle Database 12c, direct path I/O waits are collected in separate events by type of object, direct path read temp, and direct path write temp for temporary tablespaces; direct path write (lobs) for writes to uncached LOBs; and direct path read and write for others.

Direct path reads are generally used by Oracle when reading directly into PGA memory (as opposed to into the buffer cache). Direct path reads are used for:

- Sort I/O (when a sort does not fit in memory) – direct path read temp in Oracle Database 12c
- Parallel Query slaves
- Read ahead – For a parallel query using a range scan and asynchronous I/O is enabled. The process may issue an I/O request for a block that it expects to need in the near future.

Direct path writes are used for:

- Direct load operations (For example, Create Table as Select [CTAS] may use this.)
- Parallel DML operations
- Sort IO (when a sort does not fit in memory) - direct path write temp in Oracle Database 12c
- Writes to uncached "LOB" segments (later releases wait on "direct path write (lob)")

Waits on `direct path read temp` and `direct path write temp` indicate that the PGA and temporary tablespaces should be examined and if possible tuned, before implementing the solutions for direct path read and write waits below.

Waits on `direct path read (lob)` and `direct path write (lob)` indicate that uncached LOBs should be examined and if possible tuned, before implementing the solutions for direct path read and write waits below. LOB tuning is addressed in the *Oracle Database SecureFiles and Large Objects Developer's Guide 12c Release 1 (12.1)*.

For more details on tuning PGA size and temporary tablespace configuration, see the lesson titled “Tuning PGA and Temporary Space.”

Waits on direct path read and direct path write suggest that you examine:

- The `V$SESSION_EVENT` view to identify sessions with high numbers of waits
- The `V$SESSTAT` view to identify sessions with high "physical reads direct" or "physical writes direct" (statistic only present in newer Oracle releases)
- The `V$FILESTAT` view to see where the I/O is occurring
- The `V$SQLAREA` view for statements with `SORTS` and high `DISK_READS` (which may or may not be due to direct reads)

Based on the examination, possible solutions are:

- Ensure that the `DISK_ASYNCH_IO` parameter is TRUE. This is unlikely to reduce wait times from the wait event timings, but may reduce sessions elapsed times (because synchronous direct I/O is not accounted for in wait event timings).
- Ensure that the OS asynchronous I/O is configured correctly.
- Check for I/O heavy sessions/SQL and see if the amount of I/O can be reduced.
- Ensure that no disks are I/O bound.

Implementing SQL Tuning Recommendations

Tuning SQL is:

- The first tuning choice
- Usually the most cost effective
- Often the first recommendation of wait mitigation

Recommendations:

- Profiles
- Statistics
- Schema changes
- SQL Rewrites



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

SQL tuning is often the tuning recommendation with the highest impact. Even when other recommendations are higher, the SQL tuning can reduce the impact of other recommendations, such as CPU waits, insufficient memory, and I/O problems. SQL tuning should always be the first choice in tuning. It is usually the most cost effective option. The SQL Tuning Advisor runs automatically in Enterprise Edition (EE) providing recommendations for the SQL statements that use the most resources. For Standard Edition (SE), and EE without the Tuning and Diagnostics Packs, Statspack with the appropriate level will provide a list of the most resource-intensive SQL statements over the collection period.

Implementing recommendations:

- Profiles generated by SQL Tuning Advisor can be applied to the existing SQL statements to implement more efficient execution plans without modifying the SQL.
- Up-to-date statistics can help the optimizer choose better execution plans.
- Schema changes, such as additional indexes and materialized view, can help the optimizer choose a more efficient access method. This may not be allowed by your application vendor and still maintain support.
- SQL rewrites: This assumes that you have supported access to the SQL statements. Change the SQL statement to use better access paths.

Automatic Statistics Gathering

- STATISTICS_LEVEL = TYPICAL | ALL
- Statistics are gathered by the Optimizer Statistics Gathering automatic maintenance task.
- This task implicitly determines the following:
 - Database objects with missing or stale statistics
 - Appropriate sampling percentage necessary to gather good statistics on those objects
 - Appropriate columns that require histograms and the size of those histograms
 - Degree of parallelism for statistics gathering
 - Prioritization of objects on which to collect statistics



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Optimizer statistics are automatically gathered by the Optimizer Statistics Gathering task part of the Automatic Maintenance tasks. This task gathers statistics on all objects in the database that have missing or stale statistics.

This task is created automatically at database creation time and is managed by the Scheduler. This task gathers optimizer statistics by calling the DBMS_STATS.GATHER_DATABASE_STATS_JOB_PROC procedure. This procedure operates in a similar fashion to the DBMS_STATS.GATHER_DATABASE_STATS procedure by using the GATHER_AUTO option. The primary difference is that the GATHER_DATABASE_STATS_JOB_PROC procedure prioritizes the database objects that require statistics, so that those objects that need updated statistics the most are processed first. This ensures that the most needed statistics are gathered before the maintenance window closes.

You should make sure that automatic maintenance tasks are enabled, and that STATISTICS_LEVEL is set to at least TYPICAL.

Automatic Statistics Collection: Considerations

- You should still manually gather statistics in the following cases:
 - After bulk operations
 - When using external tables
 - To collect system statistics
 - To collect statistics on fixed objects
- Prevent automatic gathering for volatile tables:
 - Lock with statistics for representative values
 - Lock without statistics implies dynamic sampling.
- Set Optimizer Statistic Preferences for objects that need special handling.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

For tables that are being bulk-loaded, the statistics-gathering procedures should be run immediately following the load process.

Automatic statistics collection is not supported for external tables, system statistics, and statistics on fixed objects (such as the dynamic performance tables). These statistics should be gathered. External tables should have statistics gathered after a change in size. Statistics should be gathered on fixed objects after a change in workload.

With Oracle Database, you can lock statistics on a specified table by using the `LOCK_TABLE_STATS` procedure of the `DBMS_STATS` package.

You can lock statistics on a volatile table at a point when it is fully populated so that the table statistics are representative of the table population. This is good for interim tables that are volatile, but have a reasonable consistent size.

You can lock statistics on a table without statistics to prevent automatic statistics collection so that you can use dynamic sampling on a volatile table with no statistics. This option works best on tables that have a widely ranging size.

Optimizer statistics preferences can be set to direct the automatic task to use specific parameters when gathering statistics on specific objects or schemas.

Quiz

Automatic Statistics Gathering collects optimizer statistics:

- a. During the Automatic Maintenance tasks
- b. On fixed objects
- c. On external tables
- d. On all tables, including the data dictionary



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: a, d

Statistics are not gathered on fixed objects, such as dynamic performance views or external tables.

Statistics are gathered on any tables where the statistics become stale or are missing.

Summary

In this lesson, you should have learned how to:

- Diagnose and implement solutions for common wait events using the Performance Tuning Methodology
- Describe the Oracle Database Architecture related to common wait events
- Describe the Oracle Database Architecture related to common performance-related errors



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright © 2018, Oracle and/or its affiliates.

