

## **Oracle Database 12c: Performance Management and Tuning**

**Student Guide** | Jc`ia Y=

D79236GC10

Edition 1.0

May 2015

D86567

**ORACLE®**

**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
trcsest 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

## 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

# 1

## Introduction

ORACLE®

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

## Course Objectives

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

- Diagnose and tune common SQL-related performance problems
- Diagnose and tune common instance-related performance problems
- Use database advisors to proactively tune an Oracle Database
- Use the tools based on Automatic Workload Repository (AWR) to tune an Oracle Database
- Use the Enterprise Manager performance-related pages to monitor an Oracle Database



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

# Organization

- Monitoring and Diagnostics
  - Monitoring using available tools
  - Identifying the problem
  - Using AWR-based tools
- SQL Tuning
  - Identifying and tuning SQL statements by influencing the optimizer
  - Managing change
    - SQL Performance Management
    - Real Application Testing
- Instance Tuning
  - Tuning memory components
  - Tuning space usage and I/O



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

The focus of this course is on the performance tuning tasks that a DBA could be expected to perform in a single instance database environment. SQL tuning is limited to tasks such as adjusting the data structures and maintaining statistics because the DBA does not usually have access or permission to change the SQL code. This course is organized into three main divisions:

- **Monitoring and diagnostics:** Includes the monitoring, diagnostic, and reporting tools that the DBA needs to identify tuning issues proactively. Both the basic tools that are included with every database and those tools that require the optional packs are used.
- **SQL tuning:** Includes identifying high-load SQL statements, influencing the optimizer, and managing change. The basics of finding the problem SQL statements, and reading and tracing the execution plan are discussed. In addition, this division includes methods for managing SQL plan changes due to growth or minor changes and testing methods for anticipating major changes to the environment.
- **Instance tuning:** Includes the memory components, space management, and I/O. This division covers using the diagnostic tools to determine the specific problem area and implement a solution.

# Agenda

Day	Lesson	Topic
1	1	Introduction
1	2	Basic Tuning Diagnostics
1	3	Using Automatic Workload Repository
1	4	Defining Scope of Performance Issues
1	5	Using Metrics and Alerts
2	6	Using Baselines
2	7	Using AWR-Based Tools
2	8	Real-Time Database Operation Monitoring
2	9	Monitoring Applications
3	10	Identifying Problem SQL Statements
3	11	Influencing the Optimizer

ORACLE®

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

# Agenda

Day	Lesson	Topic
3	12	Reducing the Cost of SQL Operations
3	13	Using SQL Performance Analyzer
4	14	SQL Performance Management
4	15	Using Database Replay
4	16	Tuning the Shared Pool
4	17	Tuning the Buffer Cache
5	18	Tuning PGA and Temporary Space
5	19	Automatic Memory Management
5	20	Performance Tuning Summary
(opt)	Appx A	Using Statspack (optional)



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

## Topics Not Included in This Course

- 24x7 availability
- Online operations
- Backup performance
- Parallel operations
- Oracle Streams and Data Guard performance issues
- Real Application Clusters
- Operating system–specific issues
- Application-specific issues, such as LOB handling



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

Diagnosing performance issues in any configuration of an Oracle Database instance starts with the same basic techniques. Issues specific to certain features and options are covered in the courses specifically for those.

Performance issues with the high availability options are covered in the respective courses for Oracle Streams, Oracle Data Guard, and Oracle Real Application Clusters.

Backup performance configuration is covered in the Backup and Recovery Workshop course.

Parallel operations and partitioning are covered in the Data Warehouse curriculum.

Application-specific issues, such as handling of the storage parameter settings for Large Objects (LOB), are covered in the Application Developer documentation.

# What Is Performance Management?

Components of performance management:

- Service level agreements (SLAs)
- Goals and objectives
- Monitoring
- Tools

Managing performance:

- Planning
- Anticipating change



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

Performance management is the art of managing the actual performance of the database over time and environment changes. It also includes managing expectations of users and stakeholders.

In this course, most of the topics deal with the details of performance issues, such as “when I have this type of problem, I should do this type of diagnosis”; however, there is a higher level to consider.

The dimensions of managing performance include service level agreements (SLAs), goals and objectives, monitoring, and tools.

An SLA gives you and your customers a firm commitment of what constitutes acceptable performance. These are generally considered a contract.

Goals and objectives form a set of guidance for performance. These:

- Can be standards that are desirable, but above, or outside the “must” have items of the SLA
- Can be precursors to the SLA during design and development
- Can become de facto SLA when the customer depends on these levels of service, and expects them

Monitoring is required on a regular basis. Good decisions about what to tune come from being able to answer the questions: What has changed? What is the current bottleneck?, What is the Performance trend? To get the answers, regular performance monitoring reports and comparisons between reports over time are required. A baseline of the performance metrics when the database is performing acceptably is extremely helpful in determining what changed when the performance is degraded.

The suite of tools you choose will affect the types of performance analysis you can do. If the tools do not address a possible class of performance issues, these issues will go unnoticed.

Organizations have various methods to manage the services they provide. These can range from “We wait until someone complains” to a complete plan detailing the four Performance Management components listed. Those organizations with such a plan will seldom be surprised by a sudden degradation in performance. They may be able to anticipate and prevent major performance issues from becoming apparent to customers by performing proactive tuning. These organizations will often have test and QA instances to understand the effect changes will have on the production environment before customers see the changes.

## Who Tunes?

- Database administrators
- Application architects
- Application designers
- Application developers
- System administrators
- Storage administrators
- Network administrators



ORACLE®

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

Everyone involved with the Oracle Database software including system architects, designers, developers, database administrators, system administrators, storage administrators, and network administrators should think about performance.

If problems develop, the database administrator (DBA) usually makes the first attempt at resolving them. Therefore, the DBA should have an accurate overview of all the applications in the database and their interactions with each other. DBAs often enlist the aid of developers for application tuning, or system administrators for tuning the OS. This course is intended for DBAs responsible for tuning and monitoring an Oracle database. However, anyone involved in the design, development, and deployment of an Oracle database can also benefit.

# What Does the DBA Tune?

Performance tuning areas:

- Application:
  - SQL statement performance Shared with developers
  - Change management
- Instance tuning:
  - Memory
  - Database structure
  - Instance configuration
- Operating system interactions:
  - I/O Shared with SA
  - Swap
  - Parameters



**ORACLE**

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

To have a well-tuned system, you must carefully design systems and applications. Common performance problems can be grouped as follows:

- **Application issues:** Poorly written SQL, serialized resources, and poor session management
- **Instance issues:** Memory, I/O, and instance configuration
- **Operating system issues:** Swap, I/O, parameter settings, and network issues

You achieve the largest return on investment of time and effort by tuning the application. Tuning the SQL statements, the access paths, and the storage structures are important parts of application tuning.

Instance tuning and application tuning use different sets of skills and tools. Separate courses are available to address the specific skills and tools used in application tuning. Application tuning is dependent on the type of application. Data warehouse applications and online transaction processing applications use different access methods and data structures for performance. Operating system tuning is dependent on the operating system being used.

Some database issues require assistance from the system administrator. This course addresses some of the generic issues.

## Types of Tuning

- **Proactive monitoring or tuning:** Examining performance statistics at a regular interval to identify whether the system behavior and resource usage has changed
- **Bottleneck elimination:** Identifying an overused resource in the system and determining potential fixes



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

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.

When tuning is left until the database is in production, it often becomes a reactive process to identify and fix the most important 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.

# Tuning Methodology

Tuning steps:

- Identify the scope of the problem (OS, database, and so on).
- Tune the following from the top down:
  - The design before tuning the application code
  - The code before tuning the instance
- Tune the area with the greatest potential benefit:
  - Identify the performance problem.
  - Analyze the problem, looking for skewed and tunable components.
  - Use appropriate tools to tune the components implicated.
- Stop tuning when the goal is met.



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

Oracle Corporation has developed a tuning methodology based on years of experience. The methodology presented in this course is also presented in the *Oracle Database Performance Tuning Guide*. This methodology is applied independent of the tools that you use. The Automatic Database Diagnostic Monitor (ADDM) tool follows this methodology automatically. The basic steps are the following:

- Check the OS statistics and the general machine health before tuning the instance to be sure that the problem is in the database instance. The problem could be in the OS or even in the application server.
- Tune from the top down. Start with the design, then the application, and then the instance. As an example, try to eliminate the full table scans causing the I/O contention before tuning the tablespace layout on disk. The design should use appropriate data structures for the application and load characteristics. For example, reverse key indexes may work well in a RAC environment to reduce hot blocks due to a sequential primary key, but it may also lead to a large number of blocks being shipped across the interconnects if every instance is inserting into the same table. The applications should avoid processes that require serialization through a single resource. A simple example is a single check number generator used by multiple processes. Tuning at the instance level is often limited by design and application choices. With existing applications, this step is often not available, since the design and code are not modifiable.

- Tune the area with the greatest potential benefit. The tuning methodology presented in this course is simple. Identify the biggest bottleneck and tune it. Repeat the action. The Oracle tuning tools use DB Time to identify problem areas. All tools have some way to identify the SQL statements, resource contention, or services that are taking the most time. Oracle Database 12c provides a time model and metrics to automate the process of identifying bottlenecks.

- Stop tuning when you meet your goal. This step implies that you set tuning goals.

This is a general approach to tuning the database instance and may require multiple passes.

Ideally someone with database tuning experience will be involved in the design and development from the beginning. This individual, for example, could suggest indexes to limit full table scans on frequently accessed tables, or removing indexes to allow parallel processing (especially important in larger servers).

From a practical perspective, tuning during the design and development phases of a project tends to be more top down. The tuning efforts during testing and production phases are often reactive and bottom up. In all phases, tuning depends on actual test cases because theoretical tuning does not know all the variables that can be present. After a problem area is suspected, or discovered, a test case is created and the area tuned as in all the examples given in this course. Tune the area that has the greatest potential benefit. Reduce the longest waits and the largest service times.

As you may notice, the techniques are very much the same, no matter what life cycle phase. A test case or actual application is run, the available diagnostic tools are applied, a solution is proposed and tested.

# Effective Tuning Goals

Effective tuning goals are:

- Specific
- Measurable
- Achievable
- Cost effective



**ORACLE**

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

The tuning goal is the elimination of a defined problem. Goals are also derived from related service level agreements (SLAs). The SLA is often a contractual or business requirement that must be met. The goal may be based on an SLA or a problem. For example, the SLA says that user response time to a particular request must be no more than 30 seconds. The problem is that the average response time is 25 seconds and increasing.

A tuning goal is that user response time to a particular request is 20 seconds.

Both tuning goals and SLAs must have three characteristics to be effective. They must be:

- Specific
- Measurable
- Achievable

“Make the instance run as fast as possible” is not specific. A specific goal would be “The month end report suite must complete in less than 4 hours.”

In addition, a goal must be cost effective. Tuning simply for the sake of tuning, or for elegance, is not cost effective.

A measurable goal has objective quantities that can be measured. There is no doubt whether the goal is being met when it is measurable. A goal that is specific is easily made measurable as well. The goal of “user response time to a request is 10 seconds” is easily stated, but is this for all user requests? Is it the average response time? How do you measure average response time? Having specific definitions for the words of your goal is essential. By restating the goal as “User response time to a particular request is 10 seconds or less,” you can objectively determine when the goal has been met.

Achievable goals are possible and within the control of the persons responsible for tuning.

The following are examples of unachievable goals for a typical DBA:

- When the goal is to tune the instance to create a high-performance application, but you are not allowed to change the SQL or the data structures, there is a limited amount of tuning that is possible.
- When the goal is to have a response time of 1 second, but the network latency between the server and the client is 2 seconds. Without a change to the network, a response time of 1 second is impossible.

Even these situations are not impossible to change in an absolute sense, but the DBA always has business constraints that limit the amount of money and resources that can be applied to the solution. So every goal should consider the cost to benefit. A goal that costs a great deal but solves a problem that has a marginal benefit, is best left undone.

You should always establish measurable tuning goals. Without a tuning goal, it is difficult to determine when you have performed enough tuning.

# General Tuning Session

Tuning sessions have the same procedure:

1. Define the problem and state the goal.
2. Collect current performance statistics.
3. Consider some common performance errors.
4. Build a trial solution.
5. Implement and measure the change.
6. Decide: “Did the solution meet the goal?”
  - No? Then go to step 3 and repeat.
  - Yes? Then create a new baseline.



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

When tuning, you focus on specific areas that offer the greatest return for your tuning effort. The steps are generic and apply to any performance monitoring tool. The recommended tuning methodology is as follows:

1. **Define the problem and state the goal:** This is the analysis step. The Oracle performance and diagnostic tools use a time model that can be used to quickly identify the problem areas. The information source could be users, database statistics, metrics, or database diagnostic reports. Be sure to collect accurate and factual data that corresponds to the problem. State the problem in terms that are measurable and directly related to the database operations. As an example, if the run time on the “XYZ” report is two times the baseline, the goal becomes: Make the run time on the “XYZ” report equal to or less than the baseline.
2. **Collect current statistics:** Examine the host system and the database statistics. Collect a full set of operating system and database statistics, and compare these with your baseline statistics. The baseline statistics are a set of statistics that are taken when the instance is running acceptably. Examine the differences to determine what has changed on the system. Did the “XYZ” report change? Did the data change? Is the session producing the report waiting on something?

3. **Consider common performance errors:** From your list of differences in the collected statistics, make a comparison with common performance errors. Determine whether one of these errors has occurred on your system.
4. **Build a trial solution:** Include a conceptual model in your solution. The purpose of this model is to assist you with the overall picture of the database. You are looking for answers to the following questions:
  - Why is the performance degraded?
  - How can you resolve the problem to meet your goal?
5. **Implement and measure the change:** After you have developed the trial solution, make the appropriate change. Make only one change at a time. If you make multiple changes at the same time, you will not know which change is effective. If the changes do not solve the problem, you would not know whether some changes helped and others hindered. Collect statistics to measure the change.
6. **“Did the solution meet the goal?”** Compare the current and the baseline sets. If you determine that more tuning is required, return to step 3 and repeat the process. If your solution meets the goal, make the current set of statistics the new baseline set. You met your goal! Stop tuning!

## Quiz

Every database tuning session has some common steps.

Which of the following is *not* a tuning step?

- a. Develop a trial solution.
- b. Capture statistics.
- c. Identify the problem.
- d. Take a backup.
- e. Test the solution and measure the change.



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

### Answer: d

Backups are not generally needed for a performance tuning session.

## Summary

In this lesson, you should have learned how to:

- Identify types of tuning
- Use a tuning methodology



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



# **Basic Tuning Diagnostics**

**ORACLE®**

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

# Objectives

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

- View the top wait events to determine the highest wait
- View the time model to diagnose performance issues
- Use dynamic performance views to view statistics and wait events
- Use Enterprise Manager monitoring
- Identify the key tuning components of alert logs
- Identify the key tuning components of user trace files



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

# Performance Tuning: Diagnostics

Diagnostic tools gather and format the following types of performance data:

- Cumulative statistics:
  - Wait events with time information
  - Time model
- Metrics: Statistic rates
- Sampled statistics: Active Session History, part of the Oracle Diagnostics Pack
  - Statistics by session
  - Statistics by SQL
  - Statistics by service
  - Other dimensions



**ORACLE**

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

Basic diagnostic tools display and format tuning data; some tools can also store performance tuning data. The Oracle Database server software captures information about its own operation. Three major types of data are collected: cumulative statistics, metrics, and sampled statistics.

Cumulative statistics are counts of and timing information for a variety of events that occur in the database server. Some are important, such as “buffer busy waits.” Many others have little impact on tuning. The raw counts have little meaning until the counts are compared over time. The events that collect the most time tend to be the most important. The statistics in Oracle Database 12c are correlated by the use of a time model. The time model statistics are based on a percentage of DB time, giving them a common basis for comparison.

Metrics are statistic counts per unit. The unit could be a time measure, such as seconds, or other measure such as transaction, session, allocated space, or event. Metrics provide a basis to proactively monitor performance. You can set thresholds on a metric causing an alert to be generated. For example, you can set thresholds for when the reads per millisecond exceed a value, the archive log area is 95% full, or a snapshot too old error occurs.

Sampled statistics, part of Active Session History (included in the Oracle Diagnostics Pack), enable you to look back in time. You can view the statistics that were gathered in the past, in various dimensions, even if you had not thought of specifying data collection for these beforehand.

# Performance Tuning: Features and Tools

- Automatic performance tuning features:
  - Automatic Workload Repository (AWR)
  - Automatic Database Diagnostic Monitor (ADDM)
  - Server-generated alerts
  - Advisors
  - Database Performance page of Enterprise Manager
- Additional tools:
  - V\$ performance views
  - Alert log
  - Trace files
  - Statspack



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

Oracle Database provides several tools that enable you to gather information about database performance, monitor performance, diagnose problems, and tune applications.

Most of the Oracle Database gathering and monitoring features are automatic and are managed by Oracle background processes. You can administer the features by using Oracle Enterprise Manager, or with APIs and views. Oracle Enterprise Manager Cloud Control is the recommended tool due to its ease of use, and numerous automated monitoring and diagnostic tools.

The automatic performance tuning features will be discussed in detail in this course. Enterprise Manager Cloud Control will be used during the course practices and V\$ views will be queried as appropriate.

The alert log is a chronological listing of database events and informational messages. The alert log can give important information about the operation of the database, areas that could be tuned, and reference information related to the tuning reports.

Background and user processes create trace files when certain events occur. These trace files can sometimes give you insight into performance problems, but are primarily for capturing error conditions and debugging information.

Statspack is a set of procedures and scripts supplied with all editions of the Oracle Database software.

# Tuning Objectives

- Minimizing response time
- Increasing throughput
- Increasing load capabilities
- Reducing recovery time



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

The objectives of tuning goals can be summarized as “Get more done in less time.” Depending on your environment, this translates into:

- Minimizing response time or reducing user wait time
- Increasing throughput, which means decreasing the time to perform a job or set of jobs
- Increasing load capabilities, which means allowing more tasks or releasing capacity for other tasks

In some environments, you make trade-offs. In high-volume online transaction processing (OLTP) environments, you may allow longer user response time to get more total transactions from many users. Studies have shown that in a web-based environment, user response time must be less than 7 seconds or the user goes somewhere else. In this case, everything else is subordinate to response time.

Business requirements affect tuning goals. Performance may be limited by safety concerns, as in the “Reducing recovery time” goal. In a business environment, where down time may be measured in hundreds or thousands of dollars per minute, the overhead of protecting the instance from failure and reducing recovery time is more important than the user response time. So tuning recovery must balance the ongoing overhead of additional disk writes to maintain redo log files and the goal of protecting the business from loss.

## Top Timed Events

The “Top 10 Foreground Events by Total Wait Time” section of the AWR report displays the ten timed events or operations that consumed the highest percentage of total database time (DB time) in the snapshot set.

Top 10 Foreground Events by Total Wait Time					
Event	Waits	Total Wait Time (sec)	Wait Avg(ms)	% DB time	Wait Class
buffer busy waits	45,078	1907.9	42	41.2	Concurrency
write complete waits	766	995	1299	21.5	Configuration
free buffer waits	4,947	597.9	121	12.9	Configuration
DB CPU		281.3		6.1	
log file sync	2,058	208	101	4.5	Commit
enq: TX - index contention	3,206	145.5	45	3.1	Concurrency
library cache lock	466	137.9	296	3.0	Concurrency
log file switch (checkpoint incomplete)	36	36.4	1012	.8	Configuration
latch: cache buffers chains	12,965	30.2	2	.7	Concurrency
db file sequential read	464,369	23.2	0	.5	User I/O



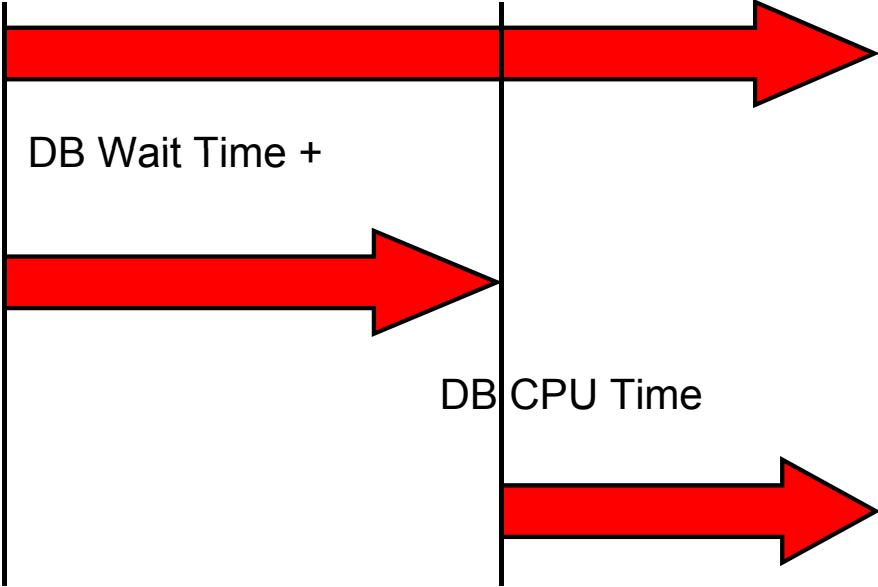
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The “Top 10 Foreground Events by Total Wait Time” section is a good place to start when tuning. This section is available in AWR and Statspack reports. At a glance, you can see the top timed events. The top timed events will always have some values, even when there is no performance problem in the database.

In this example, “free buffer waits” and “buffer busy waits” are consuming more than 50% of DB time when combined. The events reported here provide a direction for further investigation. In this case, if the instance is not performing as well as expected, the top timed events indicate that there may be a problem in the database buffer cache.

## DB Time

DB Time =



ORACLE

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

Tuning is not just about reducing waits. It aims at improving end-user response time and/or minimizing the average resources used by each request. Sometimes these go together, but in other cases, there is a trade-off (for example, with a parallel query). In general, you can say that tuning is the avoidance of consuming or holding resources in a wasteful manner.

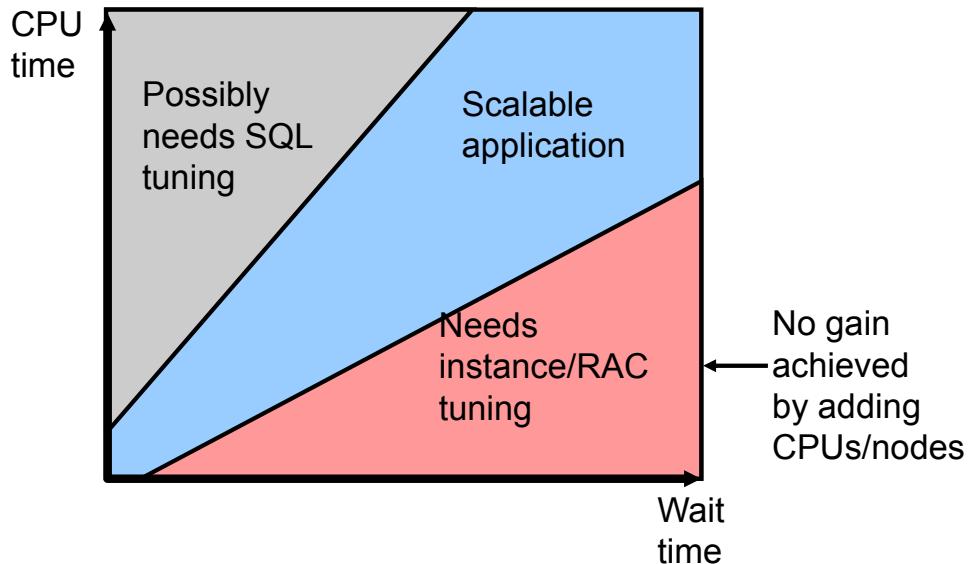
Any request to the database is composed of two distinct segments: a wait time (DB wait time) and a service time (DB CPU time). The wait time is the sum of all the waits for various database instance resources. The CPU time is the sum of the time that is spent actually working on the request. These times are not necessarily composed of one wait and one block of CPU time. Often, processes will wait a short time for a DB resource and then run briefly on the CPU, and do this repeatedly.

Tuning consists of reducing or eliminating the wait time and reducing the CPU time. This definition applies to any application type, online transaction processing (OLTP), or data warehouse (DW).

**Note:** A very busy system shows longer DB CPU times and this can inflate other times. This is because the DB CPU time includes the time that a process is waiting for the CPU. Check the OS load averages. If the load average is consistently more than 2 times the number of CPUs, the system is waiting on CPU.

# CPU and Wait Time Tuning Dimensions

$$\text{DB time} = \text{DB CPU time} + \text{DB wait time}$$



ORACLE

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

When tuning your system, it is important that you compare the CPU time with the wait time of your system. By comparing CPU time with wait time, you can determine how much of the response time is spent on useful work and how much on waiting for resources potentially held by other processes. As a general rule, systems where CPU time is dominant usually need less tuning than the ones where wait time is dominant. However heavy CPU usage can be caused by badly written SQL statements.

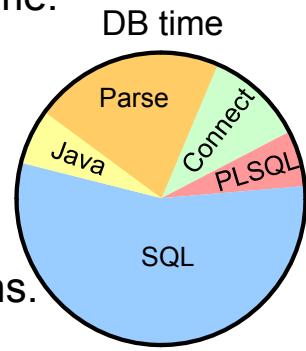
The proportion of wait time to CPU time always tends to increase as the load on the system increases; steep increases in wait time are a sign of contention and must be addressed for good scalability.

When contention is evidenced by increased wait time, adding more CPUs to a node, or nodes to a cluster, would provide very limited benefit. Conversely, a system where the proportion of CPU time does not decrease significantly as load increases can scale better, and would most likely benefit from adding CPUs or Real Application Clusters (RAC) instances.

**Note:** Automatic Workload Repository (AWR) and Statspack reports display CPU time in the Top 10 Foreground Events section, if the CPU time portion is among the top events.

## Time Model: Overview

- The time model is a set of statistics that give an overview of where time is spent inside the Oracle database.
- All statistics use the same dimension: time.
- The statistics are accessible through:
  - `V$SYS_TIME_MODEL`
  - `V$SESS_TIME_MODEL`
- DB time represents the total time spent in database calls by user sessions.
- A tuning goal is to reduce DB time.
- Using DB time, you can gauge the performance impact of any entity of the database.



**ORACLE**

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

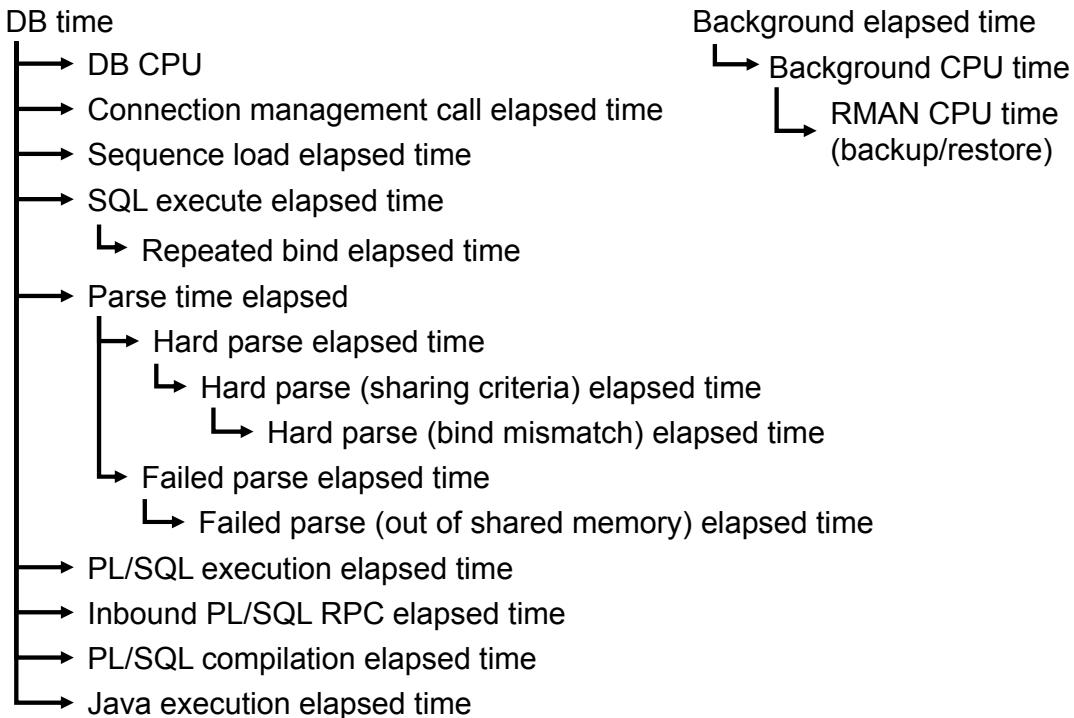
There are many components involved in tuning an Oracle database system and each has its own set of statistics. How can you measure the expected benefit from a tuning action on the overall system? For example, would the overall performance improve if you move memory from the buffer cache to the shared pool? When you look at the system as a whole, time is the only common ruler for comparison across components. In the Oracle database server, most of the advisories report their findings in time. Also statistics called “time model statistics” appear as the `V$SYS_TIME_MODEL` and `V$SESS_TIME_MODEL` performance views. This instrumentation helps the Oracle database server to identify quantitative effects on the database operations.

DB time is the most important of the time model statistics. This statistic represents the total time spent in database calls by user sessions and indicates the total instance workload. It is the sum of the CPU and wait times of all sessions not waiting on idle wait events (non-idle user sessions).

The objective for tuning an Oracle database system could be stated as reducing the time that users spend in performing some action on the database, or simply reducing DB time.

Other time model statistics provide quantitative effects (in time) on specific actions, such as logon operations, hard and soft parses, PL/SQL execution, and Java execution.

# Time Model Statistics Hierarchy



**ORACLE**

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

The relationships between the time model statistics are listed in the slide. They form two trees: DB time and background elapsed time. The time reported by a child in the tree is contained within the parent in the tree.

- **DB time:** Amount of elapsed time (in microseconds) spent performing database user-level calls. This does not include the time spent on instance background processes such as PMON. DB time is measured cumulatively from the time that the instance was started. Because DB time is calculated by combining the times from all non-idle user sessions, it is possible for the DB time to exceed the actual time elapsed since the instance started. For example, an instance that has been running for 30 minutes could have four active user sessions whose cumulative DB time is approximately 120 minutes.
- **DB CPU:** Amount of CPU time (in microseconds) spent on database user-level calls. This time includes processes on the run queue. Maximum DB CPU is elapsed time times the number of CPUs.
- **Sequence load elapsed time:** Amount of elapsed time spent getting the next sequence number from the data dictionary. If a sequence is cached, this is the amount of time spent replenishing the cache when it runs out. No time is charged when a sequence number is found in the cache. For non-cached sequences, some time will be charged for every NEXTVAL call.