Lesson 21: Oracle Database Cloud Service: Overview

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.



Unauthorized reproduction or distribution prohibited. Copyright © 2018, Oracle and/or its affiliates.

Alejandro Fonseca (alejandro.roncero@ine.mx) has a non-transferable license to use this content.

Objectives



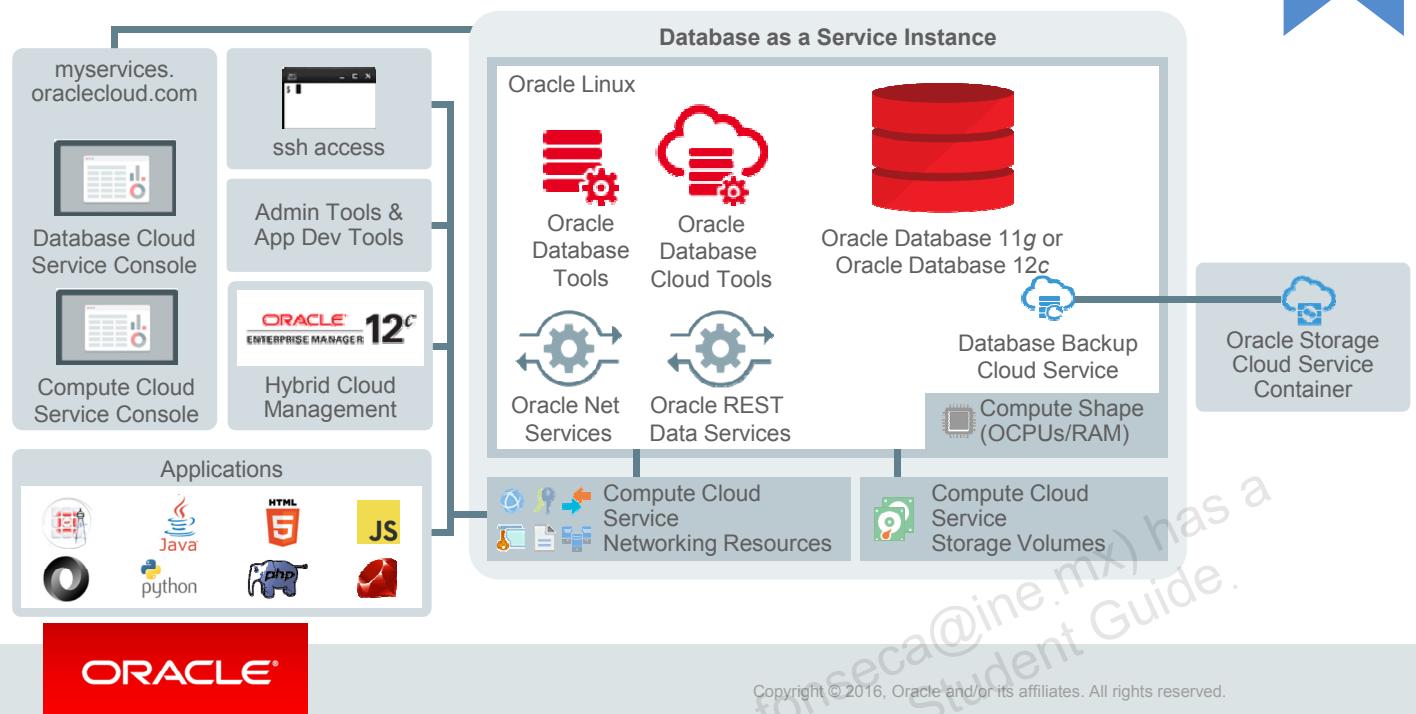
After completing this lesson, you should be able to:

- Describe the Oracle Database Cloud - Database as a Service (DBCS) architecture and features
- Describe the creation of a database deployment
- Manage the compute node associated with a database deployment
- Manage network access to a database deployment
- Describe Performance Management in DBCS Environment
- Identify what can be tuned in DBCS

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Database as a Service Architecture



Oracle Database Cloud Service provides you the ability to deploy Oracle databases in the Cloud, with each database deployment containing a single Oracle database. You have full access to the features and operations available with Oracle Database, but with Oracle providing the computing power, physical storage and (optionally) tooling to simplify routine database maintenance and management operations.

When you create database deployments, Database Cloud Service creates compute nodes to host the database, using computing and storage resources provided by Oracle Compute Cloud Service. Additionally, it provides access to the compute nodes (and thus to the database) using networking resources provided by Oracle Compute Cloud Service.

The Oracle Database Cloud Service includes Oracle Database and supporting software. An Oracle database is created using values you provide when creating the database deployment, and the database is started. Additionally, you can direct Database Cloud Service to set up automatic backups. Finally, the deployment includes cloud tooling that simplifies backup, recovery, patching and upgrade operations.

Database as a Service: Features and Tooling



Simple Provisioning



Automated Patching



Automated Backups



Any Language



Advanced Security



Local and Remote Management

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Oracle Database as a Service includes the following features and tools:

- **Simple Provisioning:** The Create Database Cloud Service Instance wizard is used to create a new database deployment.
- **Automated Patching:** Available patches are displayed through the Database Cloud Service console and can be applied with a single click.
- **Automated Backups:** Automatic backups can be set up. Cloud tooling provides for simplified backup and recovery operations.
- **Any Languages:** Database as a Service supports any language.
- **Advanced Security:** Comprehensive security, including encryption of data at rest and data in transit.
- **Local and Remote Management:** Enterprise Manager Database Control for Oracle Database 11g databases and Enterprise Manager Database Express for Oracle Database 12c database are part of the installation and can be used to monitor and manage the database. Enterprise Manager Cloud Control 13c can also be used to manage the database.



Software Editions: Included Database Options and Management Packs

Edition	Included Options	Included Packs
Standard	None	None
Enterprise	None	None
Enterprise - High Performance	Advanced Analytics, Advanced Compression, Advanced Security, Database Vault, Label Security, Multitenant, OLAP, Partitioning, Real Application Testing, Spatial and Graph	Cloud Management for Oracle Database, Database Lifecycle Management, Data Masking and Subsetting, Diagnostics, Tuning
Enterprise - Extreme Performance	Active Data Guard, Advanced Analytics, Advanced Compression, Advanced Security, Database In-Memory, Database Vault, Label Security, Multitenant, OLAP, Partitioning, Real Application Clusters, Real Application Testing, Spatial and Graph	Cloud Management for Oracle Database, Database Lifecycle Management, Data Masking and Subsetting, Diagnostics, Tuning

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

When you create a database deployment, you can choose from four software editions. The High Performance and Extreme Performance editions include the Oracle Database options and Oracle Management packs listed in the slide.