- **Parse time elapsed:** Amount of elapsed time spent parsing SQL statements. It includes both soft and hard parse time.
- **Hard parse elapsed time:** Amount of elapsed time spent hard-parsing SQL statements
- **SQL execute elapsed time:** Amount of elapsed time SQL statements are executing.  
Note that for SELECT statements, this also includes the amount of time spent performing fetches of query results.
- **Connection management call elapsed time:** Amount of elapsed time spent performing session connect and disconnect calls
- **Failed parse elapsed time:** Amount of time spent performing SQL parses that ultimately fail with some parse error.
- **Failed parse (out of shared memory) elapsed time:** Amount of time spent performing SQL parses that fail with out of shared memory error
- **Hard parse (sharing criteria) elapsed time:** Amount of elapsed time spent performing SQL hard parses when the hard parse resulted from not being able to share an existing cursor in the SQL cache
- **Hard parse (bind mismatch) elapsed time:** Amount of elapsed time spent performing SQL hard parses when the hard parse resulted from bind type or bind size mismatch with an existing cursor in the SQL cache
- **PL/SQL execution elapsed time:** Amount of elapsed time spent running the PL/SQL interpreter. This does not include time spent recursively executing or parsing SQL statements or time spent recursively executing the Java Virtual Machine.
- **PL/SQL compilation elapsed time:** Amount of elapsed time spent running the PL/SQL compiler
- **Inbound PL/SQL RPC elapsed time:** Time inbound PL/SQL remote procedure calls (RPCs) have spent executing. It includes all time spent recursively executing SQL and Java, and therefore, is not easily related to PL/SQL execution elapsed time.
- **Java execution elapsed time:** Amount of elapsed time spent running the Java VM. This does not include time spent recursively executing or parsing SQL statements or time spent recursively executing PL/SQL.
- **Repeated bind elapsed time:** Elapsed time spent on rebinding
- **Background CPU time:** Amount of CPU time (in microseconds) consumed by database background processes
- **Background elapsed time:** Total time spent in the database by background sessions (CPU time and non-idle wait time)
- **RMAN CPU time (backup/restore):** CPU time spent by RMAN backup and restore operations

## Time Model: Example

### Time Model Statistics

- Total time in database user-calls (DB Time): 24321.8s
- Statistics including the word "background" measure background process time,
- Ordered by % or DB time desc, Statistic name

Statistic Name	Time (s)	% of DB Time
sql execute elapsed time	19,564.83	80.44
connection management call elapsed time	2,901.18	11.93
DB CPU	2,394.16	9.84
PL/SQL execution elapsed time	807.35	3.32
parse time elapsed	88.90	0.37
sequence load elapsed time	81.51	0.34
hard parse elapsed time	34.55	0.14
PL/SQL compilation elapsed time	2.36	0.01
hard parse (sharing criteria) elapsed time	1.61	0.01
repeated bind elapsed time	0.16	0.00
hard parse (bind mismatch) elapsed time	0.00	0.00
failed parse elapsed time	0.00	0.00
DB time	24,321.80	
background elapsed time	3,723.82	
background cpu time	44.57	

[Back to Wait Events Statistics](#)

[Back to Top](#)

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

The example shown is from an AWR report. The time model information is also available in the Statspack report. The statistics are ordered by the % DB Time value, so the area that is taking the most time and its children are first on the list. In this example, “sql execute elapsed time” is at the top.

**Note:** The sum of the “% of DB Time” of the individual statistics is more than 100%.

## Quiz

The time model measurements on your database show that there are significant waits. Waits are taking more time than CPU time. This indicates that the application is scalable, and adding more CPUs will help performance.

- a. True
- b. False



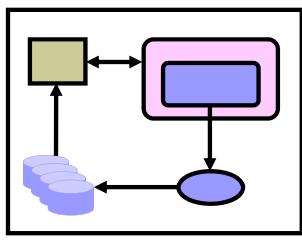
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

### Answer: b

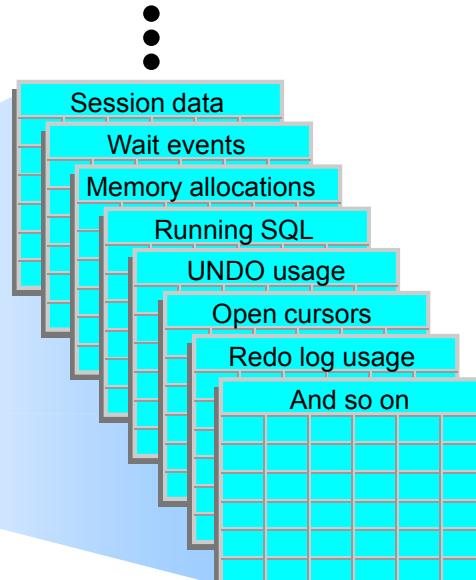
When the time model is dominated by waits, the application is not scalable. Adding more CPUs will not help performance.

# Dynamic Performance Views

Dynamic performance views provide access to information about changing states and conditions in the instance.



Oracle instance



**ORACLE**

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

The Oracle database server maintains a dynamic set of data about the operation and performance of the instance. These dynamic performance views are based on virtual tables that are built from memory structures inside the database server. That is, they are not conventional tables that reside in a database. `V$` views externalize metadata contained in memory structures of an Oracle instance. Some `V$` views can show data before a database is mounted or open. The `V$FIXED_TABLE` view lists all the dynamic views.

Dynamic performance views include the raw information used by AWR and Statspack and detail information about, but not limited to:

- Sessions
- Wait events
- Locks
- Backup status
- Memory usage and allocation
- System and session parameters
- SQL execution
- Statistics and metrics

**Note:** The `DICT` and `DICT_COLUMNS` views also contain the names of these dynamic performance views.

## Dynamic Performance Views: Usage Examples

**a**

```
SQL> SELECT sql_text, executions
  2  FROM v$sqlstats
  3  WHERE cpu_time > 200000;
```

**b**

```
SQL> SELECT * FROM v$session
  2  WHERE machine = 'ED9P1' and
  3  logon_time > SYSDATE - 1;
```

**c**

```
SQL> SELECT sid, ctime
  2  FROM v$lock WHERE block > 0;
```

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

Enterprise Manager uses dynamic performance views, and DBAs can also query these views as needed. The three examples shown in the slide answer the following questions:

- What are the SQL statements and their associated number of executions where the CPU time consumed is greater than 200,000 microseconds?
- What sessions logged in from the ED9P1 computer within the last day?
- What are the session IDs of any sessions that are currently holding a lock that is blocking another user, and how long has that lock been held? The value of `BLOCK` may be 1 or 0; 1 indicates that this session is the blocker.

Dynamic performance views are built on memory structures that hold the statistics, and allow you to view many of the statistics that are used in performance tuning. Most contain information about a specific component of the instance. In this course, you use the information from the dynamic performance views indirectly to tune specific components. For a complete list of the various dynamic performance views, refer to *Oracle Database Reference*.

Most of the dynamic performance views are not frequently used directly by the DBA. The Automatic Workload Repository (AWR) and Statspack tools summarize and present the statistics in a much more usable format. At times, it is helpful to know that these views exist to investigate details.

# Dynamic Performance Views: Considerations

- These views are owned by SYS.
- Different views are available at different times:
  - The instance has been started.
  - The database is mounted.
  - The database is open.
- You can query V\$FIXED\_TABLE to see all the view names.
- These views are often referred to as “v-dollar views.”
- All reads on these views are current reads.



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

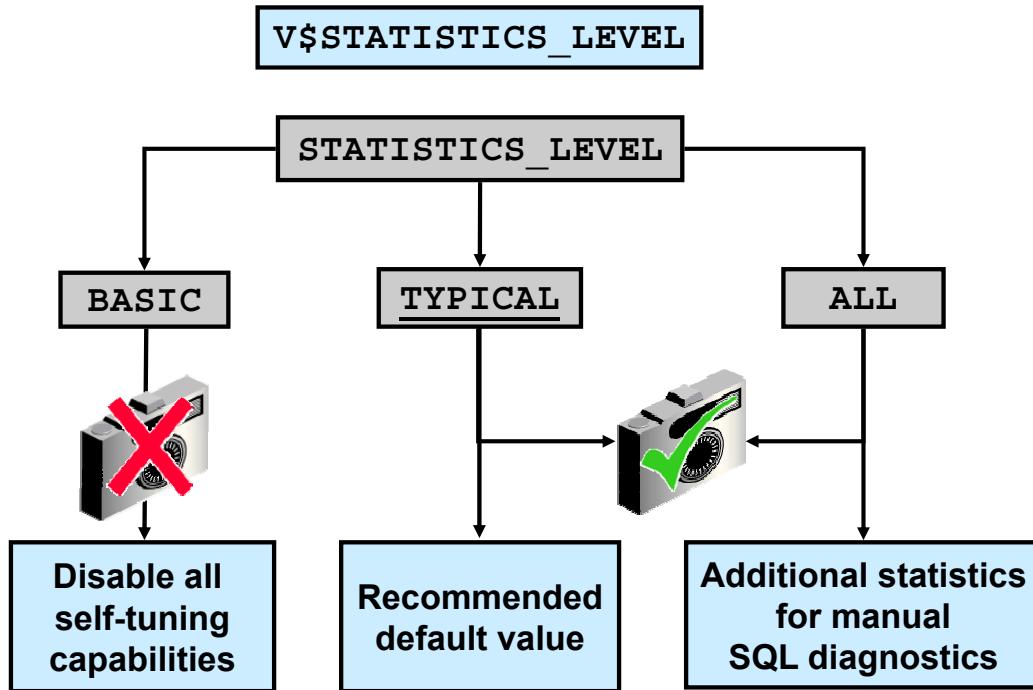
Some dynamic views contain data that is not applicable to all states of a database instance. For example, if an instance has just been started, but no database is mounted, you can query V\$BGPROCESS to see the list of background processes that are running. If you query V\$DATAFILE to see the status of database data files, then you receive an error ORA-01507: database not mounted because it is not until the database is mounted that the server reads the control file to find out about the data files associated with a database.

These views are based on memory structures and the data in them is cumulative since startup. The data is reset at startup. Therefore, you cannot know how many times X (for example, “physical reads”) happened between T1 and T2 by calculating the “delta” of its counter if the instance had been restarted during that time period.

Because all the reads on these views are current reads, there is no locking mechanism on these views, so there is no guarantee that the data would be read consistent. You occasionally see anomalies in the statistics when one or more tables related to a particular statistic were updated, but not all the tables had completed the update when the select occurred.

The SELECT\_CATALOG\_ROLE is the minimum predefined role that can be granted to allow a user to query the V\$ views. A good security practice is to create a role modeled after the SELECT\_CATALOG\_ROLE, with only the minimum required privileges.

# Statistic Levels



**ORACLE**

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

You determine the level of statistics collection on the database by setting the value of the STATISTICS\_LEVEL parameter. The values for this parameter are:

- **BASIC:** No advisory or other statistical data is collected. You can manually set other statistic collection parameters such as TIMED\_STATISTICS and DB\_CACHE\_ADVICE. Many of the statistics required for a performance baseline are not collected. Oracle strongly recommends that you do not disable statistic gathering.
- **TYPICAL:** This is the default value. Data is collected for segment-level statistics, timed statistics, and all advisories. The value of other statistic collection parameters is overridden.
- **ALL:** Collection is made of all the TYPICAL level data, the timed operating system statistics, and the row source execution statistics. The value of other statistic collection parameters is overridden.

Query V\$STATISTICS\_LEVEL to determine which other parameters are affected by the STATISTICS\_LEVEL parameter.

```
SQL> SELECT statistics_name, activation_level
  2  FROM v$statistics_level
  3  ORDER BY 2;
```

STATISTICS_NAME	ACTIVAT
Plan Execution Statistics	ALL
Timed OS Statistics	ALL
Timed Statistics	TYPICAL
Segment Level Statistics	TYPICAL
PGA Advice	TYPICAL
Shared Pool Advice	TYPICAL
Modification Monitoring	TYPICAL
Longops Statistics	TYPICAL
Bind Data Capture	TYPICAL
Ultrafast Latch Statistics	TYPICAL
Threshold-based Alerts	TYPICAL
Global Cache Statistics	TYPICAL
Active Session History	TYPICAL
Undo Advisor, Alerts and Fast Ramp up	TYPICAL
Streams Pool Advice	TYPICAL
Time Model Events	TYPICAL
Plan Execution Sampling	TYPICAL
Automated Maintenance Tasks	TYPICAL
SQL Monitoring	TYPICAL
Adaptive Thresholds Enabled	TYPICAL
V\$IOSTAT_* statistics	TYPICAL
Buffer Cache Advice	TYPICAL
MTTR Advice	TYPICAL

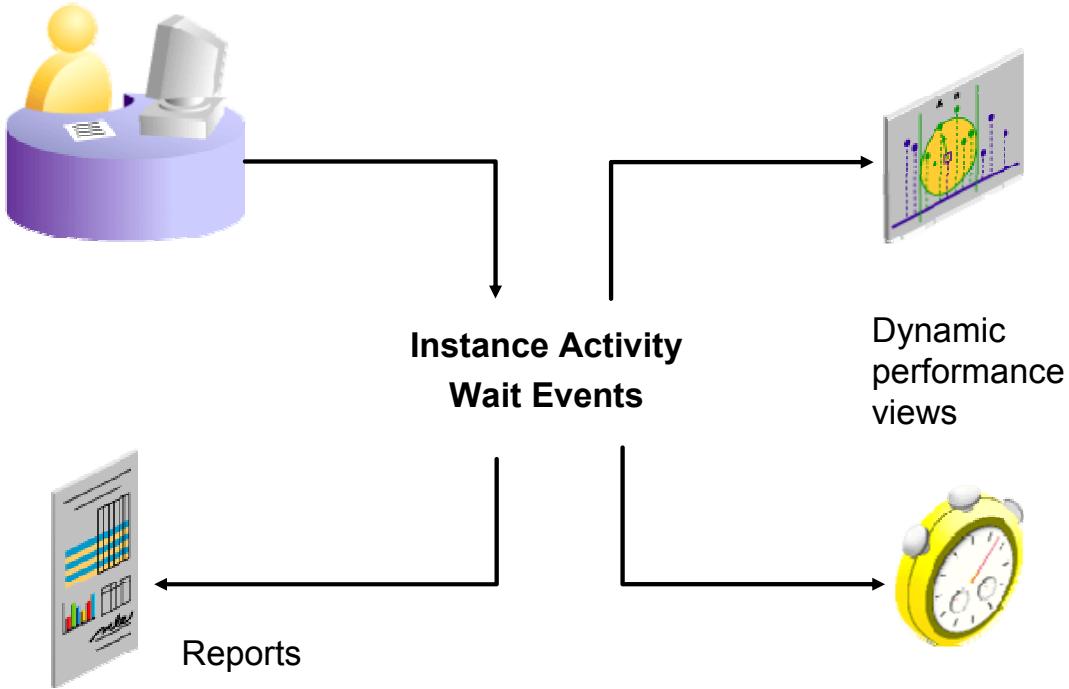
The following statistic parameters can also be set individually **TIMED\_STATISTICS**: Set to TRUE to collect statistics related to time.

- **DB\_CACHE\_ADVICE**: Accepts the following values:
  - OFF: No statistics collected and no memory used
  - READY: No statistics collected, but memory is allocated. Setting **DB\_CACHE\_ADVICE** to READY before setting it to ON prevents memory errors when collecting statistics on buffer cache utilization.
  - ON: Statistics collected and memory allocated. Changing the status of **DB\_CACHE\_ADVICE** from OFF to ON can raise an error if the required memory is not available.
- **TIMED\_OS\_STATISTICS**: Specifies the interval (in seconds) at which an Oracle instance collects operating system statistics when a request is made from the client to the server or when a request completes

When **STATISTICS\_LEVEL** is modified by **ALTER SESSION**, the following advisories or statistics are turned on or off in the local session only. Their systemwide state is not changed.

- Timed statistics
- Timed OS statistics
- Plan execution statistics

# Instance Activity and Wait Event Statistics



ORACLE®

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

The Oracle Database server maintains several metrics that reflect internal activity within an instance. These metrics are exposed to DBAs via dynamic performance views. Many of these views reflect a set of statistical counters that are initialized to 0 at instance startup and they are incremented until the instance is shut down.

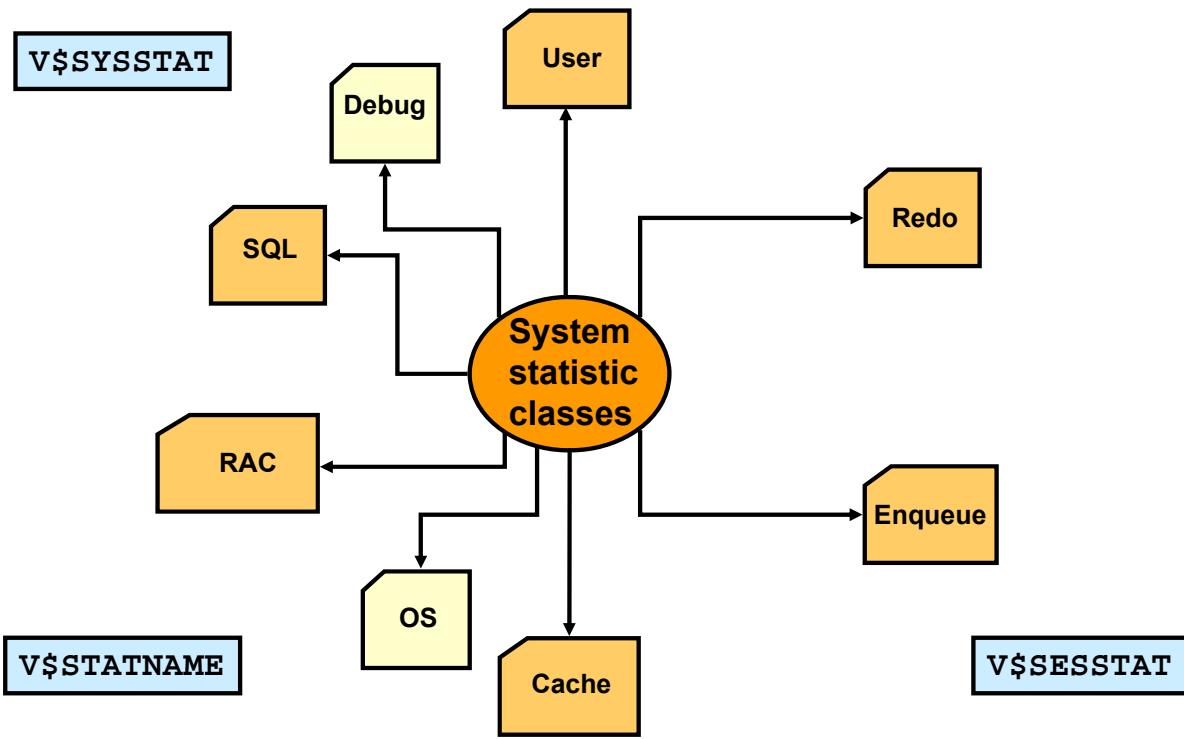
Instance activity and wait event statistics are the two classes of metrics that generally drive the performance tuning investigation process.

Instance activity statistics are provided by developers to help debug various software features. They may or may not relate directly to wait events or other metrics. Some of these statistics are useful for performance diagnostics such as “parse time cpu,” “physical reads,” and “user commits.” A full list of instance activity statistics can be seen by querying V\$STATNAME, with accumulated instance level values available via V\$SYSSTAT. There are approximately 850 statistics in 8 classes captured by the database. For more information about instance activity statistics, see *Oracle Database Reference 12c Release 1, Appendix E Statistics Descriptions*. Not all statistic names are documented.

Wait events are counters that are incremented by a server process or thread to indicate that it had to wait for a resource to become available or some other event to occur before being able to continue processing. Wait event statistics reveal various symptoms of problems that might be affecting performance, such as latch contention, buffer contention, and I/O contention. Remember that these are only symptoms of problems, not the actual causes. The full list of wait events can be found in the `V$EVENT_NAME` view, with accumulated instance level values exposed via `V$SYSTEM_EVENT`. There are more than 1500 wait events, in 12 classes. For more information about wait events, see *Oracle Database Reference 12c Release 1, Appendix C Oracle Wait Events*.

The database metrics provide the raw data that is used to determine what needs to be tuned, and whether the tuning exercise met the goal. Both Statspack and AWR take snapshots of this data, make calculations based on those snapshots, and provide reports of the derived information. AWR goes a step further than Statspack and makes recommendations through the Automatic Database Diagnostic Monitor (ADDM). AWR includes additional information that is not found in the default Statspack report, and it can present its report in HTML format.

# System Statistic Classes



**ORACLE®**

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

This slide describes the statistic classes stored in the `V$SESSTAT` and `V$SYSSTAT` views. It is necessary to create classes for those statistics because there are a large number of them. Each statistic may belong to one or more classes. The `CLASS` column for each statistic contains a number representing one or more statistic classes. The following class numbers are additive:

- 1, User
- 2, Redo
- 4, Enqueue
- 8, Cache
- 16, OS
- 32, Real Application Clusters
- 64, SQL
- 128, Debug

For example, a class value of 72 represents a statistic that relates to SQL statements (64) and caching (8).

**Note:** Some statistics are populated only if the `TIMED_STATISTICS` initialization parameter is set to `TRUE`.

# Displaying Statistics

Instance activity statistics are collected for:

- Sessions
  - All sessions: V\$SESSTAT
  - Current session: V\$MYSTAT
- Services: V\$SERVICE\_STATS
- System: V\$SYSSTAT



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

The server displays a summary of all calculated instance activity statistics at the system level in the V\$SYSSTAT view. You can query this view to find cumulative totals since the instance started. At all levels there is a statistics identifier that can be joined to the V\$STATNAME table.

## System-Level Statistics

NAME	CLASS	VALUE
logons cumulative	1	6393
logons current	1	10
opened cursors cumulative	1	101298
table scans (short tables)	64	6943
table scans (long tables)	64	344
redo entries	2	1126226
redo size	2	816992940

The results shown are only a partial display of the output.

All the statistics views listed include the NAME, CLASS, and VALUE columns. The service-level view includes the SERVICE\_NAME, and the session-level view includes the SID (session identifier) column. These allow you to join the V\$SERVICE\_NAME and V\$SESSION views to the service-level and session-level views.

## Service-Level Statistics

Service data is cumulative from instance startup. The service name allows collection of statistics by a connection service name. This is very useful for performance monitoring by application. Every user that connects uses a specific service name per application.

### Example

There are always two services defined: SYS\$BACKGROUND and SYS\$USERS. Up to 1019 additional services may be created based on the SERVICE\_NAMES parameter or set with the DBMS\_SERVICE package. Service data is cumulative from the instance startup.

The Oracle Database server displays all calculated service statistics in the V\$SERVICE\_STAT view. You can query this view to find service cumulative totals since the instance started.

```
SQL> SELECT service_name, stat_name, value
  2  FROM v$service_stats;
```

SERVICE_NAME	STAT_NAME	VALUE
SYS\$USERS	user calls	6977
SERV1	user calls	532
SYS\$BACKGROUND	user calls	0
orcl.oracle.com	user calls	18948
orclXDB	user calls	0
SYS\$USERS	DB time	84608280
SERV1	DB time	222965588
SYS\$BACKGROUND	DB time	0
orcl.oracle.com	DB time	55877745
orclXDB	DB time	0

## Session-Related Statistics

You can display current session information for each user logged on with the V\$SESSION view. The Oracle Database server displays all calculated session statistics in the V\$SESSTAT view and the statistics for the current session are shown in V\$MYSTAT.

### Example

Determine the sessions that consume more than 30,000 bytes of PGA memory.

```
SQL> SELECT username, name, value
  2  FROM v$statname n, v$session s, v$sesstat t
  3  WHERE s.sid=t.sid
  4  AND  n.statistic#=t.statistic#
  5  AND  s.type='USER'
  6  AND  s.username is not null
  7  AND  n.name='session pga memory'
  8  AND  t.value > 30000;
```

USERNAME	NAME	VALUE
SYSTEM	session pga memory	468816

# Displaying SGA Statistics

NAME	BYTES	RES	CON_ID
Fixed SGA Size	2291280	No	0
Redo Buffers	7303168	No	0
Buffer Cache Size	339738624	Yes	0
Shared Pool Size	301989888	Yes	0
Large Pool Size	8388608	Yes	0
Java Pool Size	4194304	Yes	0
Streams Pool Size	0	Yes	0
Shared IO Pool Size	29360128	Yes	0
Data Transfer Cache Size	0	Yes	0
Granule Size	4194304	No	0
Maximum SGA Size	663908352	No	0
Startup overhead in Shared Pool	124364176	No	0
Free SGA Memory Available	0		0

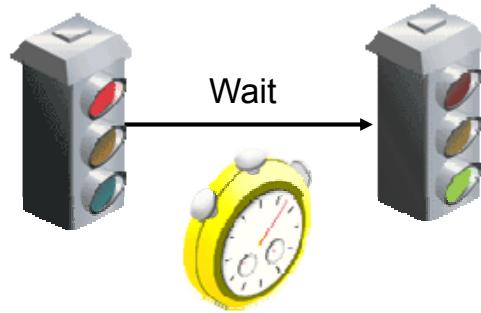


Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The V\$SGAINFO view provides the current size of the SGA components, the granule size, and free memory. A brief summary is presented in the V\$SGA view. All calculated memory statistics are displayed in the V\$SGASTAT view. You can query this view to find cumulative totals of detailed SGA usage since the instance started.

## Wait Events

- A collection of wait events provides information about the sessions that had to wait or must wait for different reasons.
- These events are listed in the `V$EVENT_NAME` view, which has the following columns:
  - `EVENT#`
  - `NAME`
  - `PARAMETER1`
  - `PARAMETER2`
  - `PARAMETER3`



ORACLE®

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

All wait events are named in the `V$EVENT_NAME` view, including:

- Free buffer waits
- Latch free
- Buffer busy waits
- Db file sequential read
- Db file scattered read
- Db file parallel write
- Undo segment tx slot
- Undo segment extension

Each event is assigned to a wait class. This assignment is shown in the `V$EVENT_NAME` view. Each event can have additional parameters returned with the event; columns `PARAMETER1` through `PARAMETER3` show the meaning of these parameters.

**Note:** Time information columns for wait events are populated only if the `TIMED_STATISTICS` initialization parameter is set to TRUE.

## Using the V\$EVENT\_NAME View

```
SQL> SELECT name, parameter1, parameter2, parameter3
  2  FROM v$event_name;
```

NAME	PARAMETER1	PARAMETER2	PARAMETER3
PL/SQL lock timer	duration		
alter system set mts_dispatcher	waited		
buffer busy waits	file#	block#	id
library cache pin	handle	addr	pin address 0*mode+name
log buffer space			
log file switch (checkpoint incomplete)			
transaction	undo	seg#	wrap#
...			count
1118 rows selected.			



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

A wait event may have up to three parameters. The meaning of each of these parameters is listed in the PARAMETER $n$  columns.

### The Buffer Busy Waits Event

The “buffer busy waits” event records the waits that are required for a buffer to become available. These waits indicate that there are some buffers in the buffer cache that multiple processes are attempting to access concurrently.

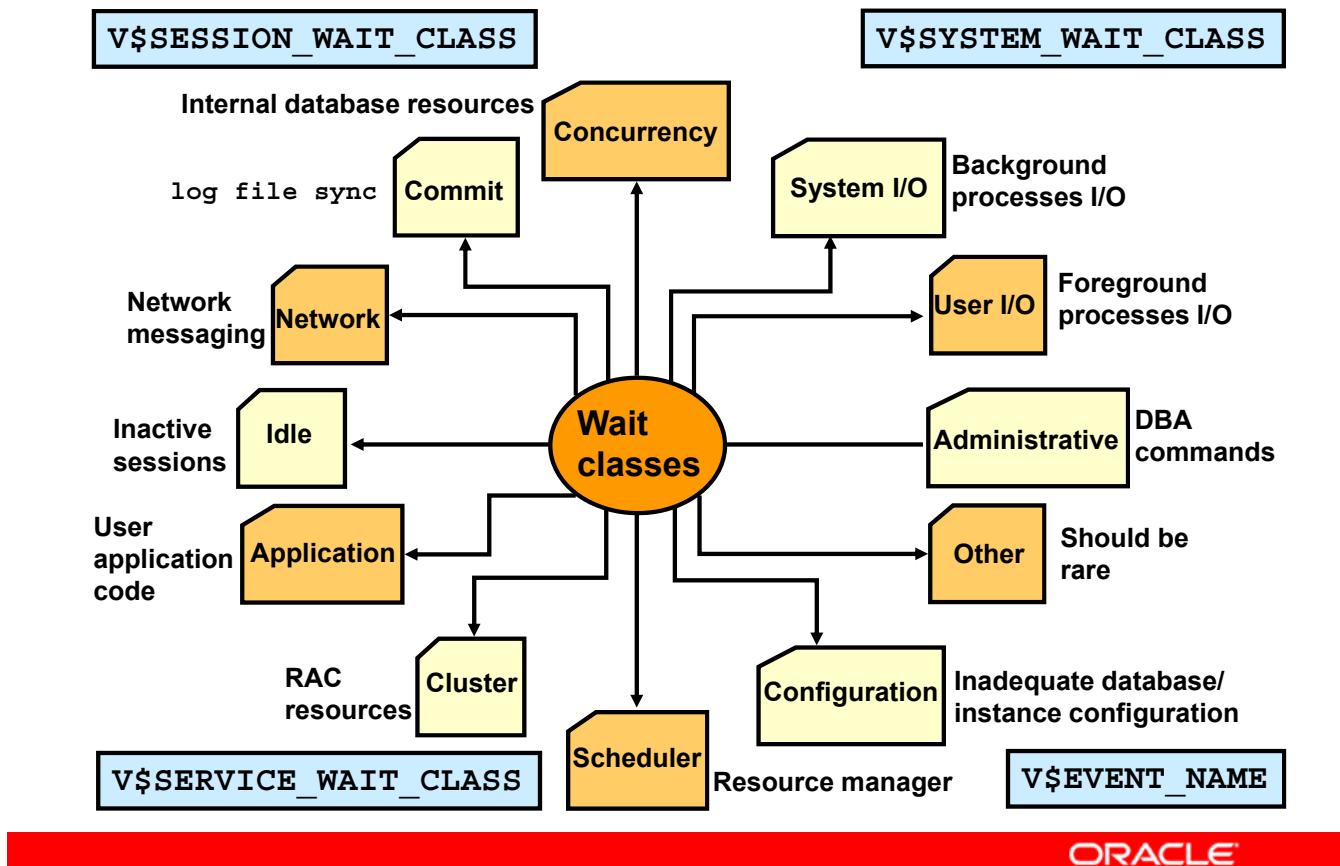
This event is accompanied by three parameters:

- FILE# and BLOCK#: These parameters identify the block number in the data file that is identified by the file number for the block for which the server needs to wait.
- ID: The “buffer busy waits” event is called from different places in the session. Each place in the kernel points to a different reason. ID refers to the place in the session calling this event.

### The Log File Switch (Checkpoint Incomplete) Event

The “log file switch (checkpoint incomplete)” event records the waits 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. This event has no parameter.