Accessing the Oracle Database Cloud Service Console

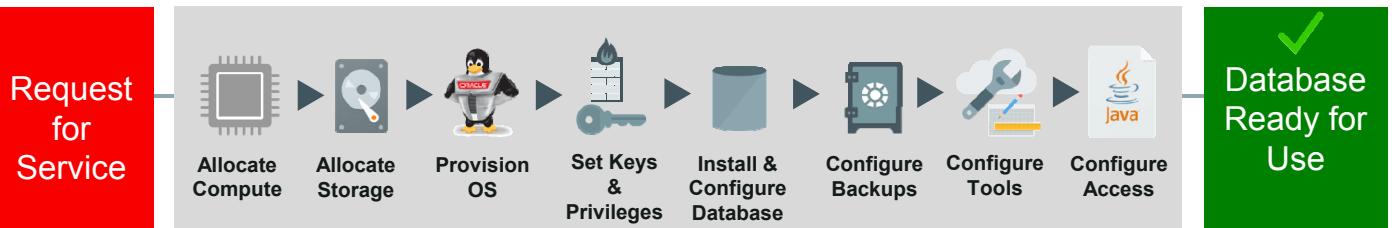


1. Display the Sign In to Oracle Cloud page by clicking the My Services URL link in your Welcome email or by following these instructions:
 - a. Open your web browser and go to the Oracle Cloud website.
 - b. Click Sign In.
 - c. In the My Services box, select the data center where your services are located.
 - d. Click Sign In to My Services.
2. On the Sign In to Oracle Cloud page, enter your username, password, and the name of your identity domain. Then, click Sign In.
3. On the My Services Dashboard, click the navigation menu in the top corner and then click Oracle Database Cloud Service.
4. If a Welcome page is displayed, view the list of database deployments by clicking Services next to “Database Cloud Service.”

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Automated Database Provisioning



Automated provisioning based on input to the Create Oracle Database Cloud Service Instance wizard.

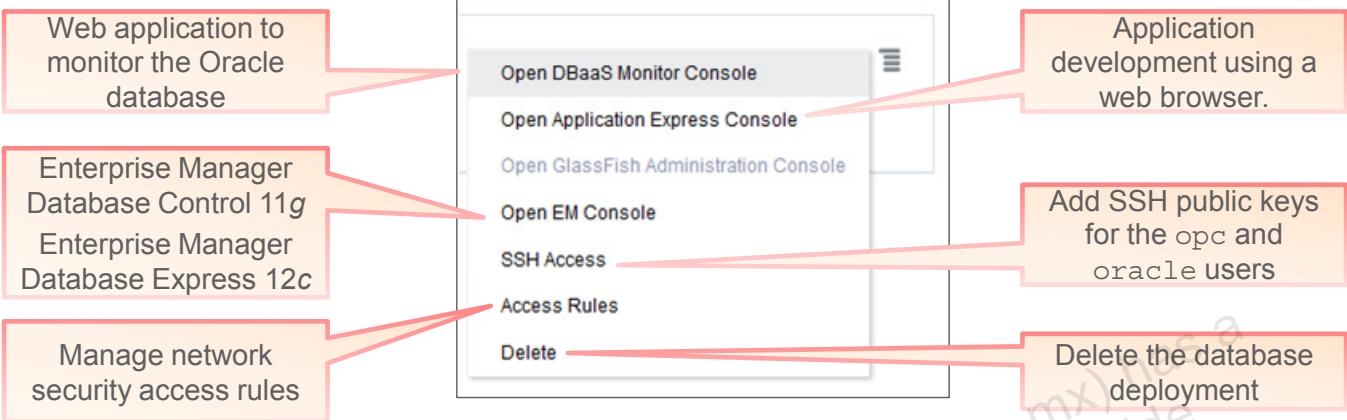
ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.



Accessing Tools from the Service Console

Expand the action menu on the Database Cloud Service console to access tools:



ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

The Oracle Database Cloud Service console enables you to manage your database deployment through the menu choices shown on the slide.

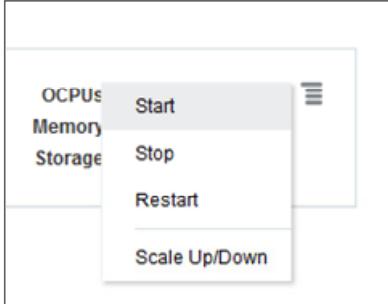
- Database deployments on Database as a Service include Oracle DBaaS Monitor Console a built-in monitor that provides a wide spectrum of information about Oracle Database status and resource usage.
- Oracle Application Express enables you to design, develop and deploy responsive, database-driven applications using only your web browser.
- To monitor and manage the Oracle database deployed on Database as a Service, you can use the standard management tool provided with the version of the database:
 - For Oracle Database 12c, use Enterprise Manager Database Express 12c.
 - For Oracle Database 11g, use Enterprise Manager 11g Database Control.
- Should the need arise, you can add SSH public keys for the `opc` and `oracle` users to the compute nodes associated with a database deployment on Oracle Database Enterprise Cloud Service.
- When you no longer require a database deployment on Database as a Service, you can delete it. Once deleted, the entry is removed from the list of database deployments displayed on the Database as a Service Console.

Note: Your choices from this menu may vary from what is shown based on the version of Database Cloud Service you are using.



Managing the Compute Node Associated With a Database Deployment

Manage the compute node from the Oracle Database Cloud Service console:



Action	Result of the Action
Start	Oracle Compute Cloud Service instance is allocated, resources are attached, and it is started.
Stop	Oracle Compute Cloud Service instance is stopped. Only actions after stop are start and delete.
Restart	Oracle Compute Cloud Service instance is stopped and immediately started again.

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Managing Network Access to Database as a Service



- By default, network access to the compute nodes associated is provided by Secure Shell (SSH) connections on port 22.
- To access network protocols and services using a port other than port 22:
 - Use the Oracle Compute Cloud Service console or the Oracle Database Cloud Service console to enable access
 - Create an SSH tunnel to the port

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Enabling Access to a Compute Node Port



- Database as a Service relies on Oracle Compute Cloud Service to provide secure network access to database deployments.
- Oracle Compute Cloud Service security rules created, but not enabled:
 - ora_p2_dbconsole: For port 1158, used by Enterprise Manager 11g Database Control
 - ora_p2_dbexpress: For port 5500, used by Enterprise Manager Database Express 12c
 - ora_p2_dblistener: For port 1521, used by SQL*Net
 - ora_p2_http: For port 80, used for HTTP connections
 - ora_p2_https: For port 443, used for HTTPS connections including Oracle REST Data Services, Oracle Application Express, and Oracle DBaaS Monitor
- Enable the security rules through the Database Cloud Service console

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Database as a Service relies on Oracle Compute Cloud Service to provide secure network access to database deployments. You can use the Oracle Compute Cloud Service console or the Oracle Database Cloud Service console to perform network access operations such as enabling access to a port on a compute node.

When a database deployment is created, the Oracle Compute Cloud Service security rules listed on the slide are created, but not enabled.

To enable access to a compute node port, you enable the appropriate security rule. When you enable one of the predefined security rules, the given port on the compute node is opened to the public internet.

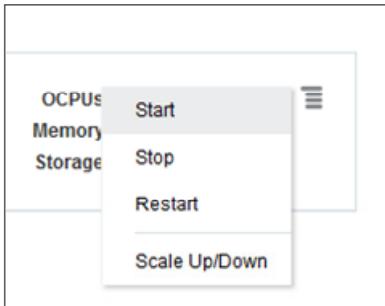
If you wish to enable access to a compute node port that is not associated with an existing security rule, you must create a new security rule to define the protocol associated with the port number and create a security rule.

If you wish to restrict access to a compute node port, to only permit connections from specific IP addresses, you must create a Security IP List and associate it to the security rule.

Scaling a Database Deployment



Scale the compute shape or storage from the Action menu in the Oracle Database Cloud Service console:



Action

Scale Up	Select a new compute shape. Add raw storage to the database deployment.
Scale Down	Select a new compute shape.

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

If a database deployment on Database as a Service is performing poorly or is running out of storage, you can scale up the environment supporting the database deployment.

Scaling the Compute Shape

When you scale the compute shape of a database deployment on Database as a Service, the deployment is put into Maintenance status during the operation and it is restarted. As a result of the restarting, any resources you've manually added using the Compute Cloud Service console become detached from the database deployment.

Scaling Storage

When you scale up the storage for a database deployment on Database as a Service, the deployment is put into Maintenance status during the operation and it is restarted. As a result of the restarting, any resources you've manually added to the database deployment using the Compute Cloud Service console become detached from the deployment.

When you scale up the storage for a database deployment, a Compute Cloud Service storage volume is created and attached to the deployment. This storage volume remains attached and available to the deployment even after it is restarted or is stopped and then started. Also, the storage volume exists until you delete the database deployment, at which time the storage volume is also deleted.

Performance Management in the Database Cloud Environment



In the Database Cloud Service (DBCS) several areas of tuning are managed in the cloud environment removing the need for tuning.

- Subscription cost is related to the resource usage
- Database instance is configured per best practices
- Resource usage is dependent on the application

In the database deployment, the cost in general is determined by resource usage. So it is in your best interest as a subscriber to reduce resource while maintaining the required performance levels.



Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Performance Monitoring and Tuning



Performance Monitoring and Tuning tasks can be either:

- Proactive monitoring/tuning:
 - Examining performance statistics at a regular interval to identify whether the system behavior and resource usage has changed.
- Reactive tuning:
 - Identifying an overused resource in the system and determining potential fixes.

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Proactive monitoring/tuning:

Proactive monitoring (or proactive tuning) usually occurs at a regularly scheduled interval and encompasses examining several performance statistics to identify whether the system behavior and resource usage has changed. Monitoring does not usually result in system configuration changes, unless the monitoring exposes a serious or developing problem. Consider monitoring as part of capacity planning, where resource consumption is examined to see changes in the way the application is being used, and the way the application is using the database and host resources.

Reactive tuning:

When tuning is left until the database is in production, it often becomes a reactive process to identify and fix the most serious bottleneck. The goal of tuning is usually to improve the use of a particular resource. In general, performance problems are a result of the overuse of a particular resource which becomes a bottleneck in the system.

What Can be Tuned in a DBCS Environment?



One of the benefits of the DBCS environment is that the amount of tuning required is minimal, but there will be uses that do not fall in the median group. So before you pay for more of whatever resource you believe is required, tune the following areas:

- Application Design
- SQL Statements
- Database Instance Initialization Parameters
- Memory Sizing
- Redo and Archive Logging

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

One of the benefits of the DBCS environment is that the amount of tuning required is minimal, but there will be uses that do not fall in the median group.

For instance: Applications that require more memory, either for Process memory, PGA, or Shared Global Memory, SGA; Compute intensive applications that require more CPU; or applications that require low latency disk access.

The DBCS instances can be scaled up, for CPU or Memory, and storage can be added to meet these kind of needs. These resources have an associated cost.

So before you pay for more of whatever resource you believe is required, tune the following areas:

1) Application Design:

A badly designed application is very difficult to tune without changing the design and code. It is good practice to have a DBA

experienced in Performance Tuning as part of your design team to avoid common design flaws and assist in developing test

cases for prototypes.

2) SQL Statements:

As a customer you do not want to pay for unnecessary resource usage due to poorly performing SQL. So tuning the application

SQL and the SQL optimization to use the available resources efficiently can reduce the overall cost of using a DBCS instance.

3) Database Instance Initialization Parameters:

The initialization parameters of the database instance affect: memory sizing (SGA and PGA), SQL optimization, SQL Plan use, resource plans, background process behavior and more. These parameters have been pre-tuned for most database deployments, but could be helpful in some deployments.

4) Memory Sizing:

In the initial database deployment an initial size is preset and the allocation between SGA and PGA is automatically adjusted as workload changes. A few uses of the database deployment may benefit by controlling the memory allocation more closely.

5) Redo and Archive Logging:

The amount of redo generated can affect performance. Configuring larger or more redo log files may help when there is a large volume of redo being generated



Summary

In this lesson, you should have learned how to:

- Describe the Oracle Database Cloud - Database as a Service architecture and features
- Describe the creation of a database deployment
- Manage the compute node associated with a database deployment
- Manage network access to a database deployment
- Describe Performance Management in DBCS Environment
- Identify what can be tuned in DBCS

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2018, Oracle and/or its affiliates.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

Using Statspack

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Install Statspack
- Create Statspack snapshots
- Generate Statspack reports
- Identify the major sections of the Statspack report

Introduction to Statspack

- Statspack is a set of scripts that capture and report on performance data from within the Oracle database.
- With Statspack scripts, you can:
 - Capture instance data by taking a “snapshot”
 - Store snapshot data in the database in a separate schema
 - Create reports between two snapshots
 - Mark snapshots as baseline information
 - Use the reports as part of a performance tuning method



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Statspack is a set of scripts and PL/SQL packages. Statspack enables you to capture point-in-time performance data called a snapshot into a schema dedicated to Statspack. You can create a report between snapshots and view the database activity and performance for the time period covered by the snapshots. You can preserve sets of snapshots by marking them as baseline information. Marking the snapshots protects them from deletion by the purge procedure.

The reports generated by Statspack provide performance tuning information that can be used as part of a tuning methodology.

Statspack Scripts

- Install Statspack by using `spcreate.sql`.
- Collect statistics with `statspack.snap`.
- Automate the collection of statistics with `spauto.sql`.
- Produce a report by using `spreport.sql` or `sprepsql.sql`.
- Purge Statspack data with `statspack.purge` or `sppurge.sql`.
- Truncate all Statspack tables with `sptrunc.sql`.
- Drop the Statspack repository with `spdrop.sql`.
- Export the Statspack repository with `spexp.par`.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Installation of the Statspack Package

The installation of the Statspack utility creates the `PERFSTAT` user, which owns all PL/SQL code and database objects created (including the Statspack tables, the constraints, and the Statspack package). During the installation, you are prompted for the `PERFSTAT` user's default and temporary tablespaces.