# Wait Classes



ORACLE

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

The many different wait events possible in Oracle Database are categorized into wait classes on the basis of the solutions related to that event. Each event is related to only one wait class. This enables high-level analysis of the wait events. For example, exclusive transaction (TX) locks are generally an application-level issue and segment space management (HW) locks are generally a configuration issue. The following are the most commonly occurring wait classes:

- **Application:** Lock waits caused by row-level locking or explicit lock commands
- **Administration:** DBA commands that cause other users to wait, such as an index rebuild
- **Commit:** Waits for redo log write confirmation after a commit
- **Concurrency:** Concurrent parsing and buffer cache latch and lock contention
- **Configuration:** Undersized log buffer space, log file sizes, buffer cache size, shared pool size, or HW enqueue contention
- **User I/O:** Waits for blocks to be read off disk by foreground processes
- **Network Communications:** Waits for data to be sent over the network
- **Idle:** Wait events related to inactive sessions such as “SQL\*Net message from client”

**Note:** The Other class contains waits that should not typically occur on a system (for example, “wait for EMON to spawn”).

# Displaying Wait Event Statistics

- Wait event statistics levels:
  - System
  - Service
  - Session
- Wait event statistics columns vary by view.

V\$SYSTEM\_EVENT

V\$SERVICE\_EVENT

V\$SESSION\_EVENT

EVENT	X	X	X
TOTAL_WAITS	X	X	X
TOTAL_TIMEOUTS	X	X	X
TIME_WAITED	X	X	X
AVERAGE_WAIT	X	X	X
TIME_WAITED_MICRO	X	X	X
EVENT_ID	X	X	X
TOTAL_WAIT_FG	X	X	X
TOTAL_TIMEOUTS_FG	X	X	X
TIME_WAITED_FG	X	X	X
AVERAGE_WAIT_FG	X	X	X
TIME_WAITED_MICRO_FG	X	X	X
SID		X	X
SERVICE_NAME	X	X	X
SERVICE_NAME_HASH	X	X	X
WAIT_CLASS_ID	X	X	X
WAIT_CLASS#	X	X	X
WAIT_CLASS	X	X	X

ORACLE

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

All wait events are cataloged in the V\$EVENT\_NAME view.

Cumulative wait event statistics for all sessions are stored in V\$SYSTEM\_EVENT, which shows the total waits for a particular event since instance startup. V\$SERVICE\_EVENT shows the cumulative wait event statistics for each service. V\$SESSION\_EVENT shows cumulative wait event statistics for each session.

Most of the wait event statistics are the same for each view. The differences are shown above. The V\$SYSTEM\_EVENT view includes a breakout of statistics for foreground processes. V\$SERVICE\_EVENT includes the service, and V\$SESSION includes the session identifier.

**Note:** V\$SERVICE\_EVENT does not include the three wait class columns, but they are easily obtained by joining to V\$EVENT\_NAME.

When you are troubleshooting, you need to know if a process has waited for any resource.

## Service Wait Events Statistics

The V\$SERVICE\_EVENT view shows by service, the total waits for a particular event since instance startup. The V\$SERVICE\_WAIT\_CLASS view aggregates the waits by service and wait class.

```
SQL> SELECT service_name, event, average_wait
  2  FROM v$service_event
  3  WHERE time_waited > 0;
```

SERVICE_NAME	EVENT	AVERAGE_WAIT
SERV1	log file sync	3
SERV1	db file sequential read	1
orcl.oracle.com	log file sync	1
orcl.oracle.com	db file sequential read	1
orcl.oracle.com	db file scattered read	2
orcl.oracle.com	latch: shared pool	1
orcl.oracle.com	latch: library cache	4

## Session Wait Events Statistics

The V\$SESSION\_EVENT view shows by session the total waits for a particular event since instance startup. The V\$SESSION\_WAIT view lists the resources or events for which active sessions are waiting. V\$SESSION also includes the current wait information.

When you are troubleshooting, you need to know whether a process has waited for any resource. The structure of V\$SESSION\_WAIT makes it easy to check in real time whether any sessions are waiting and why.

```
SQL> SELECT sid, event
  2  FROM v$session_wait
  3  WHERE wait_time = 0;
```

SID	EVENT
1	pmon timer
2	rdbms ipc message
3	rdbms ipc message
9	rdbms ipc message
16	rdbms ipc message
17	rdbms ipc message
10	rdbms ipc message
5	smon timer
8	rows selected.

You can then investigate further to see whether such waits occur frequently and whether they can be correlated with other phenomena, such as the use of particular modules.

**Note:** The wait events shown above are idle wait class events that always appear. They do not indicate a problem. There are more than 60 such idle events and they belong to the "Idle" wait class (wait class number 6).

## Commonly Observed Wait Events

Wait Event	Area
Buffer busy waits	Buffer cache, DBWR
Free buffer waits	Buffer cache, DBWR, I/O
Db file scattered read, Db file sequential read	I/O, SQL Tuning
Enqueue waits (enq:)	Locks
Library cache waits	Mutexes/Latches
Log buffer space	Log buffer I/O
Log file sync	Over-commit, I/O



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

The slide shows a list of wait events and component areas that could be the source of these waits. The internal definitions of these wait events can change from version to version, and may cause other events to become more common. For a description of wait events for a particular database version, see the *Oracle Database Reference* associated with the database version. Not all wait events are documented.

The statistics for wait events at the system level are most useful when they can be correlated to a time period with known activity. The `V$SYSTEM_EVENT` view gives only the totals since instance startup. A practical method is to capture the statistics into a table with a time and then capture the statistics again later. This will allow you to compare the change in the statistic values in a particular time period. This removes the statistics related to instance startup, and narrows the set of possible issues. This is the premise for the Active Session History described in the lesson “Using AWR-Based Tools.”

Statspack and AWR follow this method with the use of snapshots to capture statistics, wait events, and metrics.

## Using the V\$SESSION\_WAIT View

```
SQL> SELECT sid, seq#, event, wait_time, state
  2  FROM v$session_wait;
```

SID	SEQ#	EVENT	WAIT TIME	STATE
1	1284	pmon timer	0	WAITING
2	1697	rdbms ipc message	0	WAITING
3	183	rdbms ipc message	0	WAITING
4	4688	rdbms ipc message	0	WAITING
5	114	smon timer	0	WAITING
6	14	SQL*Net message from client	-1	WAITED SHORT TIME



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

This view lists the resources or events for which sessions are currently waiting. This information is also available in the V\$SESSION view. This view is helpful in diagnosing sessions that are proceeding very slowly or appear to be hung.

Set the TIMED\_STATISTICS dynamic initialization parameter to TRUE to obtain values in the WAIT\_TIME column.

### Columns

- SID: Session identifier
- SEQ#: Sequence number identifying the wait
- EVENT: Resource or event waited for
- STATE: Current wait state. Possible values are:
  - WAITING – Session is currently waiting
  - WAITED UNKNOWN TIME – Duration of the last wait is unknown; this is the value when the TIMED\_STATISTICS parameter is set to FALSE
  - WAITED SHORT TIME – Last wait was less than a hundredth of a second
  - WAITED KNOWN TIME – Duration of the last wait is specified in the WAIT\_TIME column

- Each of the three parameters (P1, P2, P3) have the following columns:
  - P1TEXT: Description of the first additional parameter, which corresponds to the PARAMETER1 described for the V\$EVENT\_NAME view
  - P1: First additional parameter value
  - P1RAW: First additional parameter value, in hexadecimal
- Note: Not all of the parameter columns are used for all events.
- WAIT\_TIME with the following possible values:
  - > 0: The session's last wait time
  - = 0: The session is currently waiting.
  - = -1: The value is less than 1/100 of a second.
  - = -2: The system cannot provide timing information.
- SECONDS\_IN\_WAIT: Number of seconds the event waited

# Precision of System Statistics

- Views that include microsecond timings:
  - V\$SESSION\_WAIT, V\$SYSTEM\_EVENT,  
V\$SERVICE\_EVENT,  
V\$SESSION\_EVENT (TIME\_WAITED\_MICRO column)
  - V\$SQL, V\$SQLAREA (CPU\_TIME, ELAPSED\_TIME columns)
  - V\$LATCH, V\$LATCH\_PARENT, V\$LATCH\_CHILDREN (WAIT\_TIME column)
  - V\$SQL\_WORKAREA, V\$SQL\_WORKAREA\_ACTIVE (ACTIVE\_TIME column)
- Views that include millisecond timings:
  - V\$ENQUEUE\_STAT (CUM\_WAIT\_TIME column)



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

The Oracle Database server captures certain performance data with millisecond and microsecond granularity. Views that include microsecond and millisecond timings are listed in the slide. The actual granularity of the timing event depends on the operating system.

**Note:** Existing time columns in other views capture centisecond times.

The timing information gathered on system statistics is cumulative since the instance was started. Some session-level views of statistics record the timing of a single event.

# Quiz

System statistics and dynamic performance views provide cumulative counts for the important measures in the instance. These statistics are the basis for all instance tuning. The system statistics are the most reliable diagnostics available.

- a. True
- b. False



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

## Answer: b

Although snapshots of system statistics are the basis for much of the diagnostic tools that are used for performance tuning, the raw cumulative statistics include everything since the instance was started including warmup and idle times. The raw statistics are not as useful as the delta values provided by snapshots that cover periods of interest.

# Enterprise Manager: Overview

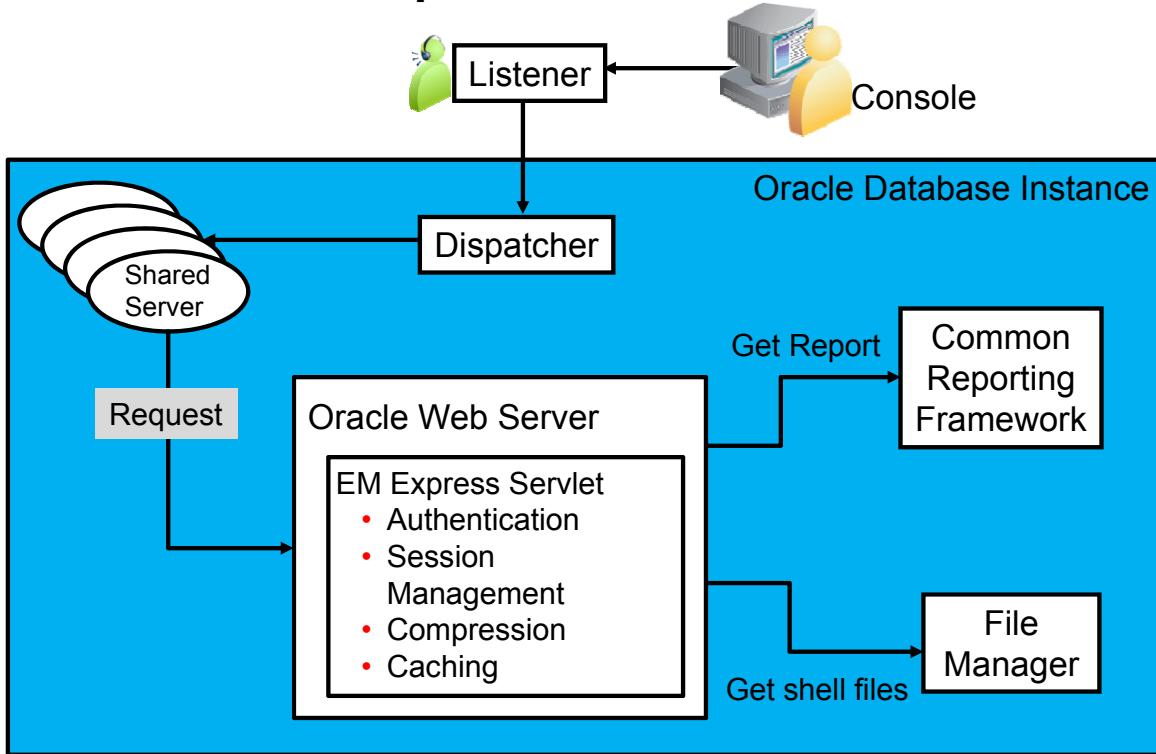
- Enterprise Manager Database Express:
  - Is a web-based database management tool built inside Oracle Database
  - Supports basic database administration and key performance management functions
- Enterprise Manager Cloud Control



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

Oracle Enterprise Manager Database Express (EM Express) is a web-based database management tool that is built inside Oracle Database. It supports basic database administration and key performance management functions.

# Oracle Enterprise Manager Database Express Architecture



ORACLE

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

Oracle Enterprise Manager Database Express is a lightweight administration tool. It provides an out-of-the-box browser-based management solution for a single Oracle database (or database cluster), including performance monitoring, configuration management, administration, diagnostics, and tuning.

Oracle Enterprise Manager Database Express uses a web-based console, communicating with the built-in web server available in XML DB.

As requests from the console are processed, the Enterprise Manager Database Express servlet handles the requests, including authentication, session management, compression, and caching. The servlet passes requests for reports to the Common Reporting Framework and actions requiring shell files to the File Manager.

Enterprise Manager Database Express is available only when the database is open. This means that Enterprise Manager Database Express cannot be used to start up the database. Other operations that require that the database change state, such as enable or disable ARCHIVELOG mode, are also not available in Enterprise Manager Database Express.

# Configuring Enterprise Manager Database Express

- Configure an HTTP listener port for each database instance.
  - Verify the DISPATCHERS parameter.

```
dispatchers= (PROTOCOL=TCP) (SERVICE=sampleXDB)
```

- Use the DBMS\_XDB\_CONFIG.setHTTPPort procedure.

```
exec DBMS_XDB_CONFIG.setHTTPPort(5500)
```

- Launch Enterprise Manager Database Express:

```
http://hostname:5500/em
```

- Use a different port for each instance.
- Browser requires Flash plug-in.



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

Oracle Enterprise Manager Database Express is configurable with a single click in Database Configuration Assistant (DBCA).

Enterprise Manager Database Express requires that the XMLDB components are installed. All Oracle version 12.1.0 databases have XMLDB installed.

To activate Enterprise Manager Database Express in a database, verify that the DISPATCHERS initialization parameter has at least one dispatcher configured for the XMLDB service with the TCP protocol.

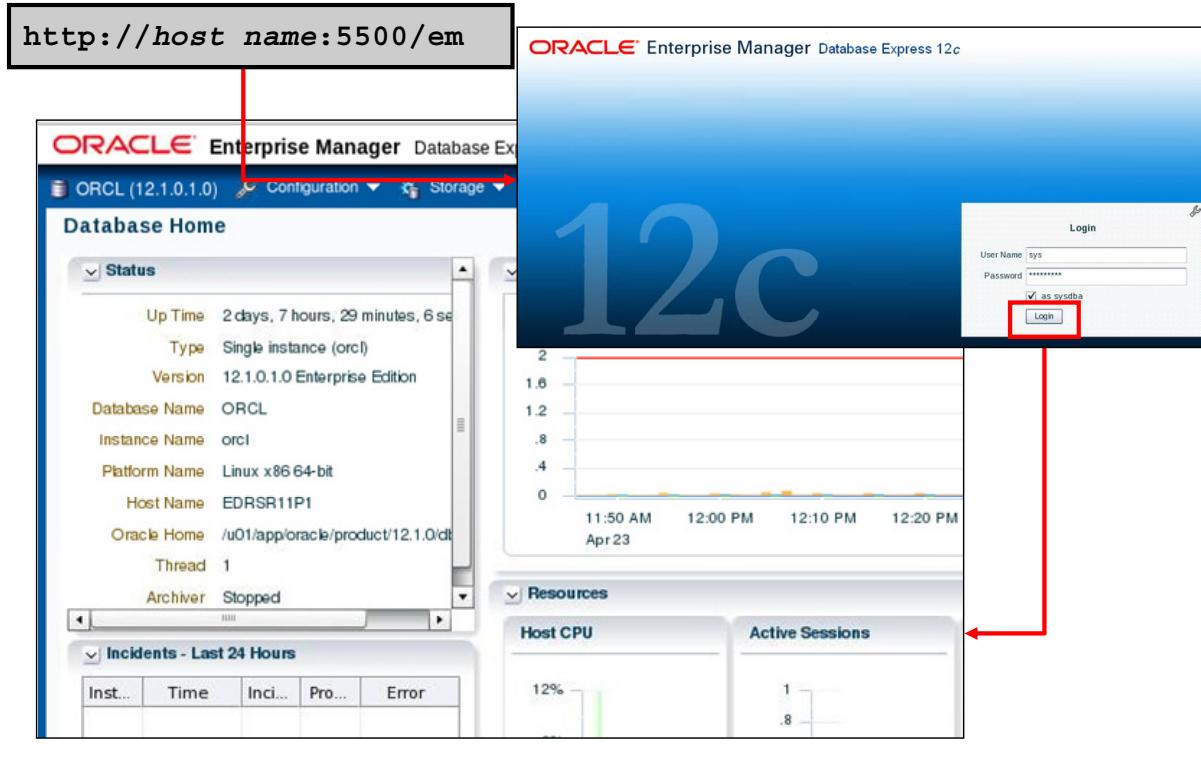
Use the SETHTTPPORT procedure in the DBMS\_XDB\_CONFIG package to configure a port on the server. Connect to the Enterprise Manager Database Express console with the URL shown in the slide. Substitute the host name of the server and the port number you set using the SETHTTPPORT procedure in the URL.

If you have multiple database instances to monitor on the same machine, set a different port for each. To find the port used for each database instance, execute the following statement:

```
SQL> SELECT dbms_xdb_config.gethttpport FROM DUAL;
```

Enterprise Manager Database Express uses Shockwave Flash (SWF) files, so the web browser must have the Flash plug-in installed.

# Accessing the Enterprise Manager Database Express Database Home Page



ORACLE

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

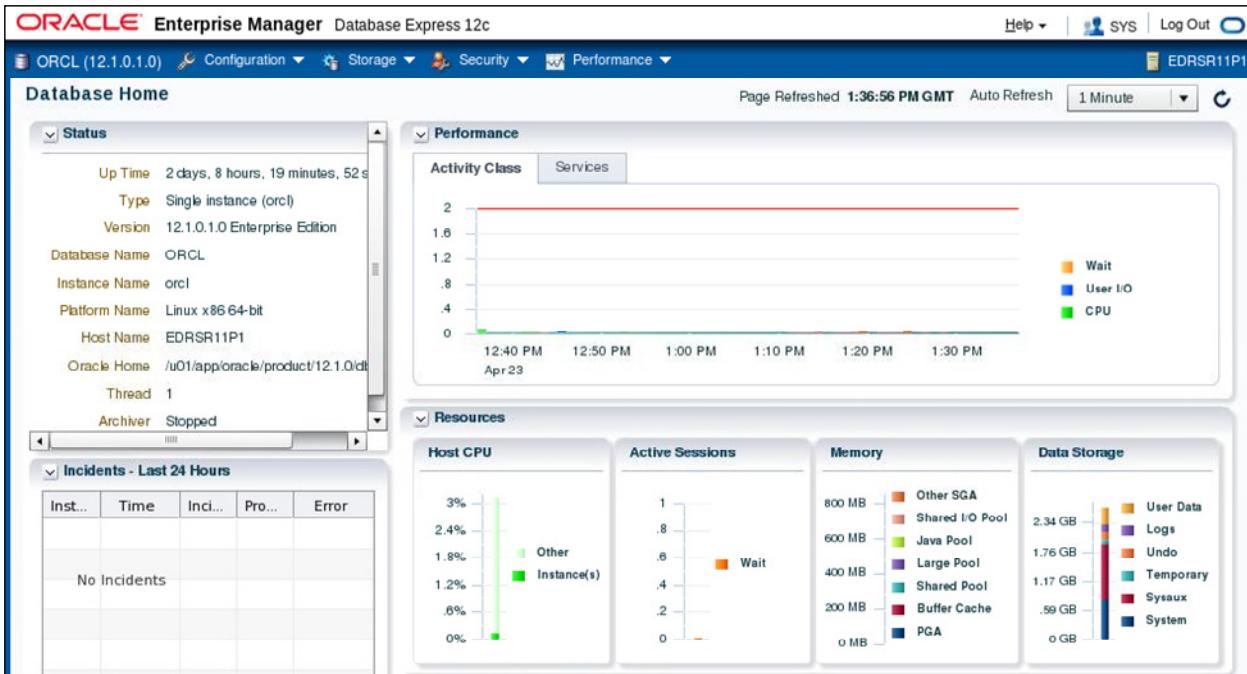
You can access Enterprise Manager Database Express by opening your web browser and entering the following URL: `http://host name:port number/em`.

*Host name* is the name or address of your computer. *Port number* is the Enterprise Manager Database Express HTTP port number that is specified during installation. The default port is 5500. You can find the configured port by using the PL/SQL function `DBMS_XDB_CONFIG.getHTTPPort`

The Enterprise Manager Database Express Database Home page is your starting point to monitor and administer your database. Use the Database Home page to:

- Determine the current status of the database by viewing a series of metrics
- Access the configuration, storage, security, and performance pages via menus

# Viewing Performance Information on the Database Home Page



**ORACLE**

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

The Database Home page enables you to monitor the status of your database and the workload it is experiencing.

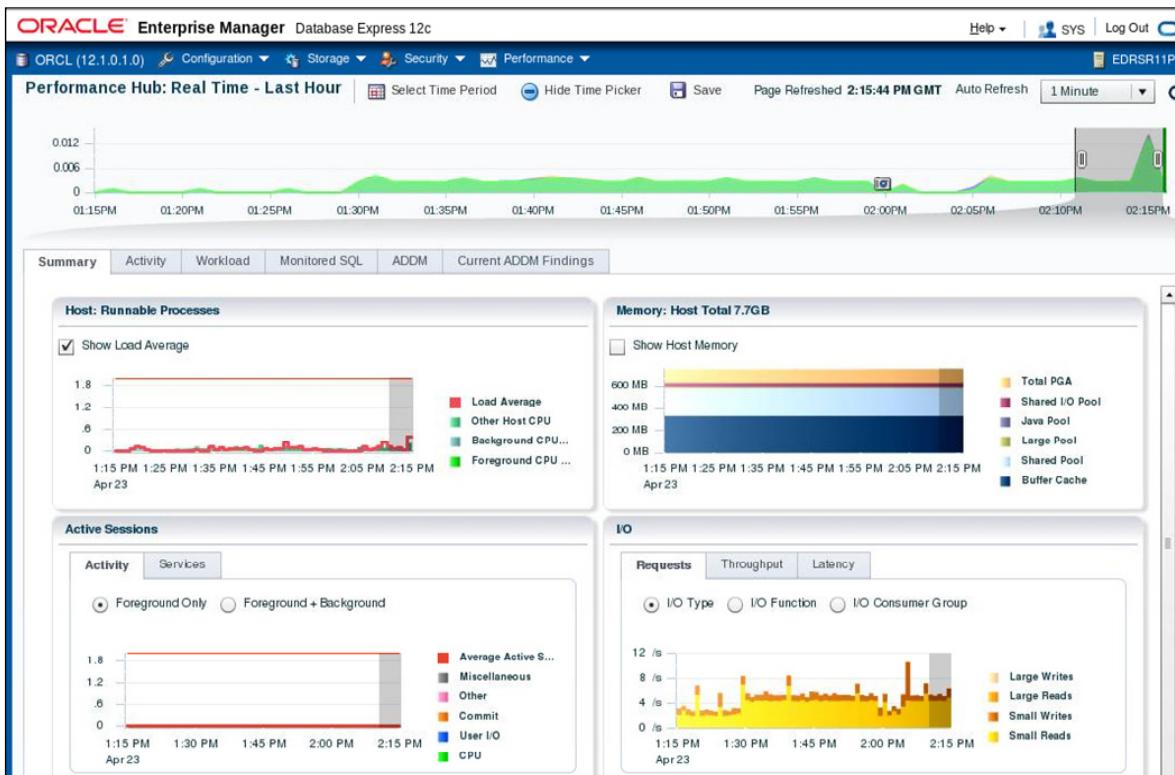
The various sections of the Database Home page provide detailed information. Menu options can be used to get more detail about problem areas, and in some cases, to obtain recommendations for resolving the problems.

The Database Home page contains the following charts and sections:

- **Activity Class chart:** Shows the average number of database sessions active for the past hour, including the type of activity for each session (on CPU, waiting for I/O, or waiting for another resource). For Oracle RAC, the chart shows activity aggregated across all instances in the cluster.
- **Services chart:** Shows the average number of database sessions active for the past hour for database services
- **Host CPU chart:** Shows the percentage of CPU time used by the database instance and other processes during the last minute, including the percentage of CPU used by foreground and background instance processes
- **Active Sessions chart:** Shows the average number of active sessions during the last minute, broken out by wait, user I/O, and CPU

- **Memory (GB) chart:** Shows the current memory utilization (as of the latest refresh time) broken out by the database shared pool, Java pool, buffer cache, PGA, and other SGA components
- **Data Storage (GB) chart:** Shows the current space usage (as of the latest refresh time) broken out by user data, database log files, undo tablespaces, and temporary, SYSAUX, and SYSTEM tablespaces
- **SQL Monitor section:** Shows information about monitored SQL statement executions. For each SQL statement, the table includes the SQL ID, session ID, and SQL text. Also information about the status, duration, parallelization, and database time is provided.
- **Incidents - Last 24 Hours section:** Displays a table that provides information about database incidents (an occurrence of a critical error) that have occurred in the past 24 hours
- **Running Jobs section:** Displays a table that shows database jobs that are currently running

# Viewing the Performance Hub Page



ORACLE

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

The Performance Hub enables you to view performance data for a specified time period. After a time period is selected, the performance information is collected and displayed based on performance topic areas.

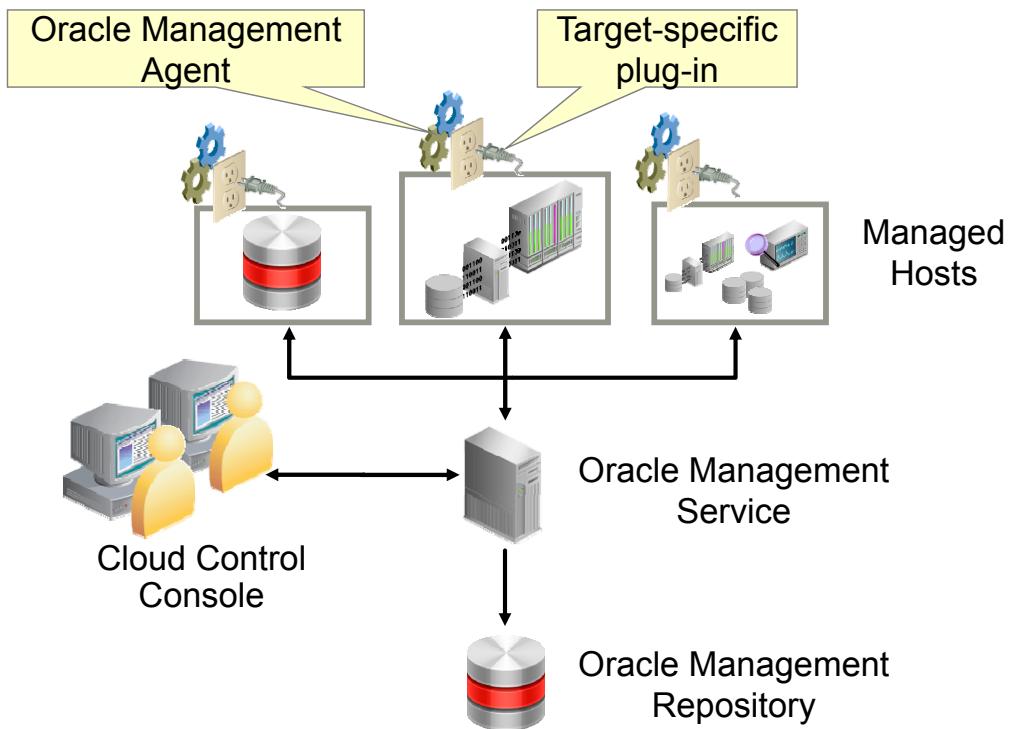
You can choose between the display of real-time data and historical data. When real-time data is selected, more granular data is presented because data points are available every minute. When historical data is selected, more detailed data is presented, but the data points are averaged out to the Automatic Workload Repository (AWR) interval.

The following tabs are available in the Performance Hub, depending on whether real-time or historical data is selected for the time period:

- **Summary:** Provides an overall view of the performance of the system for the specified time period
- **Activity:** Shows Active Session History (ASH) analytics
- **Workload:** Workload profile charts show the pattern of user calls, parse calls, Redo Size and SQL\*Net over the last 60 minutes in real-Time mode. The Sessions chart show the logon rate, current logons, and open cursors.
- **Monitored SQL:** Shows information about monitored SQL statements that were executing or that completed during the selected time period

- **ADDM:** Provides performance findings and recommendations that have been found by Automatic Database Diagnostics Monitor (ADDM) for tasks performed in the database during the selected time period
- **Current ADDM Findings:** Provides real-time ADDM findings for the past five minutes
- **Database Time:** Shows wait events by category for various metrics, and to view time statistics for various metrics for the selected time period
- **Resources:** Shows operating system resource usage statistics, I/O resource usage statistics, and memory usage statistics for the selected time period
- **System Statistics:** Shows database statistics by value, per transaction, or per second for the selected time period

# Oracle Enterprise Manager Cloud Control Components



**ORACLE**

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

Oracle Enterprise Manager Cloud Control is composed of four main components as illustrated in the slide:

- Oracle Management Repository (OMR)
- Oracle Management Service (OMS)
- Oracle Management Agent (OMA or agent) with target-specific plug-ins
- Cloud Control Console

The Oracle Management Agent runs on hosts, gathering metric data about those host environments as well as using plug-ins to monitor availability, configuration, and performance and to manage targets running on the host. The agents communicate with the Oracle Management Service to upload metric data collected by them and their plug-ins. In turn, the OMS stores the data it collects in the Oracle Management Repository where it can be accessed by the OMS for automated and manual reporting and monitoring. The OMS also communicates with the agents to orchestrate the management of their monitored targets. As well as coordinating the agents, the OMS runs the Cloud Control Console web pages that are used by administrators and users to report on, monitor, and manage the computing environment that is visible to Cloud Control via the agents and their plug-ins.

# Using Features of the Oracle Management Packs

Monitoring and tuning by using packs		
<b>Oracle Diagnostics Pack</b> <ul style="list-style-type: none"> <li>• Performance monitoring and diagnostics</li> <li>• Automatic Workload Repository (AWR)</li> <li>• Automatic Database Diagnostic Monitor (ADDM)</li> <li>• Compare Period/Real Time ADDM</li> <li>• Active Session History (ASH)</li> <li>• Exadata Administration/Performance/Resource Utilization</li> <li>• Blackouts</li> <li>• Notifications</li> <li>• Metric and Alert/Event History</li> <li>• Dynamic metric baselines/adaptive metric thresholds</li> <li>• Monitoring Templates/Template Collections</li> <li>• Bottleneck detection/component top wait event analysis</li> <li>• Memory access mode</li> </ul>		
<b>Oracle Tuning Pack</b> <ul style="list-style-type: none"> <li>• SQL Access Advisor</li> <li>• SQL Tuning Advisor</li> <li>• SQL Tuning Sets</li> <li>• Automatic SQL Tuning</li> <li>• SQL Profiles</li> <li>• Real-time Database Operations Monitoring</li> <li>• Real-time SQL and PL/SQL Monitoring</li> <li>• Reorganize objects</li> <li>• Automatic Plan Evolution of SQL Plan Management</li> </ul>	<b>Database Lifecycle Management Pack For Oracle Database</b> <ul style="list-style-type: none"> <li>• Client system analyzer</li> <li>• Compliance</li> <li>• Configuration Collection Extensibility</li> <li>• Configuration Compare/History/Save/Search</li> <li>• Configuration Instance Browser</li> <li>• Configuration Topology</li> <li>• Copy (database) objects</li> <li>• Database Patching/Provisioning/Upgrade</li> <li>• File Synchronization</li> <li>• Notifications</li> <li>• Real-time config/schema change detection</li> <li>• Schema/data comparisons/synchronization</li> <li>• Schema Change Plans</li> </ul>	Monitoring and tuning without the use of packs
<ul style="list-style-type: none"> <li>• SQL traces</li> <li>• Statspack</li> <li>• System statistics</li> <li>• Wait model</li> <li>• Time model</li> <li>• OS statistics</li> <li>• Metrics</li> <li>• Service statistics</li> <li>• Histograms</li> <li>• Optimizer statistics</li> <li>• SQL statistics</li> </ul>		



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

The management packs names and features are listed in the left part of the slide. They all require a separate license that can be purchased only with Enterprise Edition. The features in these packs are accessible through Oracle Enterprise Manager Cloud Control and APIs provided with the Oracle database software.

The CONTROL\_MANAGEMENT\_PACK\_ACCESS initialization parameter controls access to Oracle Diagnostics Pack and Oracle Tuning pack with the following values:

- DIAGNOSTIC+TUNING: Functionality for Oracle Diagnostics Pack and Oracle Tuning Pack is enabled
- DIAGNOSTIC: Functionality for Oracle Diagnostics Pack is enabled
- NONE: Functionality for Oracle Diagnostics Pack and Oracle Tuning Pack is disabled in EMCC, but the views and APIs are still accessible

**Note:** Licenses are required to use the views and APIs.

Oracle Diagnostic Pack provides automatic performance diagnostic and advanced system-monitoring functionality. Features can also be accessed through APIs and commands:

- The DBMS\_WORKLOAD\_REPOSITORY package
- The DBMS\_ADDM package
- The DBMS\_ADVISOR package, if you specify ADDM as the value for the ADVISOR\_NAME parameter, or if you specify any value starting with the ADDM prefix for the value of the TASK\_NAME parameter
- The DBMS\_WORKLOAD\_REPOSITORY package
- The V\$ACTIVE\_SESSION\_HISTORY dynamic performance view
- All data dictionary views beginning with the DBA\_HIST\_ prefix, along with their underlying tables
- All data dictionary views with the DBA\_ADVISOR\_ prefix if queries to these views return rows with the value ADDM in the ADVISOR\_NAME column or a value of ADDM\* in the TASK\_NAME column or the corresponding TASK\_ID
- The DBA\_STREAMS\_TP\_PATH\_BOTTLENECK view
- All views beginning with DBA\_ADDM\_
- The following reports found in the /rdbms/admin/ directory of the ORACLE\_HOME directory are part of this pack: awrrpt.sql, awrrpti.sql, addmrtp.sql, addmrpti.sql, ashrrpt.sql, ashrrpti.sql, awrddrpt.sql, awrddrpi.sql, awrsqrpi.sql, awrsqrpt.sql, awrextr.sql, and awrload.sql, awrextr.sql, awrload.sql, awrinfo.sql, spawrrac.sql.

Oracle Tuning Pack provides expert performance management for the Oracle Database environment, including SQL tuning and storage optimizations. The Oracle Diagnostic Pack is a prerequisite product to the Oracle Tuning Pack. Features of the pack can also be accessed through APIs and commands:

- The DBMS\_SQLTUNE package
- The DBMS\_ADVISOR package, when the value of the ADVISOR\_NAME parameter is either SQL Tuning Advisor or SQL Access Advisor
- V\$SQL\_MONITOR
- V\$SQL\_PLAN\_MONITOR
- The sqltrpt.sql report found in the /rdbms/admin/ directory of the ORACLE\_HOME directory

The Database Lifecycle Management Pack is a comprehensive solution that helps you to automate the processes required to manage the Oracle Database Lifecycle. It eliminates manual and time consuming tasks related to discovery, initial provisioning, patching, configuration management and ongoing change management. It also provides compliance frameworks for reporting and management of industry and regulatory compliance standards.

If you cannot purchase the packs mentioned above, especially the Database Diagnostics and Database Tuning packs, you can still use the traditional approach to performance tuning and monitoring by using Statspack reports, SQL traces, and most of the base statistics as shown in the previous slide.

Refer to *Oracle Database Licensing Information 12c* for additional and current information about the licensing of packs.

# Viewing the Alert Log

The screenshot shows the Oracle Enterprise Manager interface for Cloud Control 12c. The left sidebar has 'Logs' selected under 'orcl'. A red arrow points from the 'Logs' menu to the 'Text Alert Log Contents' link in the center panel. The right panel displays the 'Text Alert Log Contents' for the 'orcl' database. The log output is as follows:

```

Sweep [inc2][12461]: completed
Errors in file /u01/app/oracle/diag/rdbms/orcl/orcl/trace/orcl_j004_17702.trc (incident=12461):
ORA-04036: PGA memory used by the instance exceeds PGA_AGGREGATE_LIMIT
ORA-04036: PGA memory used by the instance exceeds PGA_AGGREGATE_LIMIT
Incident details in: /u01/app/oracle/diag/rdbms/orcl/orcl/incident/incdir_12461/orcl_j004_17702_i12461.trc
Dumping diagnostic data in directory=[cdmp_20121016221549], requested by (instance=1, osid=17702 (J004)), opidrv aborting process J004 ospid (17702) as a result of ORA-28
Tue Oct 16 22:16:30 2012
Sweep [inc][12461]: completed
Sweep [inc2][12461]: completed
Sweep [inc2][12460]: completed
Wed Oct 17 02:00:01 2012
Closing Resource Manager plan via scheduler window
Clearing Resource Manager plan via parameter
Wed Oct 17 10:32:56 2012
Thread 1 advanced to log sequence 23 (LGWR switch)
  Current log# 2 seq# 23 mem# 0: /u01/app/oracle/oradata/orcl/redo02.log
Wed Oct 17 12:23:29 2012

```

**ORACLE**

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

Each database has an `alert_<sid>.log` file. The file is on the server with the database and is stored in `$ORACLE_BASE/diag/rdbms/<db_name>/<SID>/trace` by default if `$ORACLE_BASE` is set.

The alert file of a database is a chronological log of messages such as the following:

- Any nondefault initialization parameters used at startup
- All internal errors (ORA-600), block corruption errors (ORA-1578), and deadlock errors (ORA-60) that occurred
- Administrative operations, such as the SQL statements CREATE, ALTER, DROP DATABASE, and TABLESPACE; and the Enterprise Manager or SQL\*Plus statements STARTUP, SHUTDOWN, ARCHIVE LOG, and RECOVER
- Several messages and errors relating to the functions of shared server and dispatcher processes
- Errors during the automatic refresh of a materialized view

Oracle Database uses the alert log to keep a record of these events as an alternative to displaying the information on an operator's console. (Some systems also display this information on the console.) If an administrative operation is successful, a message is written in the alert log as "completed" along with a time stamp.

Enterprise Manager Cloud Control monitors the alert log file and notifies you of critical errors. You can also view the log to see noncritical error and information messages. Because the file can grow to an unmanageable size, you can periodically back up the alert file and delete the current alert file. When the database attempts to write to the alert file again, it creates a new one.

**Note:** There is an XML version of the alert log in the \$ORACLE\_BASE/diag/rdbms/<db\_name>/<SID>/alert directory.

**To determine the location of the alert log with SQL\*Plus:**

- Connect to the database with SQL\*Plus (or another query tool such as SQL Developer).
- Query the V\$DIAG\_INFO view.

**Note:** The V\$DIAG\_INFO view provides the path to the diagnostic directories, and current trace file.

**To view the alert log with ADRCI:**

- Start ADRCI in interactive mode.

```
$ adrci
adrci> show alert
Choose the home from which to view the alert log:
```

```
1: diag/rdbms/emrep/emrep
2: diag/rdbms/orcl/orcl
3: diag/tnslnsr/EDRSR7P1/listener
Q: to quit
```

```
Please select option: 2
```

- If more than one ADR home is current, you are prompted to select a single ADR home from a list. The alert log is displayed, with XML tags omitted, in your default editor.
- Exit the editor to return to the ADRCI command prompt.
- ADRCI has the command option to search the alert log by a key value, and send output to terminal or file. To see the SHOW ALERT options, use the help command.

```
adrci> HELP SHOW ALERT
```

## Using Alert Log Information as an Aid in Tuning

The alert log file contains the following information that can be used to aid in tuning the database:

- Time to perform archiving
- Instance recovery start and complete times
- Deadlock and timeout errors
- Incomplete checkpoints
- Checkpoint start and end times



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

The information listed in the slide and additional information are written to the alert log. The information written to the alert log changes somewhat with each version of Oracle Database. Some values, such as the checkpoint start and end times, are written only when requested. These values are written to the alert log file only if the `LOG_CHECKPOINTS_TO_ALERT` parameter has been set to TRUE.

The alert log file can grow to an unmanageable size. You can safely delete the alert log while the instance is started, although you should consider making an archived copy of it first. This archived copy could prove valuable if you should have a future problem that requires investigating the history of an instance.

**Note:** Both the versions of the alert log, text and XML, should be archived and deleted periodically.

For example, suppose the DBA noticed a change in performance statistics. The DBA finds that an instance parameter has changed since the last baseline. To confirm that the performance change corresponds to the parameter change, the alert log can be searched. The alert log lists all the non-default parameter settings on each startup, and records `ALTER SYSTEM` commands with a time stamp.

## Administering the DDL Log File

- Enable the capture of certain DDL statements to a DDL log file by setting ENABLE\_DDL\_LOGGING to TRUE.
- DDL log contains one log record for each DDL statement.
- Two DDL logs containing the same information:
  - XML DDL log: named log.xml
  - Text DDL: named ddl\_<sid>.log
- Example:

```
$ more ddl_orcl.log
Thu Nov 15 08:35:47 2012
diag_adl:drop user app_user
```



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

The DDL log is created only if the ENABLE\_DDL\_LOGGING initialization parameter is set to TRUE. When this parameter is set to FALSE, DDL statements are not included in any log. A subset of executed DDL statements is written to the DDL log. Refer to the *Oracle Database Reference* for a complete list of DDL commands that are captured in the DDL log. These files are located in the \$ORACLE\_BASE/diag/rdbms/<dbname>/<sid>/log directory.

Locate the log files as follows:

```
$ pwd
/u01/app/oracle/diag/rdbms/orcl/orcl/log
$ ls
ddl  ddl_orcl.log  debug  test
$ cd ddl
$ ls
log.xml
```

**Note:** Setting the ENABLE\_DDL\_LOGGING parameter to TRUE requires licensing the Database Lifecycle Management Pack.

## Understanding the Debug Log File

- The debug log contains warnings about conditions, states, or events that do not inhibit correct operation of an Oracle Database component.
- The log is intended for use by Oracle Support when diagnosing a problem.
- It is included in incident packaging service (IPS) incident packages.
- It is written to  
`$ORACLE_BASE/diag/rdbms/<db_name>/<SID>/debug`.



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

The debug log contains warnings generated by Oracle Database components. The warnings are about conditions, states, or events that may be out of the ordinary but do prevent the component from operating correctly. For this reason, these warnings are not written to the alert log. If they are needed in the future to diagnose a problem, they are recorded in the debug log.

Typically, the debug log would be needed only by Oracle Support to diagnose a problem, so you do not need to view the contents of the debug log on a regular basis. The debug log is also included in incident packaging service (IPS) incident packages. IPS is discussed in detail in the course titled *Oracle Database 12c: Administration* and the *Oracle Database Administrators Guide 12c Release 1*.

# User Trace Files

- Server-process tracing can be enabled or disabled at the session or instance level.
- User trace files are created on a per server process basis.
- User trace files can be created by:
  - Enabling SQL TRACE
    - A trace file contains statistics for traced SQL statements in that session.
  - Performing a BACKUP CONTROL FILE TO TRACE
  - Processing errors



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

Server processes can generate user trace files at the request of the user or DBA.

## Instance-Level Tracing

Instance-level tracing should be enabled only when absolutely necessary. Tracing all sessions will create an I/O load and can fill the file system quickly. This trace logging is enabled or disabled by the `DBMS_MONITOR.DATABASE_TRACE_ENABLE()` procedure.

## Session-Level Tracing

The following statement enables the writing to a trace file for a particular session:

```
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE (8,12, waits=>TRUE,
binds=>TRUE);
```

where 8 and 12 are the system identifier and serial number of the connected user. Typically, only a DBA has the permissions required to enable tracing on any session.

The `DBMS_MONITOR` package is created when the `catproc.sql` script is run. This script is located in the following directory:

- **On UNIX:** `$ORACLE_HOME/rdbms/admin`
- **On Windows:** `%ORACLE_HOME%\rdbms\admin`

To enable the writing of a trace file for your current session, execute the following command:

```
EXECUTE DBMS_SESSION.SET_SQL_TRACE (TRUE)
```

## Background Processes Trace Files

- The Oracle Database server dumps information about errors detected by any background process into trace files.
- Oracle Support uses these trace files to diagnose and troubleshoot.
- These files do not usually contain tuning information.



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

The background processes create these files. In general, these files contain diagnostic information, not information regarding performance tuning. However, by using events, information regarding performance can be written to these files. Database events can be set by the DBA, but is usually done only under the supervision of Oracle Support. An exception to this rule is the 10053 event that can be used to trace the optimizer choices. This event will be explained later in the course.

Most trace files are difficult to read because they are intended for diagnoses and troubleshooting by Oracle Support, but they can contain valuable information that a DBA can use.

**Note:** V\$PROCESS provides the name of the trace file associated with every current process.

# Quiz

Which diagnostic tool would you use to discover when the last database startup and backup occurred?

- a. Trace files
- b. Alert log
- c. Enterprise Manager Cloud Control database home page
- d. V\$ views



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

## Answer: b, c

The alert log and the Enterprise Manager Cloud Control database home page both contain this information.

## Summary

In this lesson, you should have learned how to:

- View the top wait events to determine the highest wait
- View the time model to diagnose performance issues
- Use dynamic performance views to view statistics and wait events
- Use Enterprise Manager monitoring
- Identify the key tuning components of alert logs
- Identify the key tuning components of user trace files



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

## **Practice 2 Overview: Using Basic Tools**

This practice covers the following topics:

- Viewing the top wait events and the time model
- Using the alert log information for tuning
- Viewing system statistics
- Viewing wait events



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



# Using Automatic Workload Repository

3

ORACLE®

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

# Objectives

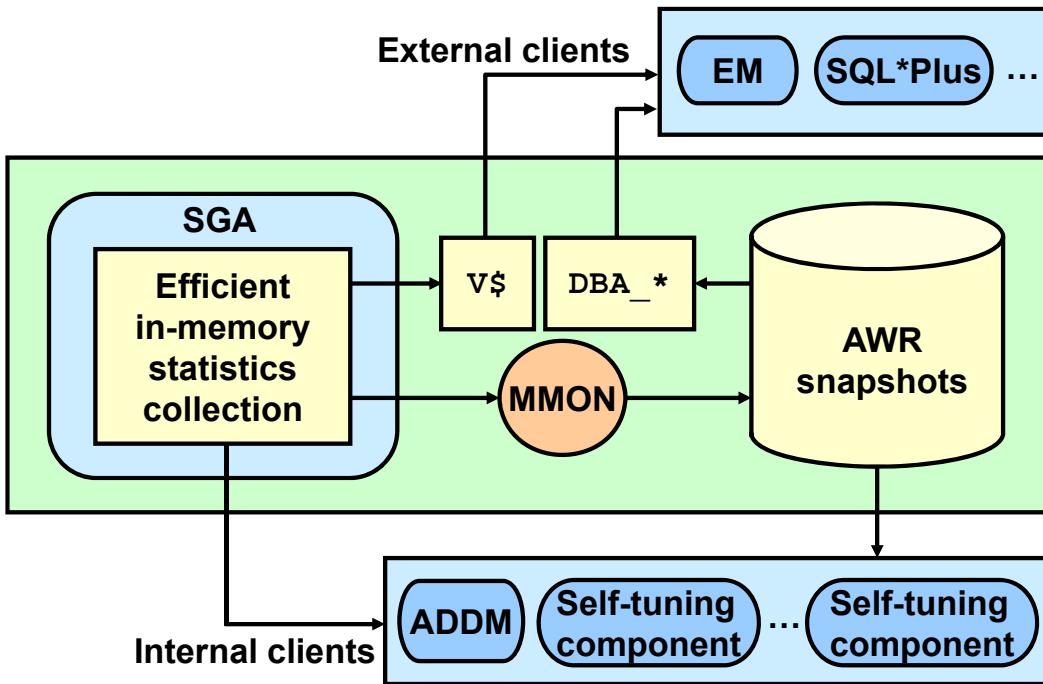
After completing this lesson, you should be able to do the following:

- Create and manage AWR snapshots
- Generate AWR reports
- Create Compare Periods reports



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

# Automatic Workload Repository: Overview



**ORACLE**

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

Automatic Workload Repository (AWR) is the infrastructure that provides services to Oracle Database components to collect, maintain, and use statistics for problem detection and self-tuning purposes.

The AWR infrastructure consists of two major parts:

- An in-memory statistics collection facility that is used by various components to collect statistics. These statistics are stored in memory for performance reasons. Statistics stored in memory are accessible through dynamic performance (v\$) views.
- AWR snapshots represent the persistent portion of the facility. The AWR snapshots are accessible through data dictionary (DBA) views and Enterprise Manager.

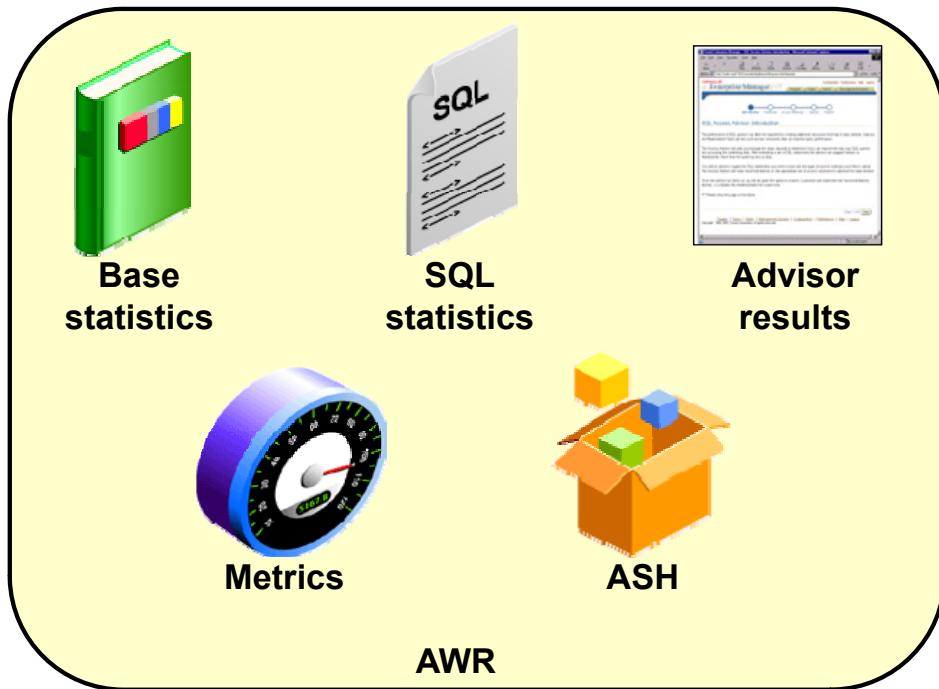
Statistics are stored in persistent storage for several reasons:

- The statistics need to survive instance crashes.
- Historical data for baseline comparisons is needed for certain types of analysis.
- Memory overflow: When old statistics are replaced by new ones due to memory shortage, the replaced data can be stored for later use.

The memory version of the statistics is transferred to disk on a regular basis by a background process called MMON (Manageability Monitor).

With AWR, the Oracle Database server provides a way to capture historical statistics data automatically, without the intervention of DBAs.

# Automatic Workload Repository Data



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

AWR captures a variety of statistics. AWR stores base statistics, that is, counters and value statistics (for example, log file switches and process memory allocated). AWR captures SQL statistics such as disk reads per SQL statement. Metrics such as physical reads per minute are also captured.

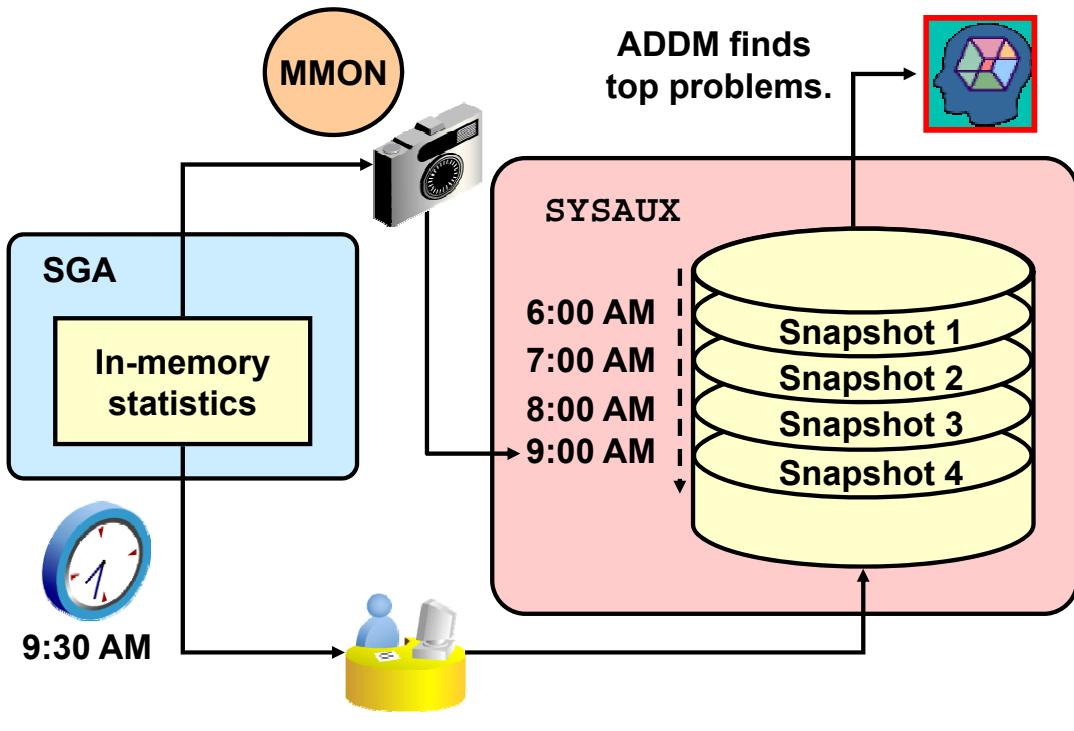
The Active Session History (ASH) data is captured first to memory at one-second intervals for only those sessions that are currently active (performing a database call). Then the ASH data is reduced by a factor of ten by storing to disk a random sample of the in-memory data. The ASH data is heavily used by Automatic Database Diagnostic Monitor (ADDM) to identify root causes of performance issues.

The advisor reports produced by ADDM, the Segment Advisor, and other advisors are also stored in AWR for later viewing.

The statistics exist at two levels: the in-memory recent statistics in v\$ views, and persistent statistics that are stored on disk as snapshots and DBA\_\* views.

**Note:** The examples in this slide are not a complete list.

# Workload Repository



ORACLE

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

The workload repository is a collection of persistent system performance statistics owned by SYS. The workload repository resides in the SYSAUX tablespace and is one of the main SYSAUX occupants.

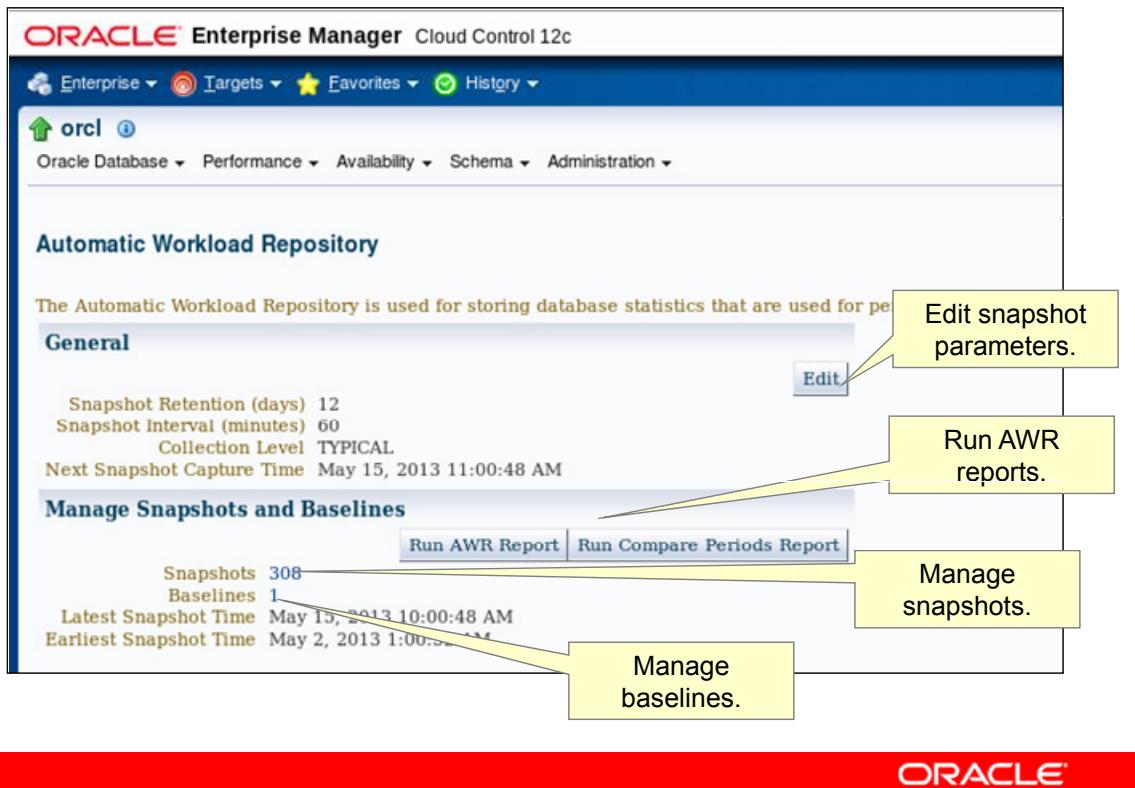
A **snapshot** is a set of performance statistics captured at a certain time. Snapshots are used for computing the rate of change of a statistic. Each snapshot is identified by a snapshot sequence number (`snap_id`) that is unique in the workload repository.

By default, snapshots are generated every 60 minutes. You can adjust this frequency by changing the `snapshot INTERVAL` parameter. Because internal advisories rely on these snapshots, be aware that adjustment of the interval setting can affect diagnostic precision. For example, if `INTERVAL` is set to 4 hours, you may miss spikes that occur within 60-minute intervals.

In a Real Application Clusters environment, each snapshot spans all nodes in a cluster. Snapshots for data in each node share the same `snap_id`, differentiated by their instance IDs. Snapshots in Real Application Clusters are captured at roughly the same time.

You can take manual snapshots by using Enterprise Manager. Taking manual snapshots is supported in conjunction with the automatic snapshots that the system generates. Manual snapshots are expected to be used when you want to capture the system behavior at two specific points in time that do not coincide with the automatic schedule.

# AWR Administration



ORACLE

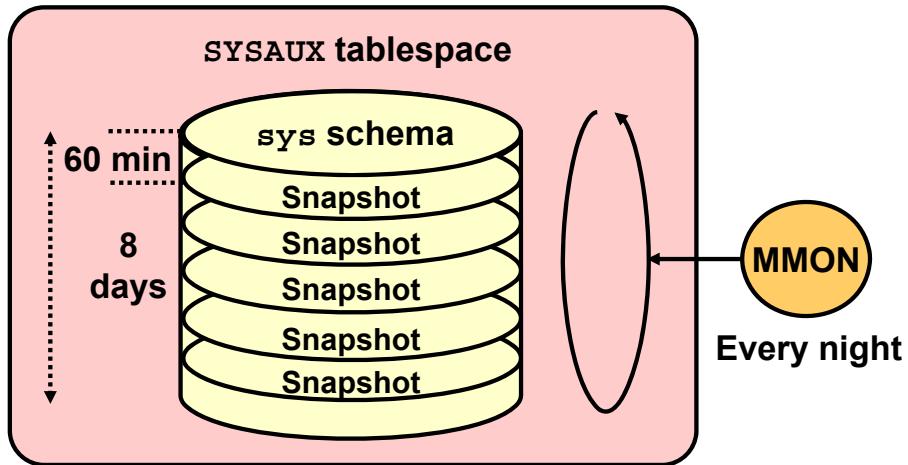
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Using Enterprise Manager Cloud Control, you can configure the RETENTION and INTERVAL parameters for capturing snapshots. To access the Automatic Workload Repository page, first navigate to the database home page. Then expand the Performance menu. Select AWR Administration from the AWR menu.

Using the Automatic Workload Repository page, you can:

- Edit the workload repository settings
- Look at the detailed information of created snapshots and manually create new ones
- Create baselines, also called preserved snapshot sets
- Generate an AWR report

# AWR Snapshot Purging Policy



**ORACLE**

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

You control the amount of historical AWR statistics by setting a retention period and a snapshot interval. In general, snapshots are removed automatically in chronological order. Snapshots that belong to baselines are retained until their baselines are removed or expire. On a typical system with 10 active sessions, AWR collections require 200 MB to 300 MB of space if the data is kept for seven days. The space consumption depends mainly on the number of active sessions in the system. A sizing script, `utlsyxsz.sql`, includes factors such as the size of the current occupants of the SYSAUX tablespace, number of active sessions, frequency of snapshots, and retention time. The `awrinfo.sql` script produces a report of the estimated growth rates of various occupants of the SYSAUX tablespace. Both scripts are located in the `$ORACLE_HOME/rdbms/admin` directory.

AWR handles space management for the snapshots. Every night the MMON process purges snapshots that are older than the retention period. If AWR detects that SYSAUX is out of space, it automatically reuses the space occupied by the oldest set of snapshots by deleting them. An alert is then sent to the DBA to indicate that SYSAUX is under space pressure.

# Managing Snapshots with PL/SQL

## DBMS\_WORKLOAD\_REPOSITORY Package

Procedure Name	Description
CREATE_SNAPSHOT	Create a manual snapshot immediately
DROP_SNAPSHOT_RANGE	Drop a range of snapshots
MODIFY_SNAPSHOT_SETTINGS	Modify the snapshot settings



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

The DBMS\_WORKLOAD\_REPOSITORY PL/SQL package contains procedures that enable you to manage the workload repository. For example, you can find procedures for managing snapshots and baselines in this package. The procedures shown are only a few of the procedures provided. Most of the procedures are used by Enterprise Manager to manage the Automatic Workload Repository, and you seldom need to use the procedures directly.

**Note:** For more information about these procedures and other procedures to manage the AWR contained in the DBMS\_WORKLOAD\_REPOSITORY package, refer to the *Oracle Database PL/SQL Packages and Types Reference* guide.