When initially installed, approximately 80 MB of the `PERFSTAT` user's default tablespace is used. This may increase later when the tables contain snapshot information.

Collecting Statistics

To take a snapshot of performance data, log on to SQL*Plus as the `PERFSTAT` user, and then execute the `STATSPACK.SNAP` procedure. This procedure stores the current performance statistics in the Statspack tables, which can be used with another snapshot taken at a later time to produce a report or a base line.

Automatically Collecting Statistics

To compare performance from one day, week, or year to the next, there must be multiple snapshots taken over a period of time. The best method to gather snapshots is to automate the collection at regular time intervals.

The `spauto.sql` script makes use of the `DBMS_JOB` package to provide an automated method for collecting statistics. The supplied script schedules a snapshot every hour, on the hour. This can be changed to suit the requirements of the system.

Producing a Report

To examine the change in statistics between two time periods, execute the `spreport.sql` file while connected as the `PERFSTAT` user. You are shown a list of collection periods and can select a start and end period. The difference between the statistics at each end point is then calculated and put into a file with a name of your choice.

Purge Statspack Data

The `sppurge.sql` script enables you to purge a range of snapshots. With the extended purge option SQL text, execution plans and segment Identifiers can also be purged. This operation cannot be rolled back.

Truncating Statspack Tables

If you want to truncate all performance data indiscriminately, it is possible to do this using `sptrunc.sql`. This script truncates all statistics data gathered, including snapshots marked as baselines.

Dropping the Statspack Repository

To drop the Statspack repository and remove the Statspack package, use the `spddrop.sql` script. If there are errors during the repository creation, you must drop the repository before you attempt to re-create it.

Exporting the Statspack Repository

If you want to share data with other sites, for example, if Oracle Support requires raw statistics, it is possible to export the `PERFSTAT` user. An export parameter file (`spuexp.par`) has been supplied for this purpose. To use this file, supply the export command with the `userid` parameter, along with the export parameter file name:

```
exp userid=perfstat/perfstat_password parfile=spuexp.par
```

This creates the `spuexp.dmp` file and the `spuexp.log` log file.

Note: It is also possible to upgrade Statspack tables to the latest version of the Oracle database that you are using. Scripts are provided to accomplish this task.

Installing Statspack

- Install Statspack:
 - Supplying parameters as requested interactively
 - Defining parameters in the session for batch mode

```
SQL> connect / as sysdba
SQL> define default_tablespace='SYSAUX'
SQL> define temporary_tablespace='TEMP'
SQL> define perfstat_password='erg8oiw'
SQL> @?/rdbms/admin/spcreate
SQL> undefine perfstat_password
```

- The spcreate.sql script creates the PERFSTAT user to own the Statspack repository and package.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Installation scripts create a user called PERFSTAT, which owns all PL/SQL code and database objects created including the Statspack tables, constraints, and the STATSPACK package.

There are two ways to install Statspack: interactively or in batch mode as shown in the slide. Batch mode is useful when you do not want to be prompted for the PERFSTAT user's password and the default and temporary tablespaces. To install in batch mode, you must assign values to the SQL*Plus variables that specify the password and the default and temporary tablespaces before running spcreate.sql. Both methods require the default tablespace to be defined. You cannot use the SYSTEM tablespace as the default tablespace for the PERFSTAT user. SYSAUX is the recommended default tablespace.

If you receive an error during the installation, execute spdrop.sql before you rerun spcreate.sql.

Note: To execute the installation script, you must use SQL*Plus and connect as a user with the SYSDBA privilege.

Capturing Statspack Snapshots

- Capture a snapshot:

```
SQL> connect perfstat/erg8oiw  
SQL> execute statspack.snap;
```

- Use the function to determine the snapshot number:

```
SQL> variable snap number;  
SQL> begin :snap := statspack.snap; end;/  
SQL print snap
```

- Automate snapshot collection:

```
SQL> connect perfstat/erg8oiw  
SQL> @spauto
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The simplest, interactive way to take a snapshot is to log in to SQL*Plus as the `PERFSTAT` user and execute the `STATSPACK.SNAP` procedure as shown in the slide.

This stores the current values for the performance statistics in the Statspack tables, and it can be used as a baseline snapshot for comparison with another snapshot taken at a later time.

If you want to automate the gathering and reporting phases (such as during a benchmark), you may need to know the `SNAP_ID` of the snapshot just taken. To take a snapshot and display the `SNAP_ID`, call the `STATSPACK.SNAP` function as shown in the slide.

To use an Oracle-automated method for collecting statistics, you can use `DBMS_JOB`. A sample script is supplied in `spauto.sql`, which schedules a snapshot every hour, on the hour. This script should be run as `PERFSTAT`. The `JOB_QUEUE_PROCESSES` initialization parameter should be set to a number greater than zero before automatic statistics gathering can run. The script also reports the corresponding job number. You can use the `DBMS_JOB` package to change the interval of statistics collection, force the job to run immediately, or to remove automatic job collection.

Note: On UNIX systems, you could use utilities such as `cron` or `at` to automate the snapshot collection.

Configuring Snapshot Data Capture

- Temporarily change a capture variable:

```
SQL> execute statspack.snap(i_snap_level=>6);
```

- Change defaults and take a snapshot:

```
SQL> execute statspack.snap -  
(i_snap_level=>10, i_modify_parameter=>'true');
```

- Modify defaults without taking a snapshot:

```
SQL> execute statspack.modify_statspack_parameter -  
(i_snap_level=>10, i_buffer_gets_th=>10000,-  
i_disk_reads_th=>1000);
```

- Include session details in a snapshot:

```
SQL> execute statspack.snap(i_session_id=>3);
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Both the snapshot level and the thresholds specified affect the amount of data Statspack captures. You can control the amount of information gathered by the package by specifying a snapshot level. The higher the snapshot level, the more the data gathered. The default level set by the installation is level 5. Level details are shown in a later slide.

There are threshold parameters that can be configured when collecting data on SQL statements. Data is captured on any SQL statements that breach the specified thresholds.

To temporarily use a snapshot level or threshold that is different from default snapshot values, simply specify the required threshold or snapshot level when taking the snapshot. This value is used only for the immediate snapshot taken, as shown in the slide.

You can save the new value as the instance default, by using I MODIFY PARAMETER, as shown in the slide. You can also change the defaults without taking a snapshot, using the STATSPACK.MODIFY_STATSPLIT_PARAMETER procedure.

To gather session statistics and wait events for a particular session, in addition to the instance statistics and wait events, specify the session ID in the call to Statspack.