## AWR Snapshot Settings

```
DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS (
    retention IN NUMBER DEFAULT NULL,
    interval   IN NUMBER DEFAULT NULL,
    topnsql    IN NUMBER DEFAULT NULL);
```



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

With the `MODIFY_SNAPSHOT_SETTINGS` procedure, you can control the snapshot parameters. You can use this procedure to change:

- The retention period. `RETENTION` is specified in minutes. The default is eight days; the minimum is one day. Setting `RETENTION` to the value 0 disables automatic purging.
- The `INTERVAL` between snapshots. The minimum value is 10 minutes, the maximum is 100 years, and the default value is 60 minutes.
- The number of Top SQL statements for which to capture performance data. You are allowed to specify the following values: `DEFAULT`, `MAXIMUM`, *n*, where *n* is the number of Top SQL statements to flush for each SQL criteria such as Elapsed Time and CPU Time. Specify `DEFAULT` to capture the Top 30 for level `TYPICAL` and Top 100 for level `ALL` of `STATISTICS_LEVEL`. Specify `MAXIMUM` to capture the complete set of SQL in the cursor cache. Specify `NONE` to keep the current setting.

**Note:** Under exceptional circumstances, automatic snapshot collection can be completely turned off by setting the snapshot interval to 0. The automatic collection of the workload and statistical data is stopped and much of the Oracle self-management functionality is not operational. In addition, you are unable to manually create snapshots. For this reason, Oracle Corporation strongly recommends that you do not turn off automatic snapshot collection.

# Manual AWR Snapshots

Create a snapshot:

```
DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT () ;
```

Drop one or more snapshots:

```
DBMS_WORKLOAD_REPOSITORY.DROP_SNAPSHOT_RANGE (
    LOW_SNAP_ID => 102,
    HIGH_SNAP_ID => 105);
```

Create and delete snapshots by using Enterprise Manager:

Select	ID	Capture Time	Collection Level
<input type="radio"/>	472	May 14, 2013 1:00:53 PM	TYPICAL

ORACLE

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

Snapshots are normally collected automatically. There are times when you may want to collect snapshots before or after particular events that do not match the automatic collection periods. These events could be test workloads or problem events that you can trigger.

# Generating AWR Reports

The screenshot illustrates the steps to generate an AWR report:

- Select Beginning Snapshot:** The 'Go' button is highlighted with a red box and a red arrow pointing down to the 'View Report' step.
- Select Ending Snapshot:** The 'Go' button is highlighted with a red box and a red arrow pointing down to the 'View Report' step.
- View Report:** The report details are shown, including Beginning Snapshot ID 479 and Beginning Snapshot Capture Time May 15, 2013 10:00:44 AM.

ORACLE

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

AWR can produce a summary report on statistics stored in the workload repository. The report contains general information about the overall behavior of the system over a time period defined by two snapshots.

To generate an AWR report, navigate to the Automatic Workload Repository page in Enterprise Manager Cloud Control. On this page, click the link corresponding to the number of snapshots. The Snapshots page appears. On the Snapshots page, select the beginning snapshot, select View Report from the Actions drop-down list, and click Go. On the View Report page, select the ending snapshot and click OK.

# Generating AWR Reports by Using SQL\*Plus

- Execute `$ORACLE_HOME/rdbms/admin/awrrpt.sql` to generate an AWR report.
- The user executing the script must have the `SELECT_CATALOG_ROLE` privilege.
- Script prompts for the following:
  - Report format: HTML or text
  - Number of days from which snapshots are to be chosen
  - Beginning and ending snapshot IDs
  - File name for the report



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

The AWR report has the same information, whether it is generated from SQL\*Plus or from Enterprise Manager. The `awrrpt.sql` SQL\*Plus script in the `ORACLE_HOME/rdbms/admin` directory produces the report. The user executing the script must have the `SELECT_CATALOG_ROLE` privilege. The script prompts for the following report options:

- HTML or text report
- The number of days of snapshots to choose from. Entering the number of days shows you the most recent snapshots being taken. You can also determine which `SNAP_ID`s you should use by querying the `DBA_HIST_SNAPSHOT` view to retrieve the mapping between a `SNAP_ID` and the wall-clock time.
- Begin `SNAP_ID`, end `SNAP_ID`: A snapshot pair that defines the reporting time period
- File name: The user-specified file into which the report is written

The report contains the same information whether it is produced as a text or as an HTML report. However, HTML reports can be viewed in a web browser, and the advantage of an HTML report is the presence of links to the detail sections of the report.

# Reading the AWR Report

**WORKLOAD REPOSITORY report for**

DB Name	DB Id	Instance	Inst num	Startup Time	Release	RAC
ORCL	1346382964	orcl		1 08-Jul-13 07:07	12.1.0.1.0	NO
Host Name	Platform	Cpus	Cores	Sockets	Memory (GB)	
EDRSR12P0	Linux x86 64-bit	2	2	1	7.65	
Snap Id	Snap Time	Sessions		Cursors/Session		
Begin Snap:	291	09-Jul-13 12:39:50		46	2.3	
End Snap:	292					
Elapsed:						
DB Time:						

Top 10 Foreground Events by Total Wait Time

Event	Waits	Total Wait Time (sec)	Wait Avg(ms)	% DB time	Wait Class
buffer busy waits	104,864	973.2	9	66.6	Concurrency
DB CPU		299.5		20.5	
library cache lock	46	124.1	2697	8.5	Concurrency
log file sync	181	9	50	.6	Commit
db file sequential read	120,270	6.3	0	.4	User I/O
local write wait	128	5	39	.3	User I/O
			528	.2	Configuration
			2	.1	Configuration
			1	.1	Concurrency
			2	.1	Concurrency

**Segments by Buffer Busy Waits**

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

Owner	Tablespace Name	Object Name	Subobject Name	Obj. Type	Buffer Busy Waits	% of Capture
SPC	TBSSPC	SPCT		TABLE	138,508	100.00

[Back to Segment Statistics](#)  
[Back to Top](#)

The first section provides an overview and the most significant diagnostics.

Additional pages show detailed statistical information for specific areas.

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

The first section of the AWR report provides a set of diagnostics that describe:

- Snapshot times
- Memory usage
- Load Profile
- Instance Efficiency percentages
- Shared Pool Statistics
- Top 10 Foreground Events by Total Wait Time
- OS statistics, CPU and memory used by the instance

The Top 10 Foreground Events by Total Wait Time section is a very good starting point for diagnosing performance problems. The purpose of the first section is to highlight the most significant issue. This sample shows a high percentage of DB time being spent on buffer busy waits.

The additional sections of the AWR report have detailed information that helps to diagnose the issues that are shown in the first section.

# 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
  - I/O
  - 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 a Statspack or an 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



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

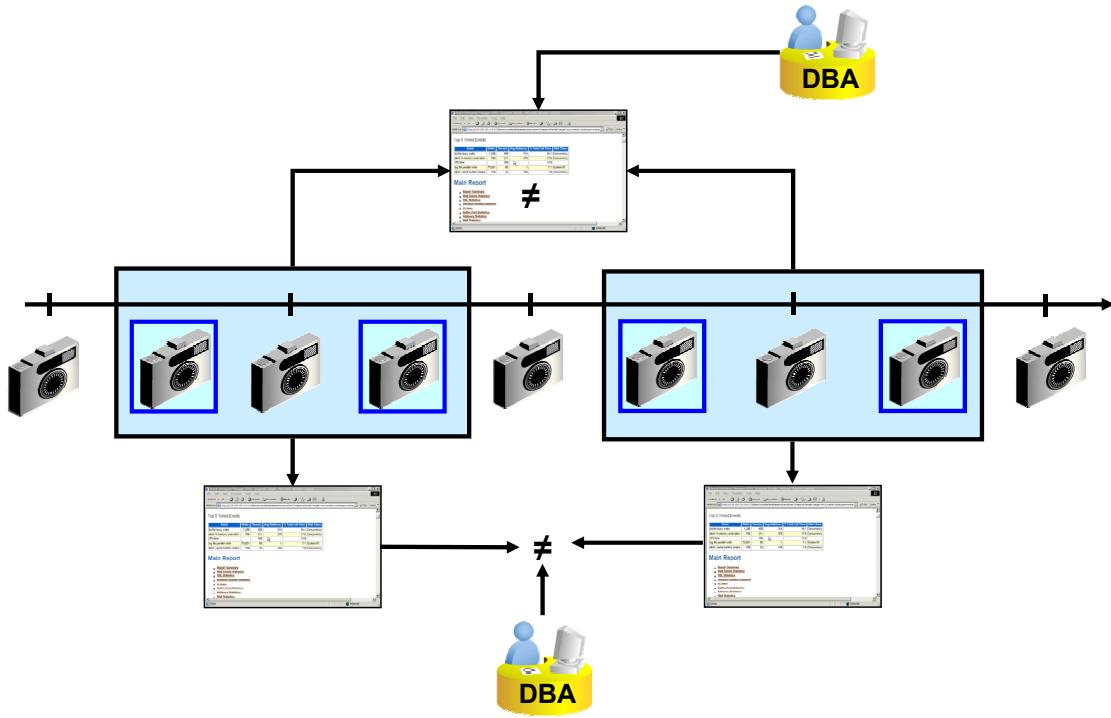
When diagnosing a possible database problem, start by checking the host hardware and OS, the middle tier or the network, and other resources outside the database. If these are not being completely consumed, it is time to look at the database.

## 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.

## Compare Periods: Benefits



**ORACLE®**

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

You can use the Workload Repository Compare Periods report to compare two periods in AWR. An AWR report shows AWR data between two snapshots (or two points in time), whereas the Workload Repository Compare Periods report shows the difference between two periods (or two AWR reports, which equates to four snapshots).

Use the Workload Repository Compare Periods report to identify detailed performance attributes and configuration settings that are different between two time periods. For example, if the application workload is known to be stable for a given time of day, but the performance on Tuesday was poor between 10:00 AM and 11:00 AM, then generating a Workload Repository Compare Periods report for Tuesday from 10:00 AM to 11:00 AM and Monday from 10:00 AM to 11:00 AM should identify configuration settings, workload profile, and statistics that were different between these two time periods. Based on the changes reported between these two time periods, the cause of the performance degradation can be accurately diagnosed. The two time periods selected for the Workload Repository Compare Periods report can be of different duration because the report normalizes the statistics by the amount of time spent on the server for each time period and presents statistical data ordered by the largest difference between the periods.

# Snapshots and Periods Comparisons

**Select Beginning Snapshot**

Go To Time: 5/15/13 12:00 PM Go  
(Example: 12/15/03)

Actions	Compare Periods	Go
Delete		
Select	ID	Capture Time ▲
	472	May 14, 2013 1:00:53 PM
		Collection Level
		TYPICAL

First Period Start   First Period End   Second Period Start   Second Period End   Review

**Compare Periods: Review**

Database: orcl

First Period	Beginning Snapshot ID: 480	Ending Snapshot ID: 481	Beginning Snapshot Capture Time: May 15, 2013 11:00:53 AM	Ending Snapshot Capture Time: May 15, 2013 12:00:01 PM
Second Period	Beginning Snapshot ID: 182	Ending Snapshot ID: 183	Beginning Snapshot Capture Time: May 2, 2013 11:00:06 AM	Ending Snapshot Capture Time: May 2, 2013 12:00:15 PM

Cancel Back Step 5 of 5 Finish

**ORACLE**

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

A Compare Periods operation enables you to define two different time periods and compare their respective AWR data sets.

From the Snapshots page, select the first snapshot of the first period. A wizard guides you to select the ending snapshot of the first period and the two snapshots of the second period. A review page is displayed as the last step of the wizard. Click finish to generate the Compare Periods report.

You can also generate a Compare Periods report over baselines that you have already defined. After at least two baselines are created, you can click the number of the baseline on the Automatic Workload Repository Baselines page. From this point, you can perform the Compare Periods operation. Follow the Compare Periods wizard to select both the baselines, and click Finish.

# Compare Periods: Results

Automatic Workload Repository > Compare Periods: Results

Logged in as SYS

**Compare Periods: Results**

**First Period**

Beginning Snapshot ID	280	Beginning Snapshot Capture Time	Jul 3, 2013 10:08:05 AM
Ending Snapshot ID	282	Ending Snapshot Capture Time	Jul 3, 2013 11:33:46 AM

**Second Period**

Beginning Snapshot ID	282	Beginning Snapshot Capture Time	Jul 3, 2013 11:33:46 AM
Ending Snapshot ID	283	Ending Snapshot Capture Time	Jul 3, 2013 12:18:27 PM

**General Report**

**View Data** **Per Transaction**

Per transaction comparison is appropriate when workload is the same.

Name	First Period Metric Ratio	Second Period Value	First Period Rate Per Transaction	Second Period Rate Per Transaction
DB cpu (seconds)		0.00		0.00
DB time (seconds)	1	2,773.45	1,747.35	9.80
db block changes	■	1,945,669.00	2,272,619.00	6,875.16
execute count	■	1,131,296.00	1,009,921.00	3,997.51

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

This slide shows a portion of the results of the Compare Periods operation, which identifies statistical differences between two snapshot periods. This report compares the same workload executed against different tablespace configurations over the elapsed time. Comparison can be made either on a per second or a per transaction basis. If the workload is the same or very similar in each period, a per transaction comparison is appropriate. The bar graphs indicate the proportional number for those metrics compared to the other time period. If the workload has changed, the Compare Periods report can help identify the changes.

Click the Report tab on this page to display an HTML report comparing the two periods, showing differences in areas such as wait events, OS statistics, services, SQL statistics, instance activity, I/O statistics, and segment statistics.

**Note:** If the sizes of the two time periods are different, the data is normalized over DBTIME before calculating the difference so that periods of different lengths can be compared.

# Compare Periods: Report

General Report

## WORKLOAD REPOSITORY COMPARE PERIOD REPORT

### Report Summary

Snapshot Set	DB Name	DB Id	Instance	Inst num	Release	Cluster	Host	Std Block Size
First (1st)	ORCL	1346382964	orcl		1 12.1.0.1.0	NO	EDRSR12P0	8192
Second (2nd)	ORCL	1346382964	orcl		1 12.1.0.1.0	NO	EDRSR12P0	8192

Snapshot Set	Begin Snap Id	Begin Snap Time	End Snap Id	End Snap Time	Avg Active Users	Elapsed Time (min)	DB time (min)
1st	280	03-Jul-13 10:08:05 (Wed)	282	03-Jul-13 11:33:46 (Wed)	0.3	85.7	24.4
2nd	282	03-Jul-13 11:33:46 (Wed)	283	03-Jul-13 12:18:27 (Wed)	0.4	44.7	17.3
%Diff					39.3	-47.9	-28.9

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

When you click the Report tab on the Compare Periods: Results page, you generate the Workload Repository Compare Period report. This report contains the same sections as the ones you can find in a Statspack or an AWR report. In addition, the Compare Period report shows you a configuration comparison for both time periods.

The other sections of the report have more detailed information about various performance areas that you will use when the main section indicates that there is a problem in that area.

**Note:** You can also generate a report with the same information by using the `awrddrpt.sql` script located in your `$ORACLE_HOME/rdbms/admin` directory.

# Quiz

The Automatic Workload repository exists in the SYSAUX tablespace. The persistent portion of AWR is the snapshots. The information included in each snapshot is controlled by:

- a. The size of the SYSAUX tablespace
- b. The setting of the STATISTICS\_LEVEL parameter
- c. The snapshot retention time
- d. The snapshot interval



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

## Answer: b

All of these answers affect the amount of space used in the SYSAUX tablespace. But only the setting of the parameter STATISTICS\_LEVEL changes what information is collected.

## Summary

In this lesson, you should have learned how to:

- Create and manage AWR snapshots
- Generate AWR reports
- Create Compare Periods reports



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

## **Practice 3 Overview: Using AWR-Based Tools**

This practice covers the following topics:

- Creating and managing Automatic Workload Repository (AWR) snapshots
- Generating and viewing the sections of an AWR report
- Generating and viewing the sections of a Compare Periods report



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

# **Defining the Scope of Performance Issues**



**ORACLE®**

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

# Objectives

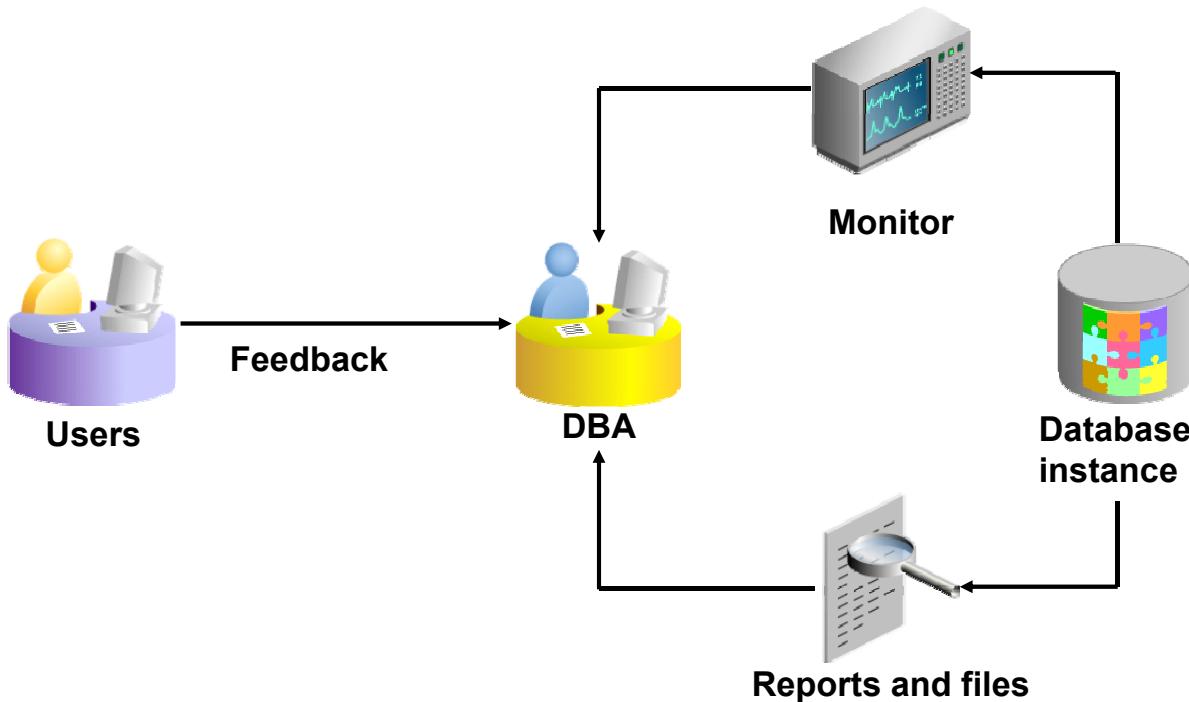
After completing this lesson, you should be able to do the following:

- Identify performance issues
- Set tuning priorities
- Interpret tuning diagnostics
- Tune for life-cycle phase



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

## Defining the Problem



ORACLE

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

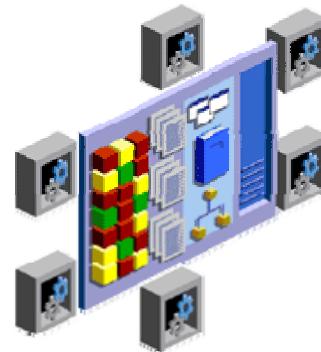
Problems can arise at any time. A proactive DBA watches for problems and corrects them before they are noticed by users. In the past, the discovery and definition step has been tedious and frequently dependent on listening to user feedback. User feedback is important, but is often subjective and not reproducible. To quickly identify the problem the DBA will:

- Monitor the current state of the database instance and compare it to a previous state.
  - Collect performance metrics regularly. Changes can point to issues before they become noticeable to users.
  - Use OS or Enterprise Manager tools to check for CPU and disk queuing, disk utilization, and memory swapping. These are the signs of an overloaded system.
- Examine AWR or Statspack reports and instance files carefully.
  - Identify SQL statements in the applications that are consuming the most resources. Have these changed?
  - Check the alert logs, and trace files for error messages that might give insight. Do not overlook system- and application-specific logs.
  - Ensure that the initialization parameter settings make sense for the system.
  - Collect instance and OS statistics. Use reports that show where the greatest waits and the greatest use of resources occur. ADDM goes further by focusing on those components with the greatest potential benefit.

# Limit the Scope

Where is the problem?

- Application (SQL)
- Instance
- Operating system



ORACLE

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

Does the performance issue originate in the operating system (OS), the instance, or the application SQL? This question is not always easy to answer. Poorly performing SQL can cause excessive physical reads and writes, appearing to be an I/O issue. Improperly sized memory components (an instance configuration issue) can lead to excessive swapping in the OS. Poor disk configuration can appear to be an instance configuration problem, causing a large redo file waits or commit waits, and other problems.

Eliminate possibilities. When the instance appears to have I/O problems, compare the instance file I/O statistics with OS-level statistics. The differences can guide you to the actual problem. For example, a higher than normal average wait time on a particular tablespace could be due to:

- **Hardware:** A file is on a slow drive or an improper RAID configuration.
- **OS:** The OS is busy with other files on the same drive or partition.
- **Instance:** The tablespace was created with properties that are different from what other tablespaces have, other busy database files are on the same disk or partition (the database I/O is not balanced across all the drives), or the objects being accessed are mostly in the same tablespace, file, or disk.
- **Application:** The application is doing excessive I/O due to poor access path choice by the optimizer due to out-of-date statistics, inefficient indexes, or other reasons.

Determine the scope of the problem to focus your efforts on the solutions that provide the most benefit.

# Determining Tuning Priorities

Choose the problem that has the greatest impact:

- Analyze system performance in terms of work done (CPU or service time) versus time spent waiting for work (wait time).
- Determine which component consumes the greatest amount of time.
- Drill down to tune that component, if appropriate.



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

Determine which problem to tune first. In the performance reports, you see many statistics; even a well-tuned database shows a set of top wait events. The Oracle server provides a set of wait event statistics for processes that are idle or waiting. The Oracle server also records CPU utilization for processes that are running. To determine the impact of a particular event, it must be compared with the overall time spent.

Each request to the database server has a response time consisting of a wait time and a service time. The service time is the time spent actively working on the request (CPU time). The wait time is by definition the time waiting for any reason. Both service time and wait time can be tuned. To tune the service time, something has to change: the processing, the SQL, the access path, or the data storage structure. Wait times can be tuned by reducing contention for the resource where the wait is occurring.

Each server process is typically in one of three states:

- **Idle:** Waiting for something to do (sleeping)
- **Running code:** Using the CPU or in a run queue
- **Waiting (blocked):**
  - For some resource to become available
  - For a requested activity to complete

# Common Tuning Problems

The most common tuning problems:

- Inefficient or high-load SQL statements
- Suboptimal use of Oracle Database by the application
- Undersized memory structures
- Concurrency issues
- I/O issues
- Database configuration issues
- Short-lived performance problems
- Degradation of database performance over time
- Unexpected performance regression after environment changes
- Locking issues



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

Tuning inefficient or high-load SQL statements has a wide impact that can reduce memory use, CPU, and I/O resources. SQL tuning issues include poorly written SQL, ineffective use of indexes, access path costs, and sorting. In this course, we assume that the DBA has little or no opportunity to change the SQL statements. This course covers the SQL problems that a DBA can tune without changing the SQL. There is another course dedicated to SQL tuning primarily for developers, *Oracle Database 12c: SQL Tuning Workshop for Developers*.

Problems such as establishing new database connections repeatedly, excessive SQL parsing, and high levels of contention for a small amount of data (also known as application-level block contention) can degrade the application performance significantly. These are all poor use of the database by the application, and require a change in the application design.

Memory issues are high on the list of instance tuning problems. Proper sizing of the System Global Area (SGA) including the Shared pool and Buffer cache, and the Process Global Area (PGA) reduces contention for memory resources and indirectly reduces I/O and CPU.

A high degree of concurrent activities, multiple processes, or users might result in contention for shared resources that can be manifested in the forms of various types of waits. Many resources can be accessed only by one process at a time. Several processes attempting to access the same resource creates contention.

In any database, I/O issues, such as database file layout on disk or RAID devices, can be a source of performance problems. In OLTP applications, the amount of redo and undo generated can create bottlenecks in memory or I/O.

Some problems are reported by users but may not be apparent from reports that span intervals of 30 minutes or longer. The Oracle Database has additional tools (Active Session History ASH, and Real-Time ADDM) that allow the DBA to view statistics and metrics over small segments of time in the recent past.

Many databases will have gradual changes: the number of user, the amount of data, the number of reports, and modules in use. These changes may lead to degradation in performance. The proactive DBA will capture and save statistics sets from when the database is performing acceptably, to compare with statistics when the database performance is poor to identify the differences.

The environment of the database is seldom static. Patches, upgrades, new hardware, or changes to the instance parameters can change the performance of the database.

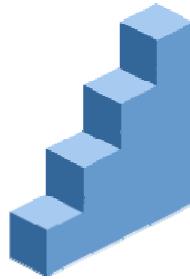
Sometimes a change improves performance in one area and causes another area to degrade.

Locking issues are not common problems, but when you have locking issues, they become very important.

# Tuning Life Cycle Phases

An application's life cycle can be divided into different phases:

- Application design and development
- Testing: Database configuration
- Deployment: Adding a new application to an existing database
- Production: Troubleshooting and tuning
- Migration, upgrade, and environment changes



ORACLE

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

Tuning will follow the general methodology in all of the life-cycle phases. Different phases of the life cycle will have somewhat different approaches.

## **Development: Application Design and Programming**

Whenever possible, you should start tuning at this level. With a good design, many tuning problems do not occur. Use a development and test database instance for proof of concept, and to check the performance of various design alternatives.

## **Testing: Database Configuration**

The testing phase is a continuation of development, with more realistic tests that use production hardware and operating system.

## **Deployment: Adding a New Application to an Existing Database**

When adding a new application to an existing system, the workload changes. You should accompany any major change in the workload with performance monitoring.

## **Production: Troubleshooting and Tuning**

Follow the methodology. Use a test instance to determine whether the solution eliminated the bottleneck.

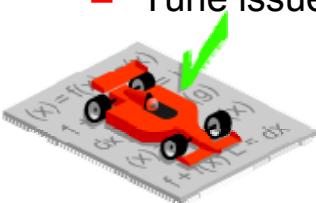
## **Migration, upgrade, and environment changes**

A change of OS or database version requires thorough testing to avoid performance issues.

# Tuning During the Life Cycle

Tuning is of two types:

- Proactive (Make it better, so it will not break.)
  - Test scenarios.
  - Find the problem areas.
  - Resolve the problem.
- Reactive (Wait until it breaks; then fix it.)
  - Monitor active instance.
  - Tune issues as needed.



Proactive



Reactive



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

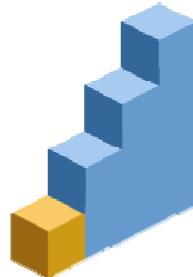
Tuning during the life cycle involves two courses of action: proactive tuning or reactive. During the design, development, and testing phases, tuning is mostly proactive (that is, scenarios and test cases are designed and tested). The results are measured and compared against other configurations. In deployment and production environments, tuning is mostly reactive. The need for hypothetical loads is removed because actual users and workloads are created, but the ability to anticipate problems also diminishes. You can monitor the database instance to observe changes and trends in performance metrics. From the information you gather by monitoring, you may be able to mitigate performance issues before they are noticed by users.

The DBA may be involved in tuning from the earliest stages of design and development. It is less expensive to correct bugs and performance problems early in the life cycle than later. The differences in tuning the later phases of the life cycle are primarily in what is allowed. Many DBAs in a production environment are not allowed to change the SQL or data structures. However, a design change to improve performance may warrant a change request to the application vendor or development team.

# Application Design and Development

The application can be tuned, even in the design and development phases, by building and tuning test cases.

- Check normalization against major functions.
- Check data structures against access times.
- Look at the points where processes are serialized.
- Tune the major reports.
- Tune the high-volume processes.



ORACLE

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

The tuning methodology that you follow during the design and development phases focuses on the common bottlenecks of any system: normalization, data structures, serialization points, major reports, and high-volume requests. Very early in the design, the major functions of the application are known. The level of normalization of the data has serious performance consequences. Over-normalization can lead to large multiway joins that can use all of the available host resources. Under-normalization brings another set of problems: inconsistent data, complex data checking, and difficulty migrating data to other schemas or databases. The solution requires a fully normalized design and careful denormalization with built-in consistency checks.

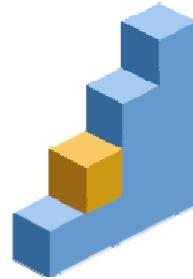
Choosing the proper data structures, such as partitioned tables for large data sets, can provide large performance benefits. The design should avoid resource contention to increase scalability. The major reports required for the application should be prototyped and tuned for expected run times. High-volume functions, in terms of either data or executions, should also be prototyped.

Each of these potential bottlenecks will have test cases. These test cases are tuned with the same methodology that is used in the production database: collect statistics, tune the bottleneck, test, and repeat until the goal is met.

# Testing: Database Configuration

The testing phase allows tuning at a deeper level.

- Check physical layout.
- Monitor for resource contention.
  - Memory utilization
  - Locks
  - Disk hot spots
- Test for resource exhaustion.



ORACLE

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

The test phase allows you to test at a deeper level. The test case should exercise the application functions, expected loads, and stress tests of improbable loads. These kinds of tests can give valuable insight into the best physical layouts, and the best OS and hardware configurations. It is important to monitor hot spots, even on fast disks. You should plan the data configuration to enable a shorter recovery time and faster data access. Incorporate the business requirements for recovery time and availability as much as possible to allow for the overhead of these requirements.

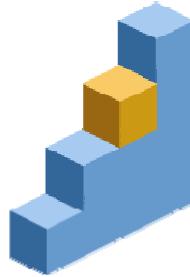
Test with loads that exhaust the machine resources. These tests identify the most used resources. These are the resources that limit the scalability of the system.

The DBA uses the time model and wait events at each phase to identify the bottlenecks and tuning sessions to fix the bottlenecks at each level.

# Deployment

Deployment of:

- New application and database
  - Take baseline.
  - Monitor growth and performance.
- New application in existing database
  - Take baseline before deployment.
  - Take baseline after deployment.
  - Compare baselines.



ORACLE®

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

When a new application is initially deployed, the performance expectations are often different than reality. There are two variations to consider here: the first for a new application on a new database, and the second for an application added to an existing database.

The new application on a new database has no baseline, so the tuning is based on current performance. Generate regular performance reports and save them as baselines. As the application grows in data set size or number of users, compare new performance reports with the previous reports. This allows you to tune before the performance degrades to an unacceptable level.

When a new application is added to an existing database, compare baseline performance reports from before and after the application is deployed. These reports show the resources that the new application is using and possible contention for resources with the existing applications.

# Production

Tuning is reactive. You need to know:

- *What has changed?*
- *Where is the baseline?*



ORACLE

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

Tuning the other phases in the application life cycle is generally proactive. You are looking for possible problems before they are apparent to users.

Tuning a production database is very often reactive tuning. Something has gone wrong: A report that ran in minutes is now taking hours, response time has increased and the users are complaining, backups are not finishing in the allotted time.

Something has changed: Are there more users? Is there a new report or application running? Has something in the OS changed?

Tuning a production system that previously ran acceptably, and now has a problem, depends on knowing or deducing what has changed. Compare the baseline statistics report taken when the database was running acceptably with a new report taken while the problem is visible. The differences should be visible.

Tuning a production database when there are no baseline statistics is more difficult but possible. The same methodology is used, and the prioritized components are tuned. The time model is used to identify the problem. Possible solutions are considered, starting with design in the top-down steps. A tuning session is then used to test the solutions.

# Migration, Upgrade, and Environment Changes

1. Create a test system.
2. Capture a representative workload.
3. Execute the workload against the test system.
4. Collect statistics.
5. Reset the test system, and make a change.
6. Execute the workload.
7. Capture statistics and compare.



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

Frequently the environment of the database instance must change in ways that impact performance. The hardware must be upgraded or replaced. The operating system is upgraded or changed to another OS. The database software is upgraded to a new revision. The disk subsystem is modified or replaced. Even smaller changes, such as OS parameter and database parameter changes, can have significant effects.

In general, you want to test the current workload on the target system before the change occurs in the production system. The difficulty comes in capturing a representative workload, and replaying the workload on a test database.

Oracle Database Enterprise Edition has the Real Application Testing option, which includes two tools, SQL Performance Analyzer and Database Replay, for overcoming these difficulties. This topic is covered in more detail in the lessons titled “Using SQL Performance Analyzer” and “Using Database Replay.”

## ADDM Tuning Session

An ADDM tuning session follows the same procedure as a manual tuning session, but combines the steps.

ADDM Tuning Session	Manual Tuning Session
Generate the ADDM report.	Collect current statistics. Compare current statistics with a previous set; look up in a performance-issues knowledge base. Define the problem and make recommendations.
Review the recommendations.	Build a trial solution.
Implement the recommendations.	Implement and measure the change.
Review the next ADDM report.	Decide: "Did the solution meet the goal?"



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

Automatic Database Diagnostic Monitor (ADDM) internally follows the steps of the manual tuning session. The steps that you follow while using ADDM are shown in the slide. The manual steps are shown as substeps to the ADDM steps.

ADDM consolidates the manual tuning session to quickly identify the areas that produce the greatest benefit. ADDM runs automatically each time an AWR snapshot is taken. The number and severity of findings in the latest ADDM report are displayed on the Database Home page in Enterprise Manager Cloud Control.

# Performance Versus Business Requirements

Factors that affect performance:

- Frequent checkpointing
- Performing archiving
- Block check sums
- Redundancy
  - Frequent backups of data files
  - Multiple control files
  - Multiple redo log members in a group
- Security
  - Auditing
  - Encryption
  - Virtual Private Database/Fine Grained Access Control



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

There is always a cost of doing business in certain ways. Often the requirements of the business can impact performance. The impact of down time and crash recovery, and even the unlikely event of block corruption, must be considered against the overhead of protecting against these events. Data Compression, National Language settings, and security requirements are all business requirements that can impact performance. Because these are business requirements, you configure the database for the business, and then tune your database accordingly. The uptime requirement, the mean time to recovery, and the amount of data that could be lost in a disk or system crash are all business issues.

Redundancy improves availability, but requires more I/Os: Oracle recommends that there are at least two control files, and one is required. Many DBAs use three or four. More control files require more writes, thus more overhead. Multiple redo log members reduce the chance of loss of data due to a disk failure, but increase the overhead of writing redo. Frequent checkpoints reduce the mean time to recovery, but can increase the number of physical writes. Each item listed in the slide has a performance cost associated with it.

Security measures are often required and each has some impact on performance, even if it is a very small impact. The question is not whether to use the security features but what is required, and use only those features.

# Performance Tuning Resources

Oracle provides a large set of resources for problem solving:

- Documentation
- My Oracle Support
- Forums
- Performance Service requests



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

Oracle provides a large set of performance tuning resources. The first and most accessible is the Database documentation set. The *Oracle Database Performance Tuning Guide* covers a wide range of performance tuning issues and the steps to find a solution. The *Oracle Database 2 Day + Performance Tuning Guide* is an overview of the Oracle performance tuning methodology and a task-oriented approach to tuning common problems.

My Oracle Support (MOS) documents are available to all who have subscribed to support services. A Customer Support Identifier is required to set up an MOS account. MOS has a large repository of documentation that describes best practices, how-to's, and diagnosis of bugs and problems. The Center of Expertise offers *Note: 390374.1 Oracle Performance Diagnostic Guide* to assist customers with classification and tuning of various performance issues.

Oracle Support will accept performance service requests. Oracle Configuration Manager (OCM) is a tool that is available through MOS and integrated with Enterprise Manager, to assist customers and Oracle Support to get problem-solving information quickly. OCM also provides proactive alerts concerning database configuration and software updates.

Finally, community discussion forums are publicly available on Oracle Technical Network.

# Filing a Performance Service Request

File a performance service request.

- Is the problem instance-wide or query specific?
- Identify the root cause.
- Provide Statspack or AWR reports, and OS statistics.
- Provide Remote Diagnostics Agent (RDA) reports.
- Provide `SQL_TRACE` reports.



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

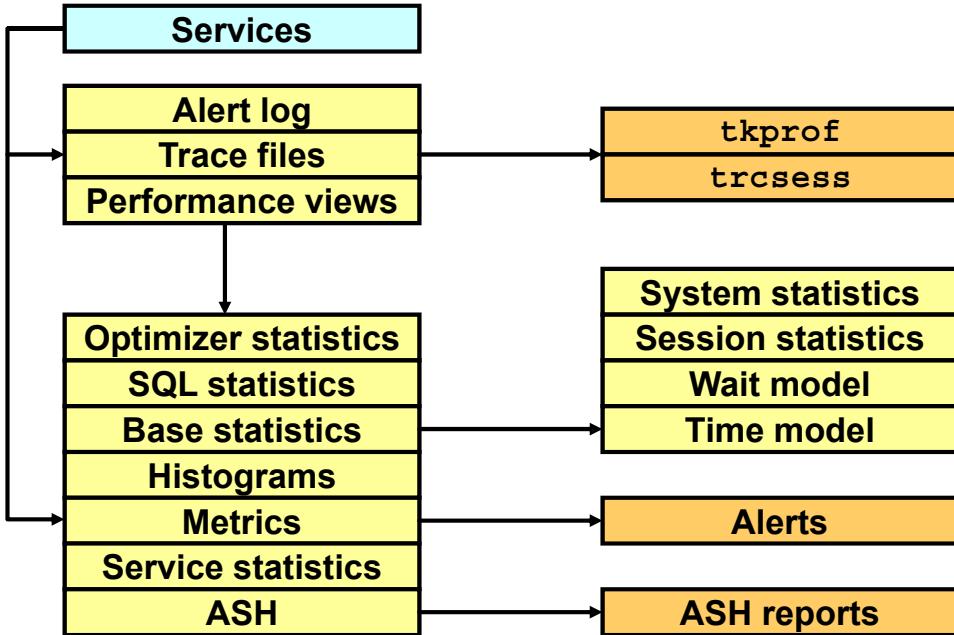
The performance problems you encounter are often unique to your application and database configuration. Occasionally, the problem is something that you cannot tune.

Oracle Support Services (OSS) provides a document, “Note: 210014.1 How to Log a Good Performance Service Request,” to guide you in logging a performance service request (SR).

OSS needs certain information:

- Is the problem instance-wide or query-specific? When does the problem occur? What is working and what is not? Give examples of both acceptable performing SQL and poorly performing SQL.
- What is the root cause? Create Statspack or AWR reports when the performance is “good” and when it is “bad,” and then compare them. Check the OS, network, and database log files for clues. Remove recent changes one at a time and record the results; even things that do not change the problem help narrow the search for a root cause. You may not be able to determine the root cause.
- Provide Statspack or AWR reports and OS statistics during the time when the database is exhibiting the problem, and a baseline when the problem does not appear, if possible. Take short-period snapshots when the problem is evident.

# Monitoring and Tuning Tools: Overview



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

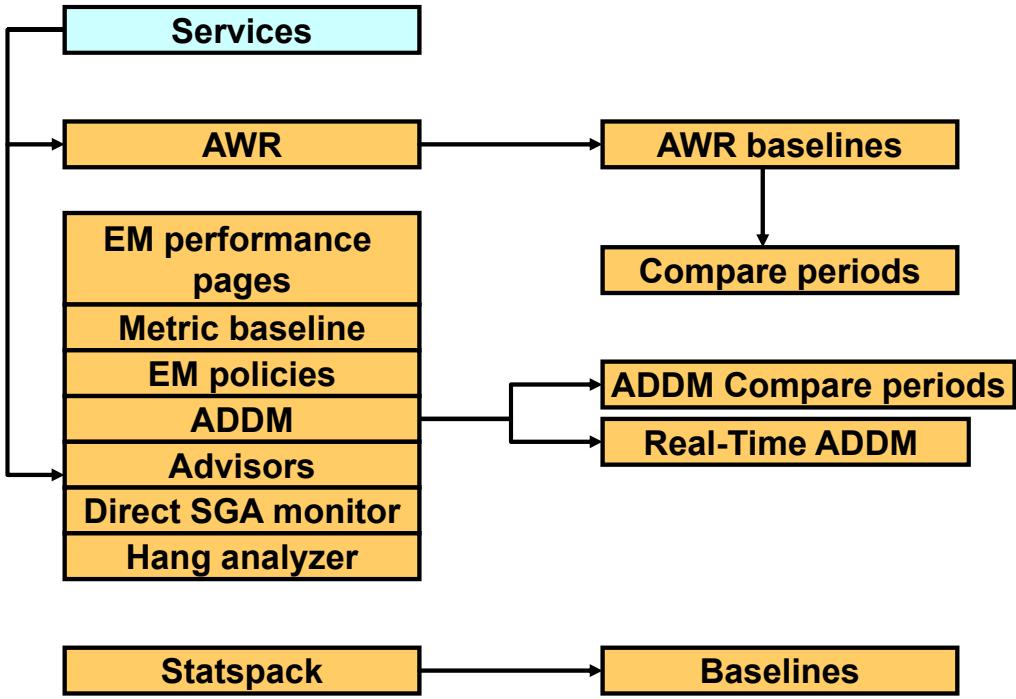
In the following lessons, you will use many of the tools listed in the slide. The light-colored boxes indicate tools that contain raw data elements. The darker boxes are tools that have used the raw data to derive more useful information. Often, the information is formatted in reports such as the Active Session History (ASH) report.

Performance views is another name for the dynamic performance views, or v\$ (v-dollar) views, that hold raw statistics in memory.

Trace files are very difficult to interpret and are primarily for use by Oracle Support. Only certain traces can be formatted with the `tkprof` utility. The `trcsess` utility is a unique tool for combining and filtering trace files to extract the statistics for a single session, service, or module across multiple trace files. The use of these tools is covered in the lesson titled “Monitoring Applications.”

The Services box indicates that the direction of performance monitoring is organized around services. Statistics are aggregated by services and several reports can report by services. Statistics gathered by services rather than schema, instance, or session can provide a unique view of application performance.

# Monitoring and Tuning Tools: Overview



ORACLE®

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

The tools listed in this slide format the data to make it more useful. Several of the tools analyze the data to provide proactive problem detection and recommendations. In the following lessons, most of the tools will be explained and used.

## Quiz

The database is performing poorly. Users are complaining of slow response times since the month-end processing started. You check the AWR report. The top timed events show I/O- and CPU-related events. Next, you check the time model. It shows SQL execute elapsed time as the major component of DB Time. What should be your next diagnostics step?

- a. Check SQL statistics in AWR or Statspack report.
- b. Use SQL Monitoring to find long-running SQL.
- c. Review Automatic Database Diagnostics Monitor (ADDM) recommendations.
- d. Use Remote Diagnostics Agent to check database configuration.
- e. Use OS tools to look for runaway processes.



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

### Answer: c, a, b

The symptoms point to an application issue with month-end processing. Some long-running SQL might be consuming large amounts of database resources.

The ADDM recommendations should point to the problem possibly pointing to poorly performing SQL.

The SQL statistics section of either AWR or Statspack report will show the top SQL statement consuming database resources.

SQL Monitoring (covered in the lesson “Real-Time Database Operation Monitoring” ) can quickly identify long-running SQL, both currently running and recent ended.

## Summary

In this lesson, you should have learned how to:

- Identify performance issues
- Set tuning priorities
- Interpret tuning diagnostics
- Tune for life-cycle phase



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

## **Practice 4 Overview: Identifying the Problem**

This practice covers using Enterprise Manager to identify an operating system problem.



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



# Using Metrics and Alerts

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

ORACLE®

# Objectives

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

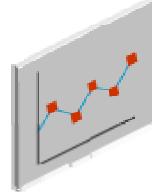
- View metrics by using the metrics history views
- Create metric thresholds
- View alerts



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

# Metrics and Alerts

- Metric: Rate of change in a cumulative statistic
- Alert: Event generated when a monitored metric crosses a threshold



ORACLE®

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

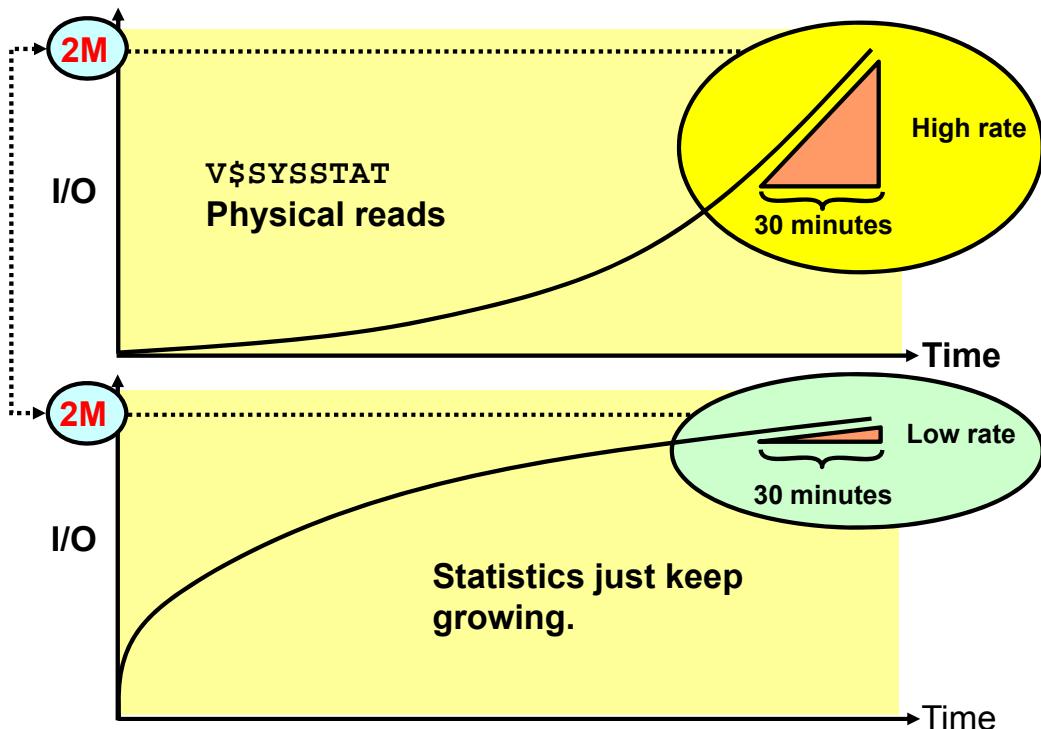
## Metrics, Alerts, and Baselines

Monitoring for performance requires certain information that goes beyond statistics. To determine whether a particular statistic is important, you need to know how much it has changed over a certain period of time. To be proactive, you need to be notified when certain conditions exist—for example, when system response time approaches the agreed maximum. To diagnose performance issues, you need to know what has changed. Metrics and alerts provide part of this information.

In general, a metric is a timed rate of change in a cumulative statistic (for example, physical reads per second). But there are other metrics that are based on events such as “tablespace full”, or errors such as “snapshot too old.”

Threshold values can be set for various metrics, and when the value of the metric crosses the threshold value, an alert is generated.

## Limitation of Base Statistics



ORACLE®

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

Statistics are counters of events that happen in the database. Statistics and wait events are the raw data. Base statistics are always only a value at a given time.

If you chart the base statistic values over a long period of time, you can see that the values just keep growing over time. The slide illustrates a possible example of such a graph for the physical reads statistic extracted from the `v$SYSSTAT` performance view. As you can see in the slide, although the statistic value is the same for both graphs, at the end of the observation period, the trend is completely different. In the upper graph, it is clear that your database is experiencing a much higher rate than in the lower graph. So, just looking at the statistic value is meaningless.

To get a better understanding of your database's behavior, you need to be able to see the curves or trends, not just the values. Thus, you need to compute statistic rates over a period of time to determine the trend over that period.

## Typical Delta Tools

- Comparison of statistics at two points in time is needed.
- The following tools produce deltas:
  - AWR reports
  - Statspack
  - Customized scripts



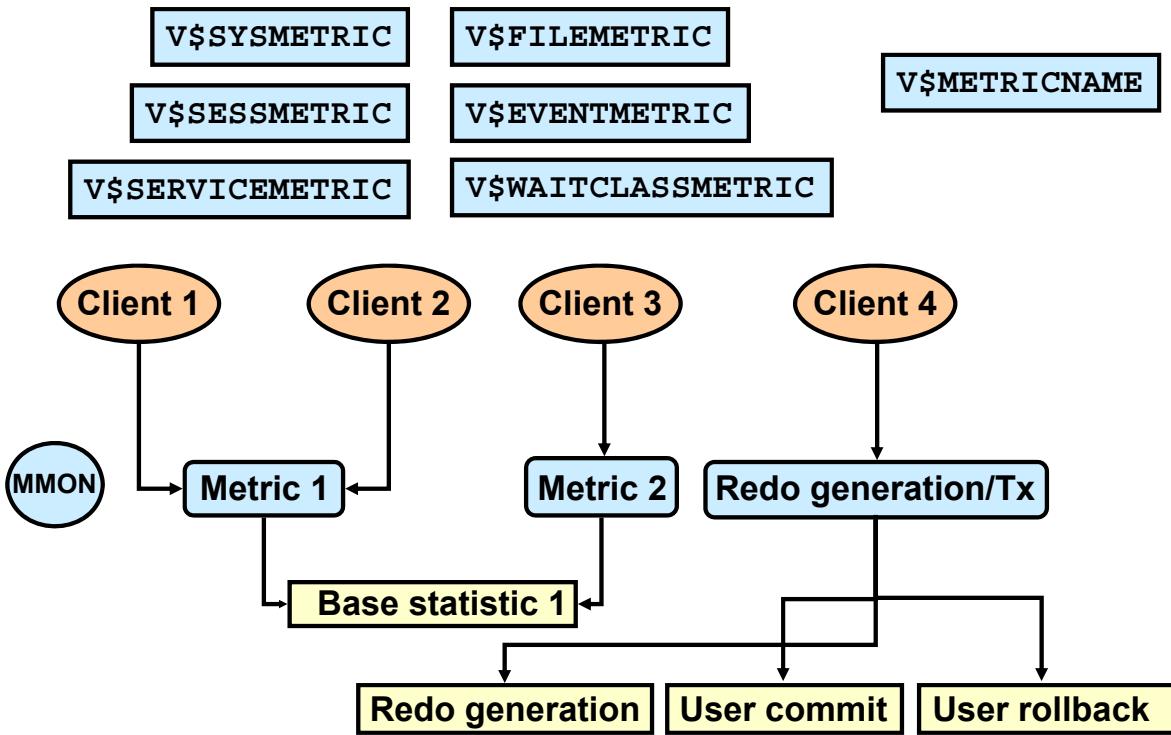
**ORACLE**

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

Base statistics are just raw numbers that accumulate from the instance startup. A snapshot is a set of statistics captured at a point in time. You can obtain meaningful statistic values taking snapshots at different times and taking a difference of the values. The difference is a delta. Several tools can produce a delta. Statspack, Automatic Workload Repository (AWR), and customized scripts can produce reports of the delta over two snapshots.

AWR and Statspack allow you to save snapshot sets for future reference.

# Oracle Database Metrics



**ORACLE®**

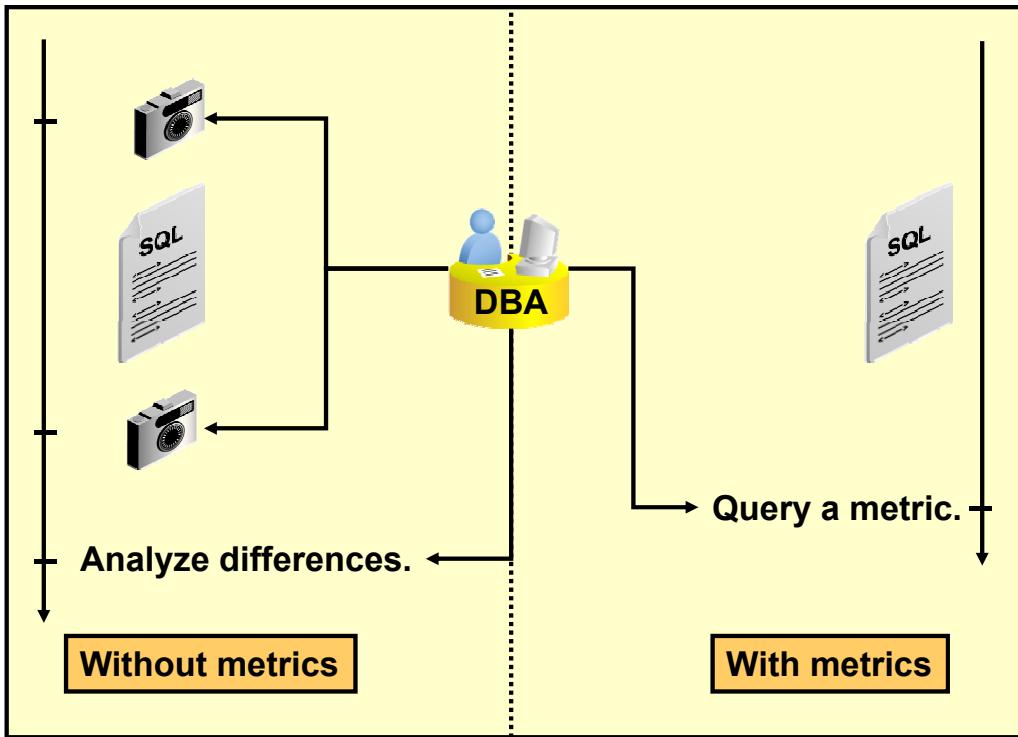
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Oracle Database server collects base statistics during normal operations. Base statistics are simple counts. For example, counting the number of physical reads in the system since startup is a base statistic.

Metrics are secondary statistics derived from base statistics. Most metrics track the rates of change of activities in the Oracle server. For example, the average physical reads in the system in the last 60 minutes is a metric. Metrics are used by internal components (clients) for system health monitoring, problem detection, and self-tuning. The Manageability Monitor process (MMON) periodically updates metric data from the corresponding base statistics.

The Oracle Database server components use metrics to perform manageability functions. For example, Automatic Database Diagnostics Monitor (ADDM) uses the average physical reads in the system in the last 60 minutes as input. Another component may need a different metric based on the same base statistic, physical reads. For example, the memory advisor may need the physical read counts during peak hours. Oracle Database supports metrics for system, sessions, files, and wait event statistics. Each metric is uniquely identified by a metric number and is associated with a metric name. The diagram lists some of the fixed views that you can access to browse metric data. For more information about these views, refer to the *Oracle Database Reference* guide.

## Benefits of Metrics



ORACLE®

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

The main benefit of keeping metrics is that when a component needs to compute the rate of change of some activity, the data is readily available.

In earlier releases, you had to capture statistics before and after running a workload to compute the changed rate for a particular base statistic. With metrics, all you need to do is run your workload and select the corresponding metrics.

Server components now have a basis for self-tuning and health checks with the server-collected metrics. Metrics provide the performance information required for the Automatic Memory Management and Automatic Database Diagnostic Monitor.

# Viewing Metric History Information

- In memory:
  - V\$SYSMETRIC\_HISTORY (last hour)
  - V\$FILEMETRIC\_HISTORY
  - V\$WAITCLASSMETRIC\_HISTORY (last hour)
  - V\$SERVICEMETRIC\_HISTORY
  - V\$SESSION\_WAIT\_HISTORY (last 10 events)
- On disk:
  - DBA\_HIST\_SYSMETRIC\_SUMMARY
  - DBA\_HIST\_SYSMETRIC\_HISTORY
  - DBA\_HIST\_SESSMETRIC\_HISTORY
  - DBA\_HIST\_SYSTEM\_EVENT (cumulative)
  - DBA\_HIST\_FILEMETRIC\_HISTORY (alerts only)
  - DBA\_HIST\_FILESTATXS (cumulative)
  - DBA\_HIST\_WAITCLASSMET\_HISTORY (alert)



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

Metric values are exposed in some V\$ views, where the values are averages over fairly small time intervals. The intervals can vary depending on the class of the metric from 15 seconds to 10 minutes. Snapshots of the data in the V\$ views persist in the DBA\_HIST tables.

The slide lists a few of the metric and AWR views.

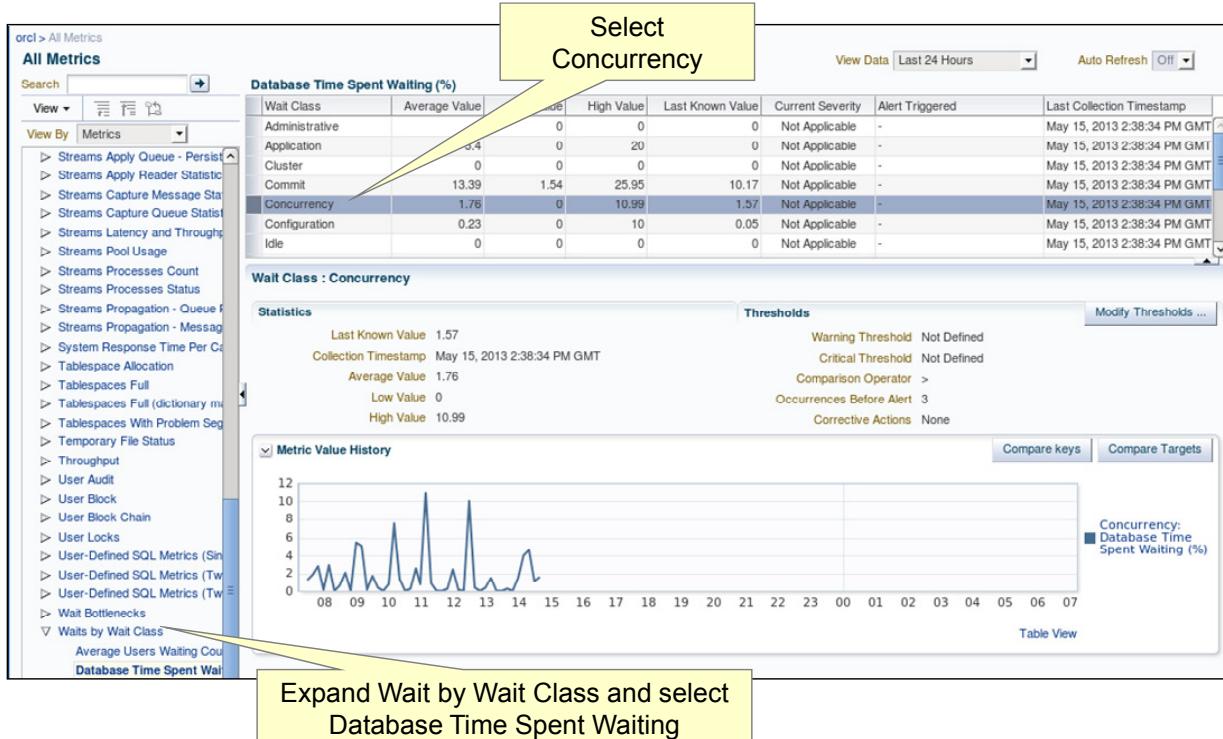
Example:

- V\$SYSMETRIC\_HISTORY displays all system metric values available in the database. Both long duration (60 seconds with 1 hour history) and short duration (15 seconds with one interval only) metrics are displayed by this view.
- DBA\_HIST\_SESSMETRIC\_HISTORY displays the history of several important session metrics. It contains samples (snapshots) of the V\$SYSMETRIC\_HISTORY view.

**Note:** Accessing the DBA\_HIST\_\* views requires a license for the Diagnostics Pack.

For more information about these views and the column details, see the *Oracle Database Reference*.

## Viewing Metric Details



**ORACLE®**

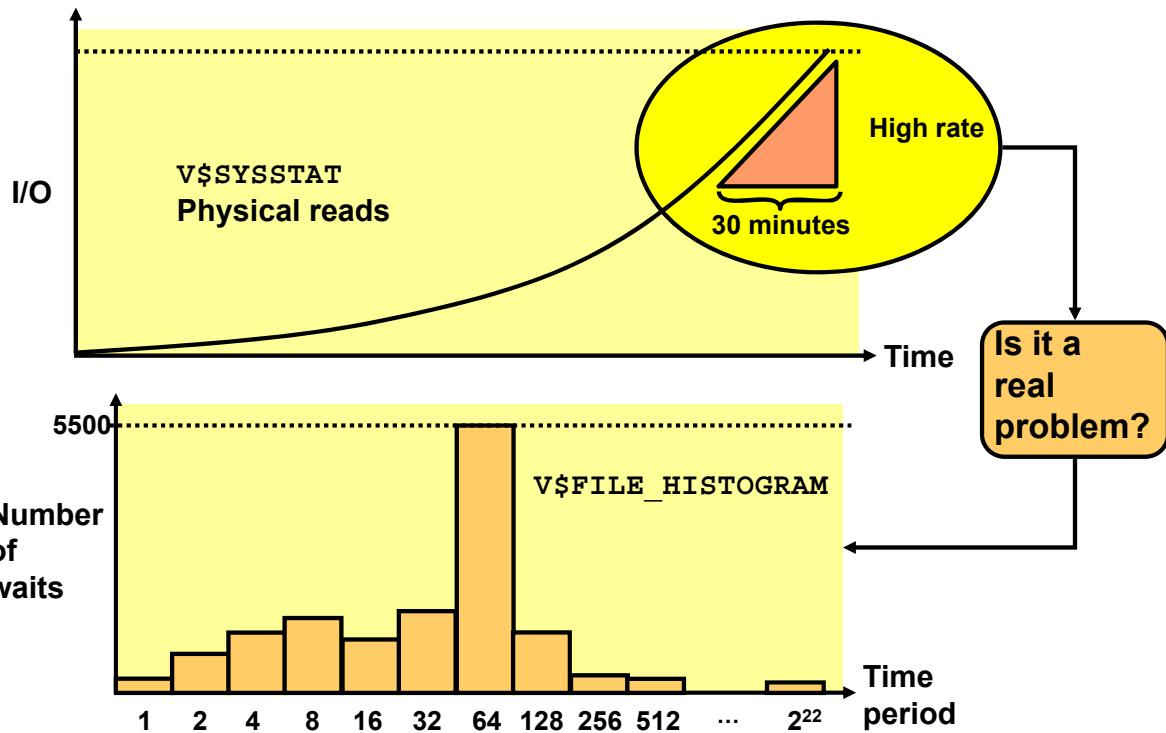
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Use the All Metrics page to view a list of all the performance metrics available for your database. You can access this page from the Database Home page by clicking the All Metrics link in the Related Links section.