Statspack Snapshot Levels

Level =>	Statistics captured
0	General performance
5	SQL statements
6	SQL plans and SQL plan usage
7	Segment-level statistics
10	Parent and child latches

AWR captures them all by default!



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

All levels contain general performance statistics including the following: wait statistics, system events, system statistics, rollback segment data, row cache, SGA, background events, session events, lock statistics, buffer pool statistics, latch statistics, resource limit, and enqueue statistics. Statistics are included for each of the following, if enabled: Automatic Undo Management, buffer cache advisory data, automatic PGA memory management, and Cluster DB statistics.

Levels 5 and above gather the performance data on SQL statements that exceed one of six predefined threshold parameters for resource usage. The time required for the snapshot to complete depends on the value of `SHARED_POOL_SIZE` and the number of SQL statements in the shared pool at the time the snapshot is taken. You can change the predefined threshold values. The thresholds are:

- Number of executions of the SQL statement (default 100)
- Number of disk reads performed by the SQL statement (default 1,000)
- Number of parse calls performed by the SQL statement (default 1,000)
- Number of buffer gets performed by the SQL statement (default 10,000)
- Size of sharable memory used by the SQL statement (default 1 MB)
- Version count for the SQL statement (default 20)

Level 6 includes all statistics gathered at the lower levels, and gathers optimizer execution plans and plan usage data for each of the high-resource-usage SQL statements captured. A level 6 snapshot gathers information that is invaluable when determining whether the execution plan used for a SQL statement has changed. Therefore, level 6 snapshots should be used whenever there is the possibility that a plan may change, such as after large data loads, or after gathering new optimizer statistics. The plan for a SQL statement is captured only if the statement is in the shared pool at the time that the snapshot is taken, and exceeds one of the SQL thresholds. You can gather plans for all statements in the shared pool, if you temporarily specify the executions threshold (`I_EXECUTIONS_TH`) to be zero (0) for those snapshots.

Level 7 includes all statistics gathered at the lower levels, and gathers the performance data on highly used segments. A level 7 snapshot captures segment-level statistics for segments that exceed one of seven threshold parameters that measure access or contention rates. You can change the default threshold values for:

- Number of logical reads on the segment (default 10,000)
- Number of physical reads on the segment (default 1,000)
- Number of buffer busy waits on the segment (default 100)
- Number of row lock waits on the segment (default 100)
- Number of ITL waits on the segment (default 100)
- Number of global cache Consistent Read blocks served (default 1,000)
- Number of global cache Current blocks served (default 1,000)

Levels ≥ 10 include all statistics gathered at the lower levels, and gathers parent and child latch information. Data gathered at this level can sometimes cause the snapshot to take longer to complete, and should only be used when advised by Oracle personnel.

Note: Refer to the `spdoc.txt` file for more information about the various levels and thresholds.

Statspack Baselines and Purging

```
SQL> exec statspack.make_baseline -
2  (i_begin_snap => 45, i_end_snap      => 50);
```

```
SQL> exec statspack.statspack.make_baseline(2,4,false);
```

```
SQL> exec statspack.make_baseline -
2  (to_date('31-AUG-2009 09:00','DD-MON-YYYY HH24:MI'),-
3  to_date('31-AUG-2009 12:00','DD-MON-YYYY HH24:MI'));
```

```
SQL> exec statspack.clear_baseline -
2  (to_date('13-DEC-2009 23:00','DD-MON-YYYY HH24:MI'),-
3  to_date('14-FEB-2010 02:00','DD-MON-YYYY HH24:MI'));
```

```
SQL> exec statspack.purge -
2  (i_begin_date=>to_date('01-JAN-2009','DD-MON-YYYY'),-
3  i_end_date   =>to_date('02-JAN-2009','DD-MON-YYYY'),-
4  i_extended_purge=>TRUE);
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The PERFSTAT schema can grow very large with many snapshots. You can purge unnecessary data from the PERFSTAT schema by using STATSPACK.PURGE. By default, PURGE removes all snapshots, but you can identify and keep snapshot data by creating baselines. After you have determined which snap IDs or times of day most represent a particular workload whose performance data you would like to keep, you can mark the data as baselines. You can later decide to clear one or more baselines. Clearing the baseline does not remove the data; it just identifies the data as candidates for purging. You cannot roll back a purge operation.

The slide shows you examples of managing baselines of Statspack data:

- Making a baseline of snaps 45 and 50 including the range of snapshots in between
- Making a baseline of only snapshots 2 and 4 (excluding the snapshots in between)
- Making a baseline of snapshots taken between August 31, 2009 at 9:00 AM and August 31, 2009 at 12:00 noon
- Clearing an existing baseline including all the snapshots between December 13, 2009 at 11:00 PM and February 14, 2010 at 2:00 AM
- Purguing all snapshots that fall between January 1, 2009 and January 2, 2009; also performing an extended purge

In earlier releases of Statspack, Statspack identifier tables that contained SQL Text, SQL Execution plans, and Segment identifiers were not purged. Now you can purge unreferenced data in these tables by requesting that the extended purge be performed at the same time as the normal purge. Purging this data can be resource intensive, so you may choose to perform an extended purge less frequently than the normal purge.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

Reporting with Statspack

```
SQL> connect perfstat/perfstat_password  
SQL> @?/rdbms/admin/spreport
```

```
SQL> connect perfstat/perfstat_password  
SQL> define begin_snap=1  
SQL> define end_snap=2  
SQL> define report_name=batch_run  
SQL> @?/rdbms/admin/spreport
```