On the All Metrics page, you can expand all or specific metric groups to see particular metrics. By selecting one metric, you display that metric's page. You can view the metric's value over a certain period of time; this view can be customized. The corresponding graphic shows you the metric's value history.

To access the page shown from the Database Home page, expand the Oracle Database menu. Expand Monitoring and select All Metrics. On the All Metrics page, expand Waits by Wait Class, then click Database Time Spent Waiting(%). On the Database Time Spent Waiting(%) page, click Concurrency.

## Statistic Histograms



**ORACLE**

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

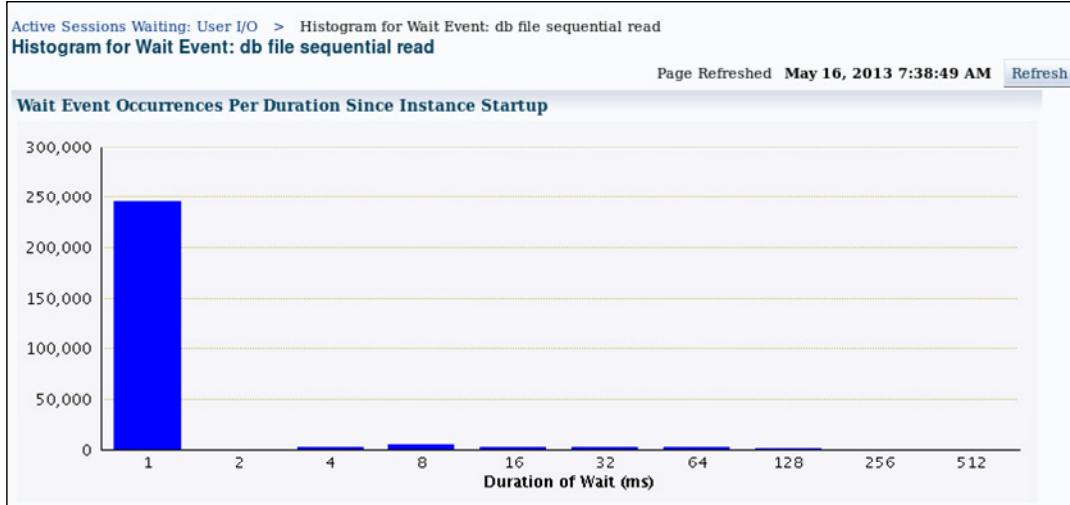
Although the metrics can give you an idea of the trend for particular statistics, they do not tell you if a particular bottleneck is affecting the whole system or if it is just localized. As an example, you can observe a high metric rate, but this sudden increase could be localized to only one or two sessions in your system. In this case, it might not be worth investigating the issue. However, if the sudden increase is generalized to the whole system, you need to investigate further. This information is available via histogram performance views.

As shown in the slide, you observe a sudden increase in your I/O rate. You can correlate this information to the corresponding I/O histogram found in `V$FILE_HISTOGRAM`. This view displays a histogram of all single block reads on a per-file basis. The histogram has buckets of time intervals, measured in milliseconds, from 1 ms up to  $2^{22}$  ms (69.9 minutes). The value in each bucket is the number of times the system waited for that amount of time. For example, you can see from the slide that the system waited 5,500 times for more than 32 ms and less than 64 ms to read blocks from disks. This is certainly a cause of concern for your system if the access times are normally less than 10 ms, and you should investigate this further. Had you seen large numbers in shorter wait time periods, you would not have worried much.

The metrics are able to alert you to a potential problem. By drilling down using histograms, you can clearly determine whether there really is a problem.

## Histogram Views

- **V\$EVENT\_HISTOGRAM:** For each event such as “db file sequential read”
- **V\$FILE\_HISTOGRAM:** For single block reads on a per data file basis



**ORACLE**

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

V\$EVENT\_HISTOGRAM displays a histogram of the number of waits on an event basis.

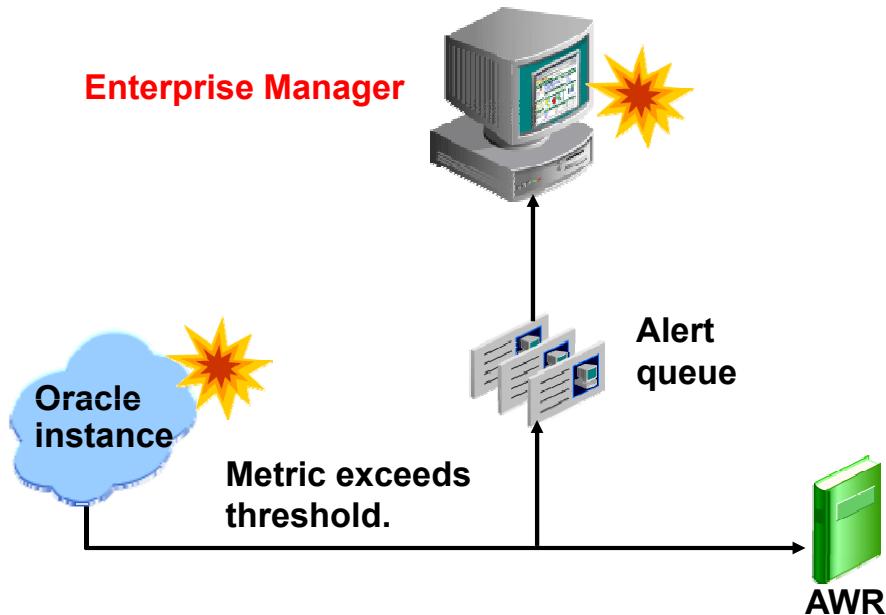
V\$FILE\_HISTOGRAM displays a histogram of all single block reads on a per file basis.

The histogram has buckets of time intervals from < 1 ms, < 2 ms, < 4 ms, < 8 ms, ... <  $2^{21}$  ms, <  $2^{22}$  ms,  $\geq 2^{22}$  ms.

You can also visualize the histogram statistics from Enterprise Manager. On the Performance page, click one of the legend area items for the Active Sessions graph, such as User I/O to drill down to the Active Sessions Waiting: User I/O page. Again, click the wait event name to the right of Active Sessions graph to reach the corresponding “Histogram for Wait Event” page. The “Histogram for Wait Event: db file sequential read” graph is shown in the slide.

**Note:** The histogram will not be filled unless the TIMED\_STATISTICS initialization parameter is set to TRUE, which is the default value. TIMED\_STATISTICS is set automatically when the STATISTICS\_LEVEL parameter is set to TYPICAL or ALL.

## Server-Generated Alerts



ORACLE®

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

Alerts are notifications of when a database is in an undesirable state and needs your attention. By default, the Oracle database sends alerts via Enterprise Manager, where they are displayed. Optionally, Enterprise Manager can be configured to send an email to the administrator about problem conditions. The Oracle Database server also keeps a history of the metric and the alerts in the workload repository.

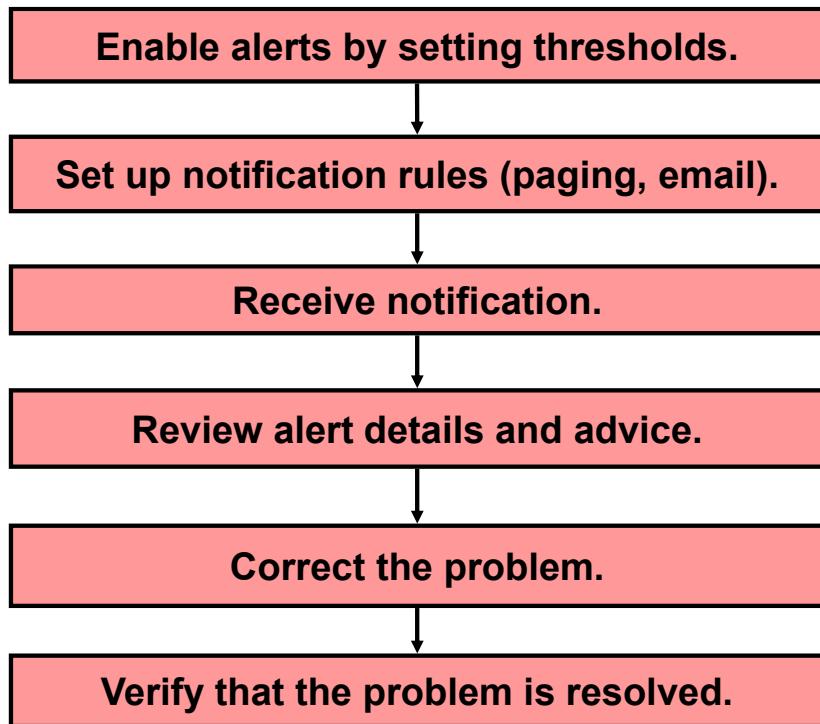
The alert queue is a persistent multiconsumer queue and is available for users who want to write customized alert handlers.

Thresholds on several key metrics, such as Tablespace Used (%), are set by default. You can set thresholds on the pertinent metrics for your system. If the database deviates from normal readings enough to cross those thresholds, then Oracle Database proactively notifies you by sending an alert. An early notification of potential problems enables you to respond quickly, and often resolve issues before users even notice them.

A few key metrics that can provide early problem notification are:

- Average File Read Time (centiseconds)
- Response Time (per transaction)
- SQL Response Time (%)
- Wait Time (%)

## Alert Usage Model



**ORACLE**

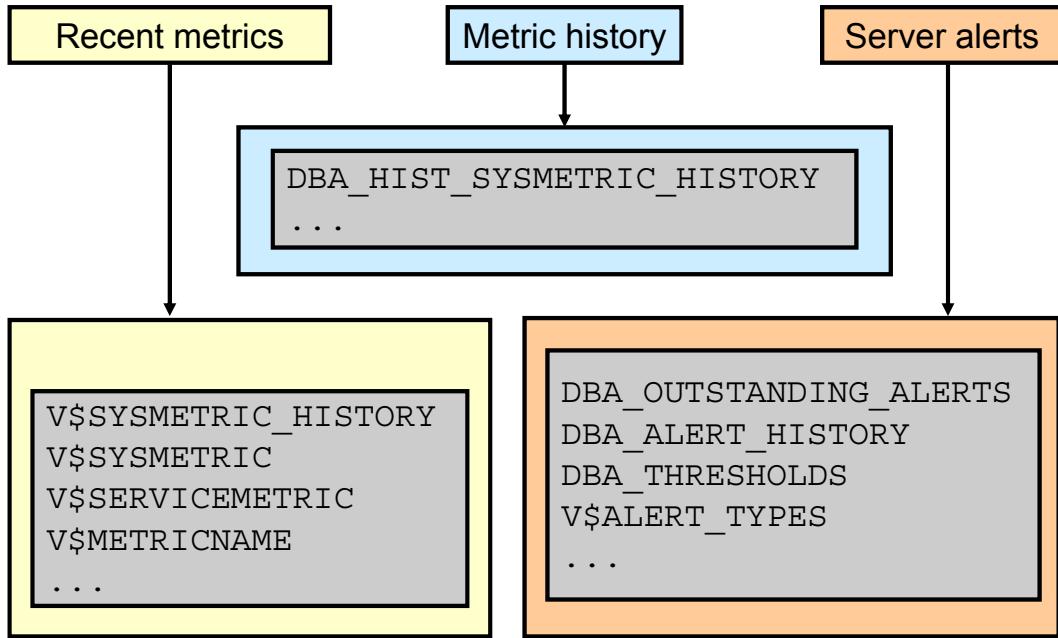
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The following is the basic usage model for server-generated alerts:

- If needed, you can change threshold settings for the server-alert metrics. You can do this by using Enterprise Manager Cloud Control or a PL/SQL procedure.
- You set up notification rules (for example, email address or blackout period) by using Enterprise Manager Cloud Control.
- When an alert is generated, Enterprise Manager displays the alert on the alert pages. Enterprise Manager Agent sends out a notification to administrators who have registered for it.
- When you get an alert, you can follow the advice that is given in the alert to correct the problem.

**Note:** You should ensure that the `STATISTICS_LEVEL` initialization parameter is set to `TYPICAL` or `ALL`.

## Metric and Alert Views



**ORACLE**

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

When enabled, the metric values are regularly computed by MMON and are kept in memory for one hour. The in-memory values for system-level metrics are viewable through the `V$SYSMETRIC` and `V$SYSMETRIC_HISTORY` views. Similar views are available for service-level metrics.

On-disk metric collection for all these metrics is enabled simply by enabling the automatic snapshot mechanism of AWR. The on-disk values for the metrics are viewable through the `DBA_HIST_*` views. The purge policy for metric history is the same as that for other snapshot data.

The following dictionary views enable you to access information about server alerts:

- `DBA_OUTSTANDING_ALERTS` describes alerts that the Oracle database server considers to be outstanding.
- `DBA_ALERT_HISTORY` represents a time-limited history of alerts that are no longer outstanding.
- `DBA_THRESHOLDS` gives you the threshold settings defined for the instance.
- `V$ALERT_TYPES` gives you information about each server alert type.

**Note:** For more information about these views, see the *Oracle Database Reference* guide.

# Quiz

An alert appeared on the Enterprise Manager database about two hours ago, and then disappeared. You do not remember the text. You can look in DBA\_ALERT\_HISTORY to find the cleared alert.

- a. True
- b. False



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

## Answer: a

The DBA\_ALERT\_HISTORY keeps cleared alerts. These alerts are purged on a regular basis. The purge policy for metric history is the same as that for other snapshot data.

## Summary

In this lesson, you should have learned how to:

- View metrics by using the metrics history views
- Create metric thresholds
- View alerts



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

## **Practice Overview 5: Working with Metrics**

This practice covers the following topics:

- Viewing metrics by using the metrics history views
- Creating metric thresholds
- Viewing alerts
- Clearing alerts



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



# 6

## Using Baselines

ORACLE®

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

# Objectives

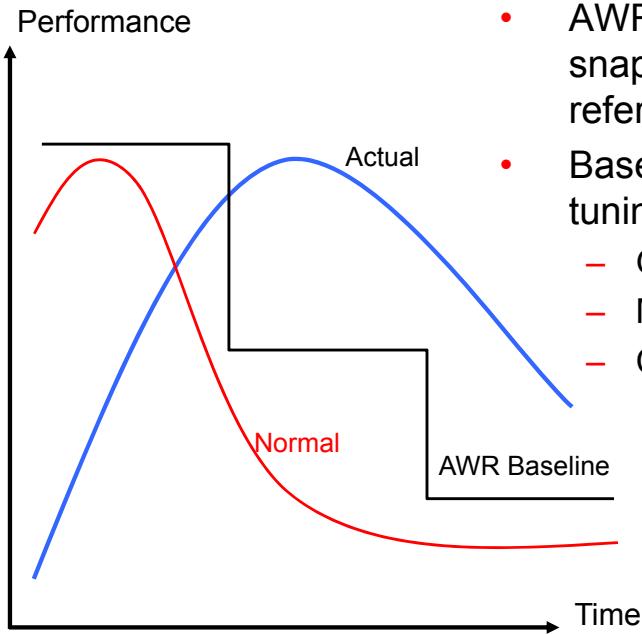
After completing this lesson, you should be able to do the following:

- Create AWR baselines
- Enable adaptive thresholds
- Create AWR baselines for future time periods



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

# Comparative Performance Analysis with AWR Baselines



- AWR baseline contains a set of AWR snapshots for an “interesting or reference” period of time.
- Baseline is key for performance tuning to:
  - Guide setting of alert thresholds
  - Monitor performance
  - Compare advisor reports

ORACLE®

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

What is the proper threshold to set on a performance metric? What is it that you want to detect? If you want to know that the performance metric value indicates that the server is nearing capacity, an absolute value is correct. But if you want to know that the performance is different today than it was at this time last week, or last month, the current performance must be compared to a baseline.

A *baseline* is sets of metrics and statistics. A single set is called a snapshot. A baseline is made up of two or more snapshots. Usually a baseline is captured during a period of normal or acceptable operation, but it can capture any period of interest. These snapshots are grouped statistically to yield a set of baseline values that vary over time. For example, the number of transactions per second in a certain database varies depending on the time of the day. The values for transactions per second are higher during working hours and lower during nonworking hours. The baseline records this variation and can be set to alert you if the current number of transactions per second is significantly different from the baseline values. When performance is not as expected, another set of metrics can be captured and compared with the baseline. This method allows the data to clearly point to the performance issues.

Oracle Database baselines provide the data required to calculate time-varying thresholds based on the baseline data. The baseline allows a real-time comparison of performance metrics with baseline data and can be used to produce AWR reports that compare two periods.

# Automatic Workload Repository Baselines

Automatic Workload Repository baselines enable you to:

- Specify adaptive thresholds
- Schedule the creation of a baseline by using baseline templates
- Rename baselines
- Set expiration dates for baselines

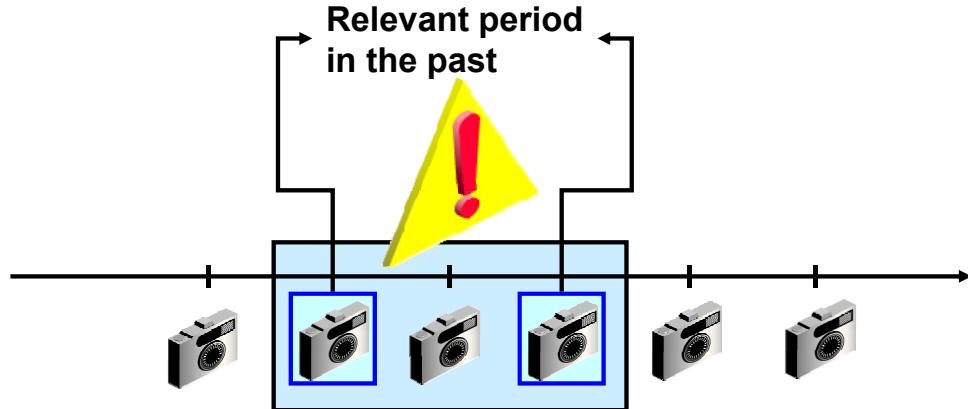


Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Automatic Workload Repository (AWR) baselines provide powerful capabilities for defining dynamic and future baselines, and considerably simplify the process of creating and managing performance data for comparison purposes.

Baselines are enabled by default if `STATISTICS_LEVEL=TYPICAL` or `ALL`.

## AWR Baselines



```
DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE (
    start_snap_id IN NUMBER,
    end_snap_id   IN NUMBER,
    baseline_name  IN VARCHAR2);
```

**ORACLE**

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

A baseline is a set of AWR snapshots. This is usually a set of snapshot data for important periods that you tag and retain in the AWR. A baseline is defined on a pair of snapshots; the snapshots are identified by their snapshot sequence numbers (`snap_ids`) or a start and end time. Each snapshot set has starting and ending snapshots and includes all the snapshots in between. Snapshot sets are used to retain snapshot data. Therefore, by default, snapshots belonging to snapshot sets are retained until the snapshot sets are dropped. An expiration value can be set to a number of days that the snapshot will be retained.

A baseline is identified by a user-supplied name. Execute the `CREATE_BASELINE` procedure to create a baseline from a set of snapshots, and specify a name and a pair of snapshot identifiers. A baseline identifier that is unique for the life of a database is assigned to the newly created baseline. Usually you set up baselines from representative periods in the past, to be used for comparisons with current system behavior. You can also set up threshold-based alerts by using baselines in Enterprise Manager Cloud Control. You can set the expiration time in a number of days with the `expiration` parameter of this procedure. The default is `NULL`, meaning “never expire.”

You can get the `snap_ids` directly from `DBA_HIST_SNAPSHOT`, or from Enterprise Manager Cloud Control

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

## Types of Baselines

Two types of baselines:

- Static (fixed): Corresponds to a fixed, contiguous time period in the past
- Moving window: Corresponds to all AWR data that exists within the AWR retention period



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

Oracle Database provides the ability to collect two kinds of baselines: *static* baselines and *moving window*. Static baselines can be either single or repeating. A single AWR baseline is collected over a single time period. A repeating baseline is collected over a repeating time period (for example, every Monday in June).

A moving window baseline corresponds to all AWR data that exists within the AWR retention period.

Oracle Database has, by default, a system-defined moving window baseline that corresponds to all of the AWR data within the AWR retention period. There can only be one moving window baseline.

# Moving Window Baseline

- There is one moving window baseline:  
`SYSTEM_MOVING_WINDOW`
  - A moving window baseline that corresponds to the last eight days of AWR data
  - Created automatically
- By default, the adaptive thresholds functionality computes statistics on this baseline.

Automatic Workload Repository > AWR Baselines > Baseline  
Edit Baseline: **SYSTEM\_MOVING\_WINDOW**

**General**

Name **SYSTEM\_MOVING\_WINDOW**  
ID 0  
Type **MOVING\_WINDOW**  
Adaptive Thresholds Enabled? **No**  
Window Size (Days) **8**

**Validity**

Interrupted by Shutdown? **YES**  
% of Total Time **82**  
Error Count **0**

**Time Interval**

Start Time **5/9/13 9:00 AM**  
End Time **5/17/13 8:01 AM**  
Start Snap ID **348**  
End Snap ID **509**



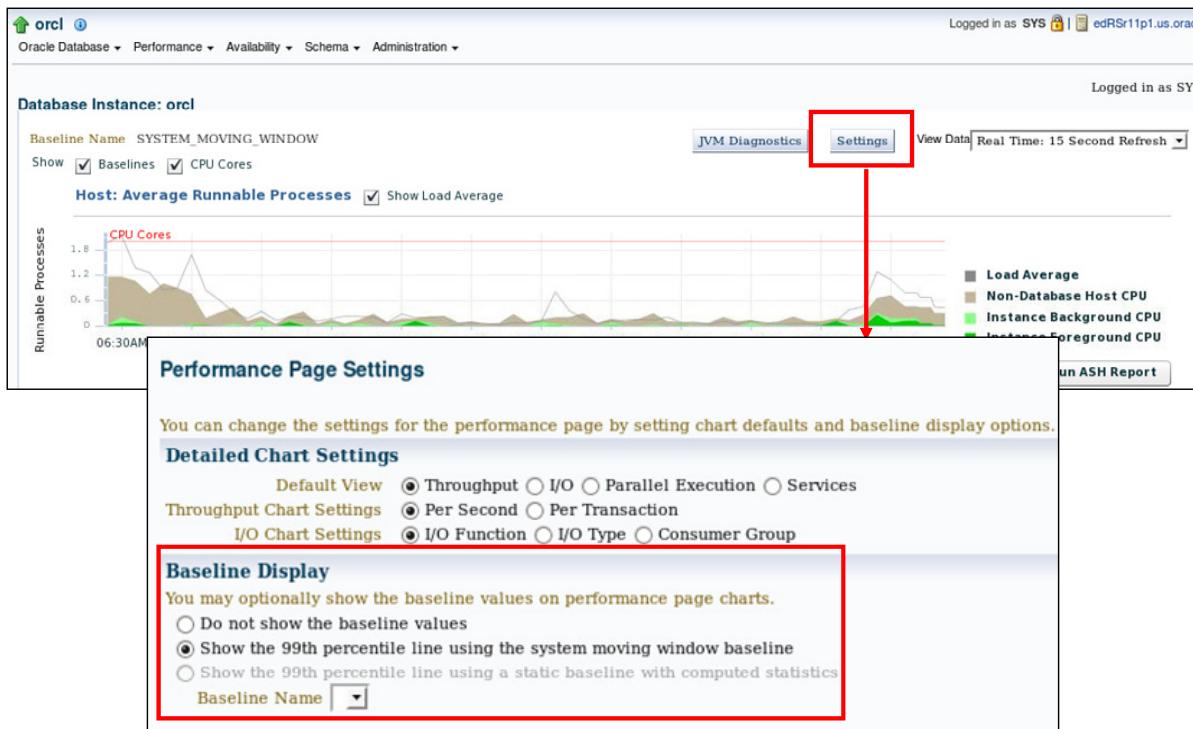
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Oracle Database automatically maintains a system-defined moving window baseline. The default window size for the system-defined moving window baseline is the current AWR retention period, which by default is eight days. For any production database, consider using a larger moving window (such as 30 days) to accurately compute threshold values. You can resize the moving window baseline by changing the number of days in the moving window to a value that is equal to or less than the number of days in the AWR retention period. Therefore, to increase the size of a moving window, you first need to increase the AWR retention period accordingly. The AWR retention period and window size for the system-defined moving window baseline are two separate parameters. The AWR retention period must be greater than or equal to the window size for the system-defined moving window baseline.

This system-defined baseline provides a default baseline for Enterprise Manager performance pages to show a comparison of the performance with the current database performance.

To view the AWR Baseline page, expand Performance, expand AWR and select AWR Administration. Click the Baselines link in the Manage Snapshots and Baselines section of the Automatic Workload Repository page. On the AWR Baselines page, select the `SYSTEM_MOVING_WINDOW` baseline and click Edit to change the window size.

# Baselines in Performance Page Settings



**ORACLE**

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

The data for any defined baseline in the past is available in Oracle Database. The baseline data may be displayed on the Performance page of Enterprise Manager. You have three display options:

- Do not show baseline information.
- Show the information from a specified static baseline.
- Show the information from the system moving baseline.

**Note:** The system moving window baseline becomes valid after sufficient data has been collected and the statistics calculation occurs. By default, the statistics calculation is scheduled for every Saturday at midnight.

## Baseline Templates

- Templates enable you to schedule the creation of baselines for time periods of interest in the future:
  - Single time period in the future
  - Repeating schedule
- Examples
  - A known holiday weekend
  - Every Monday morning from 10 AM to 2 PM
- When the end time for a baseline template changes from future to past, MMON detects the change and creates the baseline.



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

Creating baselines for future time periods enables you to mark time periods that you know will be interesting. For example, you may want the system to automatically generate a baseline for every Monday morning for the whole year, or you can ask the system to generate a baseline for an upcoming holiday weekend if you suspect that it is a high-volume weekend.

A nightly MMON task goes through all the templates for baseline generation and checks to see if any time ranges have changed from the future to the past within the last day. For the relevant time periods, the MMON task then creates a baseline for the time period.

# Creating AWR Baselines

The screenshot shows the Oracle Database Automatic Workload Repository Baselines page. At the top, it says "Logged in as SYS". Below that, it shows the path "Automatic Workload Repository > Baselines" and the sub-path "AWR Baselines". It displays a table of baselines, with one row selected: "SYSTEM\_MOVING\_WINDOW" (Type: MOVING\_WINDOW (8 Days), Valid: No, Statistics Computed: Yes). The "Create" button in the top right of the table area is highlighted with a red box. A red arrow points from this button down to the "Create Baseline: Baseline Interval Type" page.

Select	Name	Type	Valid	Statistics Computed	Last Time Computed	Start Time	End Time	Error Count
<input checked="" type="radio"/>	SYSTEM_MOVING_WINDOW	MOVING_WINDOW (8 Days)	No	Yes	May 12, 2013 8:00:04 AM	May 9, 2013 9:00:50 AM	May 17, 2013 8:01:00 AM	0

**Related Links**

- AWR Baseline Templates
- Baseline Metric Thresholds

ORACLE®

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

You can create two types of AWR baselines: single and repeating. The Create Baseline: Baseline Interval Type page gives the following explanations:

- The single type of baseline has a single and fixed time interval.
- The repeating type of baseline has a time interval that repeats over a time period: for example, every Monday from 10:00 AM to 12:00 PM.

To view the AWR Baseline page, expand Performance, expand AWR and select AWR Administration. Click the Baselines link in the Manage Snapshots and Baselines section of the Automatic Workload Repository page. On the AWR Baselines page, click Create and follow the wizard instructions to create your baseline.

**Note:** Before you can set up AWR baseline metric thresholds for a particular baseline, you must compute the baseline statistics. Select Schedule Statistics Computation from the actions menu to compute the baseline statistics. There are several other actions available.

# Creating a Single AWR Baseline

Automatic Workload Repository > AWR Baselines > Baseline  
Create Baseline: Single Baseline

Logged in as SYS

The single type of baseline has a single and fixed time interval. For example, from Jan 1, 2007 10:00 AM to Jan 1, 2007 12:00 PM.

\* Baseline Name

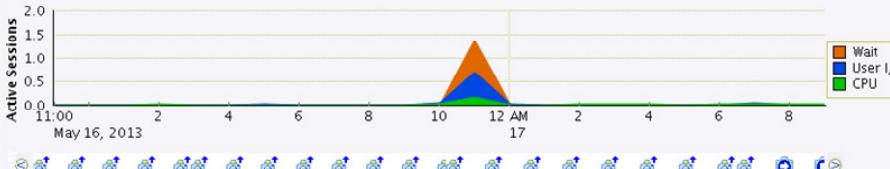
**Baseline Interval**

Snapshot Range

**Select Time Period**  
Choose the Period Start Time option, then click a snapshot icon in the chart to select the period start time. Repeat the process for the period end time.

Period Start Time

Period End Time



Time Range

Start Time  (example: May 17, 2013)

End Time  (example: May 17, 2013)

TIP If both the Start Time and the End Time are in the future, a baseline template with the same name as the baseline will be created.



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

If you selected the Single option on the Create Baseline: Baseline Interval Type page, you access the page shown in this slide.

Select the time period corresponding to your interest in one of two ways:

- Select the Snapshot Range option, and then set the period start time and period end time by following the directions on the page. If the icon that you want to select is not shown, you can change the chart time period.
- Specify the time range, with a date and time for start and end times. With the Time Range option, you can choose times in the future.

When you are finished, click Finish to create the static baseline.

**Note:** If the end time of the baseline is in the future, a baseline template with the same name as the baseline will be created.

# Creating a Repeating Baseline and Template

Automatic Workload Repository > AWR Baselines > Baseline  
**Create Baseline: Repeating Baseline Template**

The repeating type of baseline has a time interval that repeats over a time period. For example, every Monday from 10:00 AM

\* Baseline Name Prefix **MON\_REPEAT\_BASELINE**

**Baseline Time Period**

Start Time **12** AM Duration (Hours) **1**

**Frequency**

Daily  
 Weekly  
 Monday  Tuesday  Wednesday  Thursday  Friday  Saturday  Sunday

**Interval of Baseline Creation**

Start Time **May 18, 2013** **9** **45** **AM** End Time **May 18, 2013** **10** **00** **AM**  
(example: May 17, 2013) (example: May 17, 2013)

**Purge Policy**

Retention Time (Days) **30**

**TIP** A baseline template with the same name as the baseline name prefix will be created.

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

You can define repeating baselines by using Enterprise Manager. After selecting Repeating on the Create Baseline: Baseline Interval Type page, you can specify the repeat interval as shown in this slide. You specify the start time and the duration of the baseline. Then specify when the baseline statistics will be collected (daily or weekly; if weekly, for which days). Specify the range of dates for which this baseline template will collect statistics. Retention Time sets an expiration value for the baseline; a value of NULL indicates that the baseline never expires.