```
SQL> connect perfstat/perfstat_password  
SQL> define begin_snap=39  
SQL> define end_snap=40  
SQL> define hash_value=1988538571  
SQL> define report_name=batch_sql_run  
SQL> @sprepsql
```



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can generate a performance report from two snapshots. There are two reports available:

- The instance report (`spreport.sql` and `sprepins.sql`) is a general instance health report, covering all aspects of instance performance. The instance report calculates and prints ratios, increases, and so on for all statistics between the two snapshot periods.
- The SQL report (`sprepsql.sql` and `sprsqins.sql`) is a report for a specific SQL statement. The SQL report is usually run after examining the high-load SQL sections of the instance health report. The SQL report displays SQL-specific statistics, the complete SQL text, and, if a level 6 snapshot or above has been taken, information about any SQL plans associated with that statement.

Both reports prompt for the beginning snapshot ID, the ending snapshot ID, and the report name. The SQL report also requests a hash value for the SQL statement to be reported on. You can configure each report by using parameters such as the total number of rows of SQL output to display in each SQL section or the number of days of snapshots to list. Separating the phase of data gathering from producing a report provides the flexibility of basing a report on any data points selected.

Note: The instance must not have been shut down between the times that the beginning and ending snapshots were taken.

Statspack Considerations

- Set STATISTICS_LEVEL = TYPICAL.
- Set TIMED_STATISTICS = TRUE to collect timing information.
- Avoid overlap between Statspack and AWR.
- Choose a sensible snapshot interval.
- Fix interval when “#####” appears in the report output.
- Gather optimizer statistics on the PERFSTAT schema.



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The STATISTICS_LEVEL parameter is set to TYPICAL by default. When STATISTICS_LEVEL is set to TYPICAL or ALL, Time Model data is populated by the server, and can be captured by Statspack. At a minimum, to have the most useful data available for tuning, you should set TIMED_STATISTICS to TRUE before the creation of a snapshot. Whenever STATISTICS_LEVEL is set to TYPICAL or ALL, TIMED_STATISTICS is automatically set to TRUE.

If STATISTICS_LEVEL is TYPICAL or ALL, and you do not use the Database Diagnostics pack, then you should disable AWR data capture by setting the AWR snapshot schedule interval to zero. If you use both Statspack and AWR, schedule the Statspack snapshots to avoid the times when AWR is attempting to capture data. By default, AWR snapshots are taken every hour, on the hour.

In general, snapshot data should be collected in intervals of an hour. In the unusual situation that finer-grained analysis is required, the snapshot interval can be changed for the duration of that problem analysis. Snapshot intervals of less than 15 minutes are seldom useful.

When a value is too large for a Statspack field, it is represented by a series of pound signs, such as #####. The main reason for this overflow is an analysis interval that is too large. Reduce the length of time between snapshots selected for the report, or choose snapshots that are closer in time.

For best performance when running the performance reports, optimizer statistics should be gathered on the Statspack schema. The Oracle Database server automatically gathers optimizer statistics on database segments when the segments become stale. If you have disabled the automatic job, you should gather statistics manually using the following command:

```
execute dbms_stats.gather_schema_stats(ownname=>'PERFSTAT', cascade=>true);
```

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.

Statspack and AWR Reports

The Statspack and AWR reports are designed to be used top down.

- The first few pages contain the following:
 - Summary Information
 - Load Profile (useful with baseline)
 - Instance Efficiency (useful with baseline)
 - Top 10 Foreground Events
 - Time Model
- The following pages show detailed data:
 - SQL
 - IO
 - Wait Statistics



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Statspack and AWR reports are designed to be used top down. The first few pages always indicate the biggest bottleneck to investigate. The Summary Information section shows the time interval being reported. Load Profile shows the level and type activity. This is very useful as a sanity check, to be sure that the workload has not changed significantly. The Instance Efficiency section gives the traditional ratios plus additional efficiency percentages that help identify and confirm the areas that need tuning.

The Top 10 Foreground Wait Events section identifies which events are accounting for the most time by percentage. The Time Model section shows the time and the ratio of time consumed by a specific event to DB_TIME.

The Time Model entries that relate to the Top 5 Timed Events indicate the possible impact tuning each event could have.

The subsequent pages of the Statspack and AWR reports give detailed information about a variety of database areas. These detailed sections are used to confirm diagnoses and plan remedies for the symptoms shown on the first pages.

Reading an Statspack or AWR Report

- Start with the summary data at the top.
 - Top 10 Foreground Events
 - Wait Events and Background Wait Events
 - Wait Event Histogram
 - Load Profile (useful with baseline)
 - Instance Efficiency (useful with baseline)
 - Time Model
- Drill down to specific sections.
 - Indicated by top wait events



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When diagnosing a possible database problem, start by checking the host hardware and OS, the middle tier or the network, and other resources outside the database. If these are not being completely consumed, it is time to look at the database.

Start at the Top

Use the Top 10 Foreground Events section to identify which events are accounting for the most time by percentage. After these are identified, look at the Wait Events and Background Wait Events sections for the average wait time for the highest-ranking events. This data can sometimes provide insight into the scale of the wait. Compare the average with data in the Wait Event Histogram section. Is the average indicative of the typical wait time, or is the average wait time severely skewed by a small number of atypical waits?

Look at the Load Profile, Instance Efficiency, and Time Model sections. Pay attention to any statistics or ratios that are related to the top wait events. Is there a single consistent picture? If not, note other potential issues to investigate while looking at the top events, but do not be distracted from the top wait events. Scan the other statistics. Are there any statistics in the Load Profile that are unusually high for this site, or any ratios in the Instance Efficiency section, which are atypical for this site (when compared to a baseline)? Next, drill down to the appropriate section in the Statspack report for additional data.

Statspack/AWR Report Drilldown Sections

Drilldown	Section
LC	Shared Pool Statistics
CPU	Host CPU
CPU	Instance CPU
MEM	Virtual Memory Paging
CPU	SQL ordered by CPU
IO	SQL ordered by Elapsed
CPU	SQL ordered by Gets
IO	SQL ordered by Reads
LC	SQL ordered by Parse Calls

CONT: Block contention
CPU: CPU consumption
ENQ: Enqueue
IO: IO consumption
LAT: Latch contention

LC: Library cache
MEM: Memory consumption
PGAM: PGA memory consumption
RBS: Rollback segment
UNDO: Automatic undo
SP: Shared pool



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The relevant sections to examine are indicated by the top wait events.

The slide shows you a partial list of the most important Statspack sections indicating which sections can be used for possible drilldown. Each wait event falls into a type of problem. These types are listed in the slide. Use the type to determine the section to examine for more detailed information.

While drilling down, determine whether the data in these sections is consistent with the wait events? If not, identify related areas in the report for an alternative data source.

Analyze the detailed data in the drilldown sections after identifying candidate problems and resources showing contention. Consider whether there is sufficient data to build a plausible theory for the cause and resolution of the problem.

Use the *Oracle Database Performance Tuning Guide* as a resource to identify the cause and the resolution of the contention. The manual includes detailed descriptions of how to diagnose causes and solutions for wait events and how to optimally configure the instance to avoid problems.

For more details on using Statspack for tuning, see My Oracle Support notes; two are especially helpful: Note 228913.1: *Systemwide Tuning using STATSPACK Reports* and Note 942241.1: *FAQ Statspack Complete Reference*.

Note: In Oracle Database, many of the commonly requested resources have their own wait events, such as common latches and all enqueues. This makes it easier to identify contention. For example, library cache latch-related wait events would include “latch: library cache,” “latch: library cache lock,” and “latch: library cache pin.”

Drilldown	Section
LC	SQL ordered by Sharable Memory
LC	SQL ordered by Version Count
IO	IO Sections
IOMEM	Buffer Cache
PGAMIO	PGA Memory Stats
PGAM	Process Memory Stats
ENQ	Enqueue activity
RBS UNDO	Rollback and Undo
LAT	Latch Statistics
CPU	Segments by Logical Reads
IO	Segments by Physical Reads
CONT	Segments by Row Lock Waits
CONT	Segments by ITL Waits
CONT	Segments by Buffer Busy Waits
LC	Dictionary Cache
LC	Library Cache Activity
STREA	Streams
SP LC	SGA breakdown difference
SP LC	SQL Memory Statistics

Report Drilldown Examples

- If the top timed event is related to I/O waits, look at:
 - SQL ordered by Reads
 - SQL ordered by Elapsed
 - Tablespace IO Stats
 - File IO Stats
 - File IO Histogram
- If the top timed event is related to CPU usage, look at:
 - Load Profile
 - Time Model
 - SQL ordered by CPU
 - SQL ordered by Gets



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

If the top events are I/O related (db file sequential read and db file scattered read), look at the “SQL ordered by Reads” or “SQL ordered by Elapsed” sections, and the Tablespace IO Stats, File IO Stats, and File IO Histogram sections.

If the top timed event is CPU time, look at the number of logical reads in Load Profile, and the top statistics in the Time Model data. Look at SQL by CPU or SQL by Gets. Correlate the output. Does the evidence point to SQL execution?

Note: On a well-performing system, the top events are likely to be CPU time, db file scattered read, and db file sequential read.

Load Profile Section

- Allows characterization of the application
- Can point toward potential problems:
 - High hard parse rate
 - High I/O rate
 - High login rate
- Is more useful if you have a comparable baseline
- Answers “What has changed?”
 - Txn/sec change implies changed workload.
 - Redo size/txn implies changed transaction mix.
 - Physical reads/txn implies changed SQL or plan.

Load Profile	Per Second	Per Transaction
Redo size:	186,146.10	518.05
Logical reads:	1,670.36	4.65
Block changes:	1,451.40	4.04
Physical reads:	0.01	0.00
Physical writes:	0.66	0.00
User calls:	8.22	0.02
Parses:	11.20	0.03
Hard parses:	1.03	0.00
Sorts:	9.01	0.03
Logons:	0.20	0.00
Executes:	382.20	1.06
Transactions:	359.32	
% Blocks changed per Read:	86.89	Recursive Call %: 99.39
Rollback per transaction %:	0.01	Rows per Sort: 8.63



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Load Profile section indicates the nature of the workload on the system. It includes the key V\$SYSSTAT statistics needed to characterize the system. From the Load Profile, you can determine the rates of:

- Logical and physical writes
- Logical and physical reads
- Hard and soft parses
- Session logons
- User activity

The values are shown per second (for absolute rates) and per transaction (for easier comparison with your application per-transaction rates to determine whether the application code, application use, or execution plans have changed).

This is where knowing what is typical on a system helps. If, for example, you know the typical number of logical reads/sec and redo size/sec when the system performs well, you can look at these values and see whether they are different when the system performs badly. To continue with the example, is the system doing twice the number of logical I/Os now than when the system performed well last week? Has the workload increased, or has the execution plan of key statements changed?

Time Model Section

The Time Model section lists a set of statistics. It:

- Gives an overview of where time is spent inside the Oracle Database server
- Is reported from the `V$SYS_TIME_MODEL` view
- Gauges the performance impact of any entity of the database, and provides drilldown

Statistic	Time (s)	% of DB time
sql execute elapsed time	620.7	98.8
DB CPU	78.1	12.4
PL/SQL execution elapsed time	10.7	1.7
parse time elapsed	7.9	1.3
hard parse elapsed time	4.1	.7
connection management call elapsed	2.5	.4
PL/SQL compilation elapsed time	0.4	.1
hard parse (sharing criteria) elaps	0.0	.0
repeated bind elapsed time	0.0	.0
sequence load elapsed time	0.0	.0
DB time	628.4	
background elapsed time	62.1	
background cpu time	13.6	

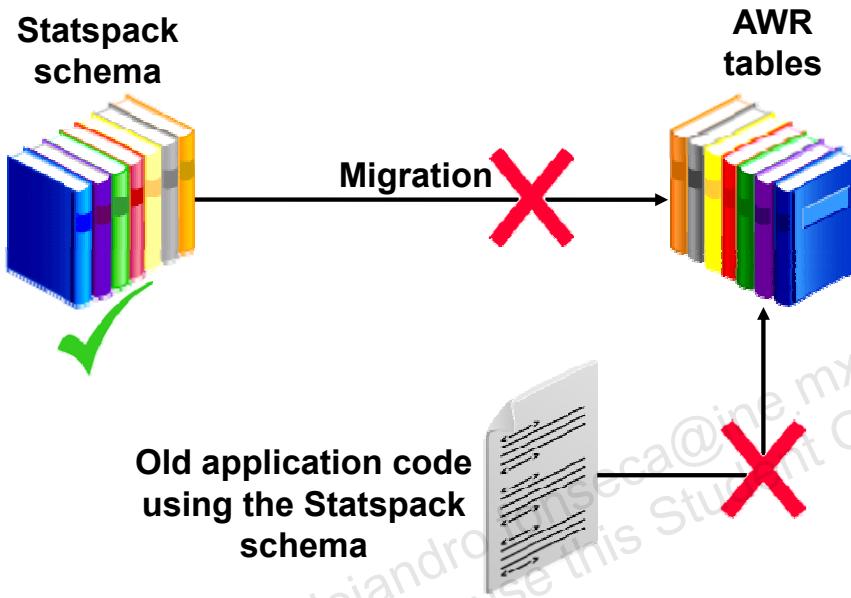


Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Time Model section is captured from a view called `V$SYS_TIME_MODEL`. Time Model data is intended to be used to help identify which sections to drill down to. For example, in earlier releases, a strange set of wait events and latches would indicate that the application was performing an excess of connection management (logon/logoff activities). In prior releases, only by knowing which latches to look for was it possible to track down this inefficient activity. The Time Model tracks time by different types of operations in the database, thus making it simpler to identify the types of operations that the instance is spending its time performing.

Some of the statistics measure the CPU time used for an operation, and others measure the total elapsed time for that operation. The name of the statistic indicates which of these two it is measuring. Elapsed time includes wait time.

Statspack and AWR



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In the past, historical data could be obtained manually by using Statspack. Some DBAs wrote applications to report on Statspack data. These reports can still be used in Oracle Database 12c; however, if you want to use Automatic Workload Repository instead, then you need to change your application code.

Statspack users should switch to Automatic Workload Repository in Oracle Database 12c Enterprise Edition. AWR is not available to Standard Edition users.

There is no supported path to migrate Statspack data into AWR. Also, there is no view created on top of AWR to simulate the Statspack schema.

Summary

In this lesson, you should have learned how to:

- Install Statspack
- Create Statspack snapshots
- Generate Statspack reports
- Identify the major sections of the Statspack report

Practice Overview: Use Statspack

This practice covers the following topics:

- Installing Statspack
- Creating Statspack snapshots
- Generating Statspack reports

Unauthorized reproduction or distribution prohibited. Copyright© 2018, Oracle and/or its affiliates.

Alejandro Fonseca (alejandro.fonseca@ine.mx) has a
non-transferable license to use this Student Guide.