# Managing Baselines by Using the DBMS\_WORKLOAD\_REPOSITORY Package

Procedure Name	Description
CREATE_BASELINE	Creates a single AWR baseline
DROP_BASELINE	Drops a single AWR baseline
RENAME_BASELINE	Renames a baseline
CREATE_BASELINE_TEMPLATE	Creates a baseline template
DROP_BASELINE_TEMPLATE	Drops a baseline template
MODIFY_BASELINE_WINDOW_SIZE	Modifies the window size for the Default Moving Window Baseline

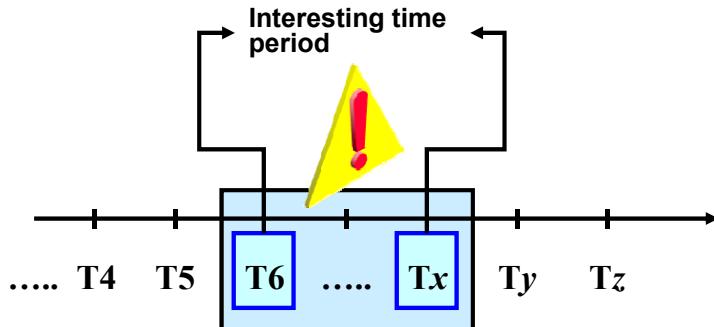


Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The DBMS\_WORKLOAD\_REPOSITORY PL/SQL package contains procedures that enable you to manage the workload repository. For example, you can find procedures for managing snapshots and baselines in this package. The procedures shown are only a few of the procedures provided. Most of the procedures are used by Enterprise Manager to manage the Automatic Workload Repository, and you seldom need to use the procedures directly.

**Note:** For more information about these procedures and other procedures to manage the AWR contained in the DBMS\_WORKLOAD\_REPOSITORY package, refer to the *Oracle Database PL/SQL Packages and Types Reference* guide.

# Generating a Baseline Template for a Single Time Period



```
BEGIN
  DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE_TEMPLATE (
    start_time  => to_date('21-JUN-2013','DD-MON-YYYY'),
    end_time    => to_date('21-SEP-2013','DD-MON-YYYY'),
    baseline_name  => 'FALL13',
    template_name  => 'FALL13',
    expiration      => NULL );
END;
```

ORACLE

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

You can create a template for how baselines are to be created for different time periods in the future, for predictable schedules. If any part of the period is in the future, use the CREATE\_BASELINE\_TEMPLATE procedure.

For the baseline template, when the end time becomes a time in the past, a task using these inputs automatically creates a baseline for the specified time period when the time comes. The example creates a baseline template that creates a baseline when 0:0:0 21-SEP-2013 is in the past.

Using time-based definitions in baseline creation does not require the start-snapshot and end-snapshot identifiers. For the CREATE\_BASELINE\_TEMPLATE procedure, you can specify an expiration duration for the baseline that is created from the template. The expiration duration, specified in days, represents the number of days that you want the baselines to be maintained.

A value of NULL means that the baselines never expire.

To create a baseline over a period in the past, use the CREATE\_BASELINE procedure.

# Creating a Repeating Baseline Template

```
BEGIN  
DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE_TEMPLATE (   
    day_of_week          => 'SATURDAY',  
    hour_in_day          => 6,  
    duration             => 20,  
    start_time           => to_date('21-JUN-2012','DD-MON-YYYY'),  
    end_time              => to_date('21-JUN-2013','DD-MON-YYYY'),  
    baseline_name_prefix => 'SAT_MAINT_WIN'  
    template_name         => 'SAT_MAINT_WIN',  
    expiration            => 90,  
    dbid                 => NULL );  
END;
```



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

Use the CREATE\_BASELINE\_TEMPLATE procedure to generate baseline templates that automatically create baselines for a contiguous time period based on a repeating time schedule. You can also specify whether you want the baseline to be automatically removed after a specified expiration interval (expiration).

The example in the slide generates a template that creates a baseline for a period that corresponds to each SATURDAY\_MAINTENANCE\_WINDOW for a year. The baseline is created over a 20-hour period (duration) that starts at 6:00 AM (hour\_in\_day) each Saturday (day\_of\_week). The baseline is named 'SAT\_MAINT\_WIN' with time information appended to make the name unique. The template is named 'SAT\_MAINT\_WIN', and each baseline will be kept for 90 days (expiration). This template is created for the local database (dbid => NULL ).

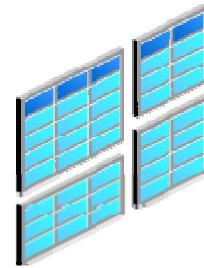
Use this baseline to compare the resources that are used each Saturday during the maintenance window.

The DBMS\_WORKLOAD\_REPOSITORY package has several procedures to manage baselines. See *Oracle Database PL/SQL Packages and Types Reference* for more information.

# Baseline Views

Data dictionary support for baselines:

- DBA\_HIST\_BASELINE
- DBA\_HIST\_BASELINE\_DETAILS
- DBA\_HIST\_BASELINE\_TEMPLATE
- DBA\_HIST\_BASELINE\_METADATA



**ORACLE**

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

The following data dictionary views provide information about AWR baselines:

- DBA\_HIST\_BASELINE displays information about baselines taken in the system. For each baseline, this view displays the complete time range and whether the baseline is the default baseline. Additional information includes the date created, time of last statistics calculation, and type of baseline.
- DBA\_HIST\_BASELINE\_DETAILS displays information that allows you to determine the validity of a given baseline, such as whether there was a shutdown during the baseline period and the percentage of the baseline period that is covered by the snapshot data.
- DBA\_HIST\_BASELINE\_TEMPLATE holds the baseline templates. This view provides the information needed by MMON to determine when a baseline will be created from a template and when the baseline should be removed.
- DBA\_HIST\_BASELINE\_METADATA displays metadata information for the baselines, including name, type, creation time, template, and expiration.

For detailed information about each view, refer to *Oracle Database Reference*.

# Performance Monitoring and Baselines

- Performance alert thresholds are difficult to determine, because expected metric values vary by:
  - Workload type
  - System load
- Baselines metric value statistics are:
  - Automatically computed over the system moving window
  - Manually computed over static baselines



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

When they are properly set, alert thresholds provide a valuable service—an alert—by indicating a performance metric that is at an unexpected value. Unfortunately, in many cases, the expected value varies with the workload type, system load, time of day, or day of the week. Baselines associated with certain workload types or days of the week capture the metric values of that period. The baseline can then be used to set the threshold values when similar conditions exist.

The statistics for baselines are computed to place a minimal load on the system; statistics for static baselines are manually computed. You can schedule statistics computation on the AWR Baselines page. Statistics for the system moving window are automatically computed according to the `BSLN_MAINTAIN_STATS_SCHED` schedule. By default, this schedule starts the job every week at noon on Saturday.

# Performance Monitoring and Baselines

Baseline metric statistics determine alert thresholds.

- Unusual values versus baseline data = significance level thresholds.
- Close or exceeding peak value over baseline data = percentage of maximum thresholds.



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

Metric statistics computed over a baseline enable you to set thresholds that compare the baseline statistics to the current activity. There are three methods of comparison: significance level, percentage of maximum, and fixed values.

Thresholds based on *significance level* use statistical relevance to determine which current values are unusual. In simple terms, if the significance level is set to .99 for a critical threshold, the threshold is set where 1% of the baseline values fall outside this value and any current values that exceed this value trigger an alert. A higher significance level of .999 or .9999 causes fewer alerts to be triggered.

Thresholds based on *percentage of maximum* are calculated based on the maximum value captured by the baseline.

Threshold values based on *fixed values* are set by the DBA. No baseline is required.

# Defining Alert Thresholds Using a Static Baseline

The screenshot illustrates the Oracle Database 12c AWR Baseline Metric Thresholds feature. It shows the 'Baseline Metric Thresholds' page where various metrics like Average Active Sessions, Average Synchronous Single-Block Read Latency (ms), Response Time (centi-seconds per call), Response Time (per transaction), Cumulative Logons (per second), Executes (per second), I/O Requests (per second), Network Bytes (per second), Number of Transactions (per second), Physical Reads (per second), Physical Writes (per second), Redo Generated (per second), and User Calls (per second) are listed. The 'Edit Thresholds' page is open for the 'Number of Transactions (per second)' metric, showing threshold settings for 'Significance Level' (Very High (0.99)), 'Critical' (0), 'Warning' (High (0.95)), and 'Occurrences' (2). A chart displays the transaction count over time, with a legend for Metric Data, Critical Threshold, Warning Threshold, and Median of Baseline.

**ORACLE**

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

After AWR baseline statistics are computed for a particular baseline, you can set metric thresholds specific to your baseline.

Compute baseline statistics directly from the Baselines page (as previously discussed). Then go to the AWR Baseline Metric Thresholds page and select the type of metrics that you want to set. When done, select a specific metric and click Edit Thresholds.

On the corresponding Edit AWR Baseline Metric Thresholds page, specify your thresholds in the Thresholds Settings section, and then click Apply Thresholds.

You can specify thresholds based on the statistics computed for your baseline. This is illustrated in the slide. In addition to "Significance Level," the other possibilities are "Percentage of Maximum" and "Fixed Values."

**Note:** After a threshold is set using Baseline Metric Thresholds, the previous threshold values are forgotten forever and the statistics from the associated baseline are used to determine the threshold values until they are cleared (by using the Baseline Metric Threshold UI or PL/SQL interface).

# Configuring a Basic Set of Thresholds

Database Instance: orcl > Baseline Metric Thresholds

**Baseline Metric Thresholds**

Threshold Configuration Metric Analysis

View Basic Metrics

Category/Name	Alerts (Last 7 Days)	AWR Baseline	Threshold Type	Adaptive	Edit Thresholds
Performance Metrics	0 0				
Average Active Sessions	0 0				
Average Synchronous Single-Block Read Latency (ms)	0 0				
Response Time (centi-seconds per call)	0 0				
Response Time (per transaction)	0 0				
Workload Volume Metrics	0 0				

**Quick Configuration: Baseline Metric Thresholds**

Select one of the following workload profiles to configure a basic set of thresholds that you can expand or change later, if desired.

Workload Profile

- Primarily OLTP (pure transaction processing 24 hours a day)
- Primarily Data Warehousing (query and load intensive)
- Alternating (OLTP during the daytime and batch during the nighttime)

! By choosing a profile and proceeding with quick configuration, some thresholds that have already been set will be cleared. Refer to the Configuration effects for each system profile.

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

Enterprise Manager Cloud Control enables you to select adaptive thresholds for database performance metrics, with full integration with AWR baselines as the source for the metric statistics. Enterprise Manager offers a quick configuration option in a one-click starter set of thresholds based on OLTP or Data Warehouse workload profiles.

Select the appropriate workload profiles from the pop-up window. By making this simple selection, the system automatically configures and evolves adaptive thresholds based on the SYSTEM\_MOVING\_WINDOW baseline for the group of metrics that best correspond to the chosen workload.

## Quiz

You want a baseline that always covers the last three months because there is a peak in processing every Friday. You want to set a threshold to alert you if the load exceeds 110% for the maximum of the previous 90 days. Which of the following steps would you perform to accomplish this?

- a. Create a static baseline over the last 90 days.
- b. Change the system baseline window size to 90 days.
- c. Schedule a job to calculate the statistics on the system moving window every 90 days.
- d. Create a repeating baseline that creates a new baseline every 90 days.



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

### Answer: b

Only the system moving baseline will be adjusted as time progresses. By setting the window to 90 days, you will see the last 3 month end peaks and be able to set thresholds based on the load of the previous 3 months.

A is false, because a static baseline will cover only the specified 90-day period and does not change. The static baseline does not move.

C is false, because the calculation of statistics on the system baseline is scheduled by the system for every Saturday and cannot be changed.

D is false, because a repeating baseline creates a set of static baselines over fixed periods.

## Summary

In this lesson, you should have learned how to:

- Create metric baselines
- Enable adaptive thresholds
- Create AWR baselines for future time periods



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

## **Practice 6: Overview Using AWR Baselines**

This practice covers creating AWR baselines:

- Creating a baseline over a past time interval
- Applying the baseline to the performance page graphs
- Creating a repeating period baseline over periods in the future



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





## Using AWR-Based Tools

**ORACLE**

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

# Objectives

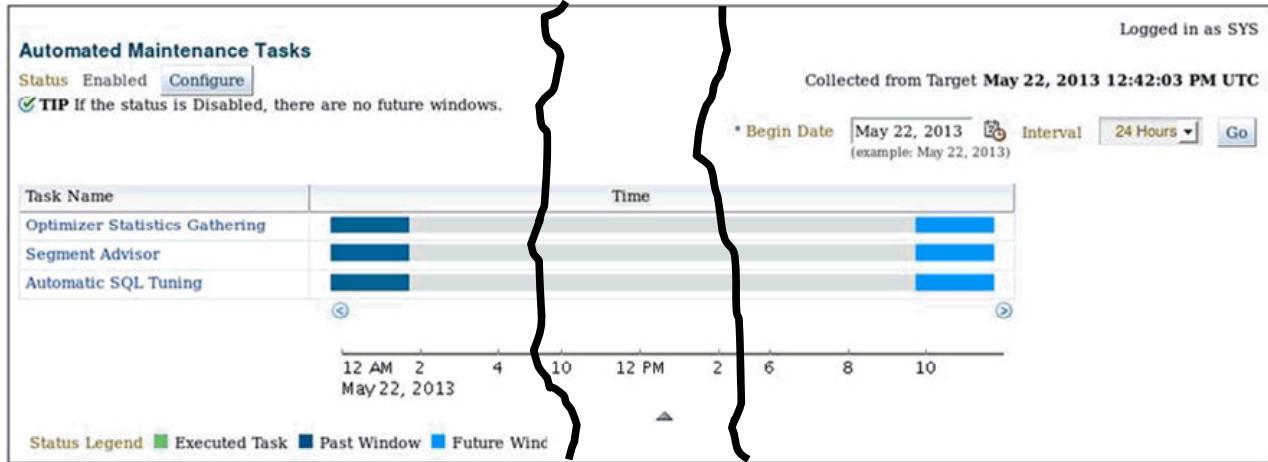
After completing this lesson, you should be able to do the following:

- Tune automatic maintenance tasks
- Generate ADDM reports
- Generate Compare Period ADDM reports
- Generate Active Session History (ASH) reports
- Perform Emergency Monitoring
- Use Real-Time ADDM



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

# Automated Maintenance Tasks



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

Three automated maintenance tasks are built into Oracle Database: Optimizer Statistics Gathering, Segment Advisor, and Automatic SQL Tuning. The SPM Evolve Advisor Task is a subtask of Automatic SQL Tuning.

These tasks run in a set of predefined job windows. The tasks are assigned to resource consumer groups. The resource plan that is in effect during the window controls the resources that the tasks are allowed to consume, such as CPU.

These automatic tasks help you monitor space usage, collect optimizer statistics, and tune high-load SQL statements.

You should monitor these tasks. The results can appear in the form of recommendations referenced from Automatic Database Diagnostic Monitor (ADDM) reports or be referenced from the Advisor Central page. The execution of these tasks is prioritized. If the tasks do not have sufficient time or resources to finish, they are rescheduled for the next window. When the tasks are filling the available windows, the duration and resource allocation may need to be changed.

# Maintenance Windows

## Maintenance Window Group

10 PM – 2 AM Mon to Fri

6 AM – 2 AM Sat to Sun

**Scheduler Windows**

Following are the system windows that specify resource usage limits based on time-duration windows.

Logged in as SYS

[Create](#)

View	Edit	Delete	Create Like	Go					
Select	Name	Resource Plan	Enabled	Next Open Date	End Date	Duration (min)	Active	Description	
<input checked="" type="radio"/>	WEEKNIGHT_WINDOW			Feb 1, 2013 10:00:00 PM		480	FALSE	Weeknight window - for compatibility only	
<input type="radio"/>	WEEKEND_WINDOW			Feb 2, 2013 12:00:00 AM		2880	FALSE	Weekend window - for compatibility only	
<input type="radio"/>	WEDNESDAY_WINDOW	DEFAULT_MAINTENANCE_PLAN	<input checked="" type="checkbox"/>	May 22, 2013 10:00:00 PM		240	FALSE	Wednesday window for maintenance tasks	
<input type="radio"/>	THURSDAY_WINDOW	DEFAULT_MAINTENANCE_PLAN	<input checked="" type="checkbox"/>	May 23, 2013 10:00:00 PM		240	FALSE	Thursday window for maintenance tasks	
<input type="radio"/>	FRIDAY_WINDOW	DEFAULT_MAINTENANCE_PLAN	<input checked="" type="checkbox"/>	May 24, 2013 10:00:00 PM		240	FALSE	Friday window for maintenance tasks	
<input type="radio"/>	SATURDAY_WINDOW	DEFAULT_MAINTENANCE_PLAN	<input checked="" type="checkbox"/>	May 25, 2013 6:00:00 AM		1200	FALSE	Saturday window for maintenance tasks	
<input type="radio"/>	SUNDAY_WINDOW	DEFAULT_MAINTENANCE_PLAN	<input checked="" type="checkbox"/>	May 26, 2013 6:00:00 AM		1200	FALSE	Sunday window for maintenance tasks	
<input type="radio"/>	MONDAY_WINDOW	DEFAULT_MAINTENANCE_PLAN	<input checked="" type="checkbox"/>	May 27, 2013 10:00:00 PM		240	FALSE	Monday window for maintenance tasks	
<input type="radio"/>	TUESDAY_WINDOW	DEFAULT_MAINTENANCE_PLAN	<input checked="" type="checkbox"/>	May 28, 2013 10:00:00 PM		240	FALSE	Tuesday window for maintenance tasks	



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

The Automated Maintenance Tasks feature relies on the Resource Manager being enabled during the maintenance windows. The resource plan associated with the window is automatically enabled when the window opens. The goal is to prevent maintenance work from consuming excessive amounts of system resources. Each maintenance window is associated with a resource plan that specifies how the resources will be allocated during the window duration.

Automated tasks are assigned to specific windows. All daily windows belong to MAINTENANCE\_WINDOW\_GROUP by default.

You can define other maintenance windows, as well as change start times and durations for the daily maintenance windows. Likewise, any maintenance windows that are deemed unnecessary can be disabled or removed. The operations can be done by using Enterprise Manager or Scheduler interfaces.

# Default Maintenance Plan

```
SQL> SELECT name FROM v$rsrc_plan
  2 WHERE is_top_plan = 'TRUE';

NAME
-----
DEFAULT_MAINTENANCE_PLAN
```

Resource Plans > View Resource Plan: DEFAULT\_MAINTENANCE\_PLAN  
[View Resource Plan: DEFAULT\\_MAINTENANCE\\_PLAN](#)

Logged in as SYS

Actions [Create Like](#) [Go](#) [Edit](#) [Return](#)

[General](#) [Parallelism](#) [Thresholds](#) [Idle Time](#)

A Resource Plan contains directives that specify how resources are allocated to Consumer Groups. For each Consumer Group, a directive specifies the amount of CPU resources are allocated. It also specifies limits, such as the maximum degree of parallelism, execution time, and amount of I/O, that each session in the Consumer Group can consume. You can enable a Resource Plan manually or automatically, using Scheduler Windows.

Plan	DEFAULT_MAINTENANCE_PLAN
Description	Default plan for maintenance windows that prioritizes SYS_GROUP operations, leaving 5% for automated maintenance operations.
<input type="checkbox"/> Activate this plan <input type="checkbox"/> Automatic Plan Switching Enabled	

**Resource Allocations**

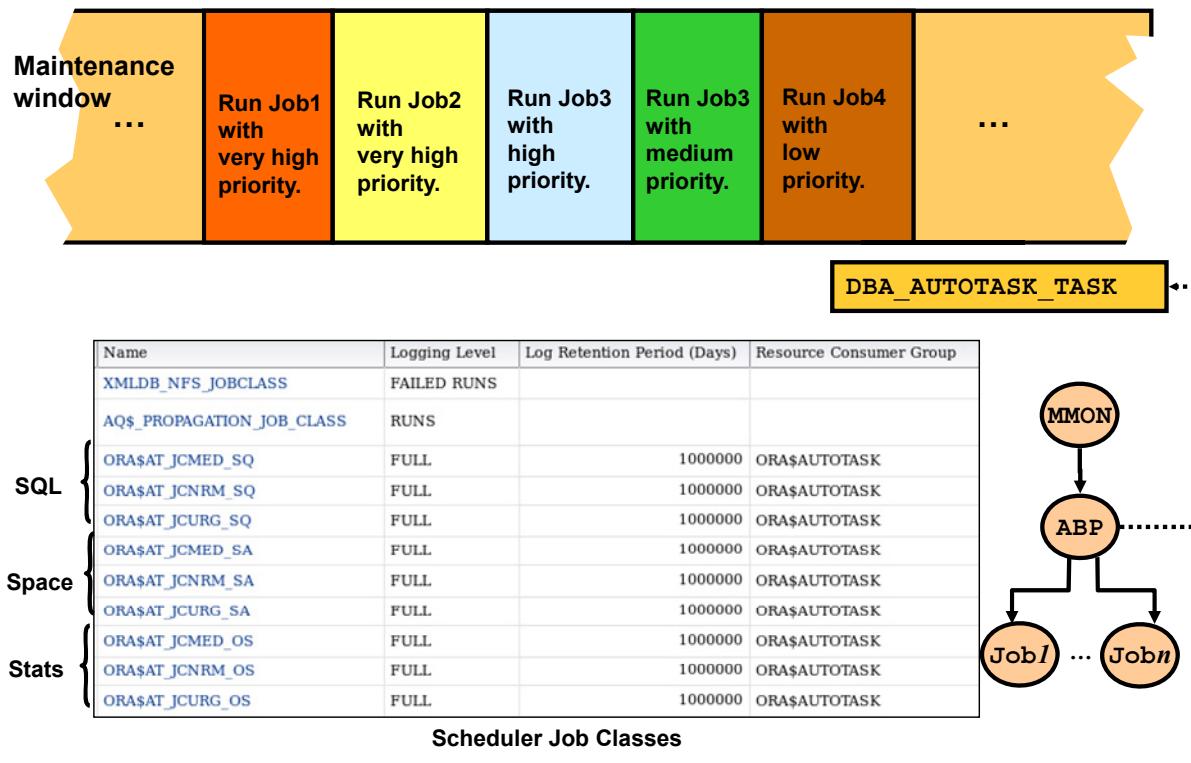
Group/Subplan	Shares	Percentage	Utilization Limit %	<a href="#">Add/Remove</a>
ORA\$AUTOTASK	5	5	90	
SYS_GROUP	75	75	100	
OTHER_GROUPS	20	20	100	
Total Shares:	100			



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

When a maintenance window opens, the DEFAULT\_MAINTENANCE\_PLAN in the Resource Manager is automatically set to control the amount of CPU resources used by the automated maintenance tasks. To be able to give different priorities to each possible task during a maintenance window, various consumer groups are assigned to DEFAULT\_MAINTENANCE\_PLAN.

# Automated Maintenance Task Priorities



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

ORACLE

The Automated Maintenance Tasks feature is implemented by Autotask Background Process (ABP). ABP functions as an intermediary between automated tasks and the Scheduler. Its main purpose is to translate automated tasks into AUTOTASK jobs for execution by the Scheduler. Just as important, ABP maintains a history of execution of all tasks. ABP stores its private repository in the SYSAUX tablespace; you view the repository through DBA\_AUTOTASK\_TASK.

ABP is started by MMON at the start of a maintenance window. There is only one ABP required for all instances. The MMON process monitors ABP and restarts it if necessary.

ABP determines the list of jobs that need to be created for each maintenance task. This list is ordered by priority: very high, high, medium, low, very low. Within each priority group, jobs are arranged in the preferred order of execution. ABP creates jobs in the following manner: all very high priority jobs are created first, all high jobs are created next, and all very low priority jobs are created last.

ABP assigns jobs to various Scheduler job classes. These job classes map the job to a consumer group based on priority.

**Note:** There is no job that is permanently associated with a specific automated task. Therefore, it is not possible to use DBMS\_SCHEDULER procedures to control the behavior of automated tasks; use the DBMS\_AUTO\_TASK\_ADMIN procedures instead.

# Configuring Automated Maintenance Tasks

Window	Optimizer Statistics Gathering	Segment Advisor	Automatic SQL Tuning
WEDNESDAY_WINDOW	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
THURSDAY_WINDOW	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
FRIDAY_WINDOW	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
SATURDAY_WINDOW	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
SUNDAY_WINDOW	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
MONDAY_WINDOW	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
TUESDAY_WINDOW	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>



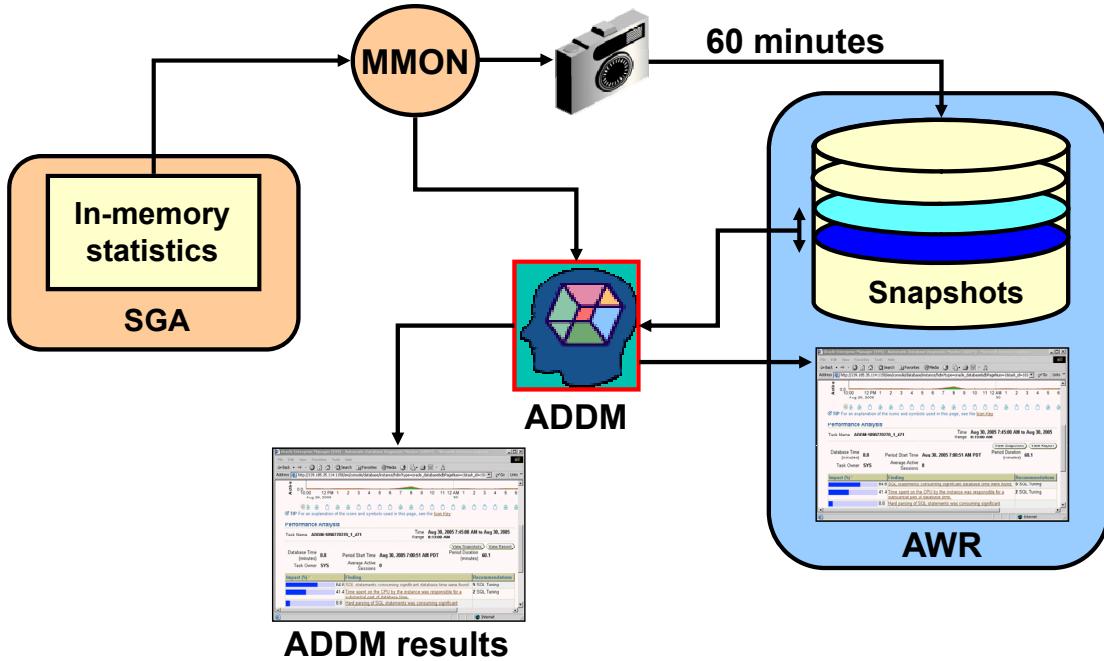
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Automated Maintenance Tasks feature determines when—and in what order—tasks are performed. As a DBA, you can perform the following configuration tasks:

- Adjust the duration and start time of the maintenance window if the maintenance window turns out to be inadequate for the maintenance workload.
- Control the resource plan that allocates resources to the automated maintenance tasks during each window.
- Enable or disable individual tasks in some or all maintenance windows.
- In a RAC environment, shift maintenance work to one or more instances by mapping maintenance work to a service. Enabling the service on a subset of instances shifts maintenance work to these instances.

Enterprise Manager is the preferred way to control Automated Maintenance Tasks. However, you can also use the `DBMS_AUTO_TASK_ADMIN` package.

# ADDM Performance Monitoring



**ORACLE**

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

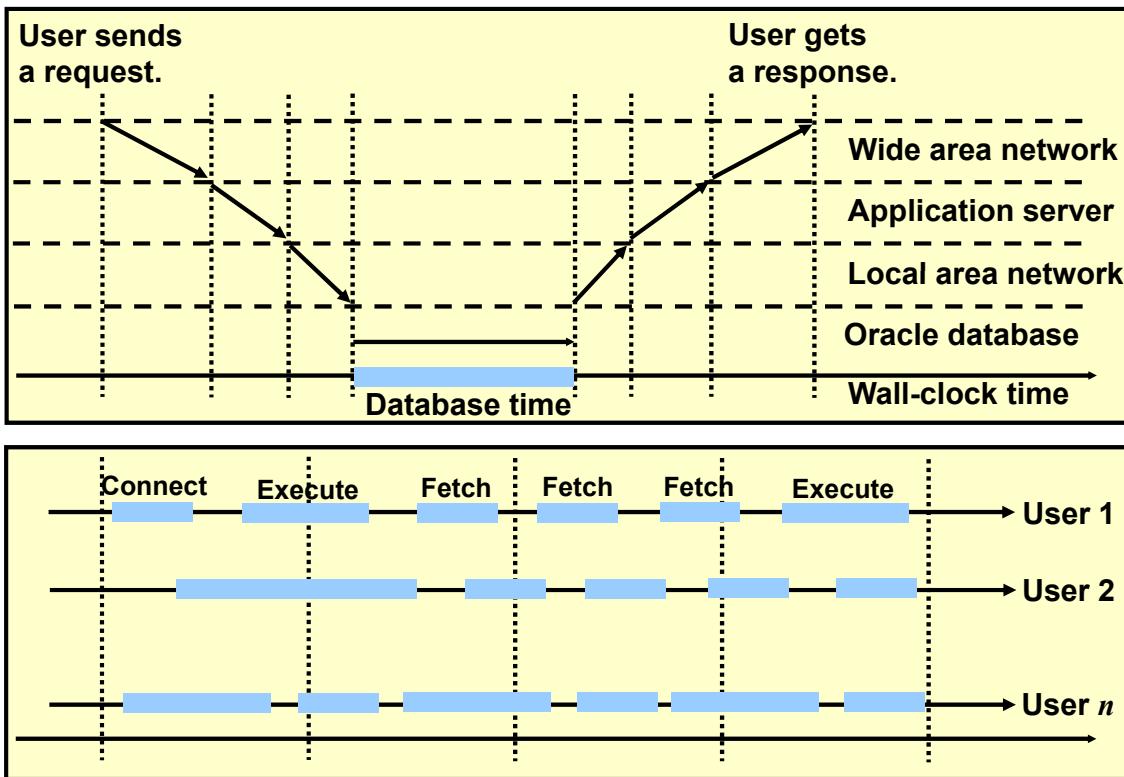
By default, the Oracle Database server automatically captures statistical information from the SGA every 60 minutes and stores it in the Automatic Workload Repository (AWR) in the form of snapshots. These snapshots are stored on disk and are similar to Statspack snapshots. However, they contain more precise information than the Statspack snapshots.

Additionally, ADDM is scheduled to run automatically by the MMON process on every database instance to detect problems proactively. Each time a snapshot is taken, ADDM is triggered to perform an analysis of the period corresponding to the last two snapshots. This approach proactively monitors the instance and detects bottlenecks before they become a significant problem.

The results of each ADDM analysis are stored in Automatic Workload Repository and are also accessible through Enterprise Manager.

**Note:** Although ADDM analyzes Oracle database performance over the period defined by the last two snapshots, it is possible to manually invoke an ADDM analysis across any two snapshots.

## ADDM and Database Time



**ORACLE**

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

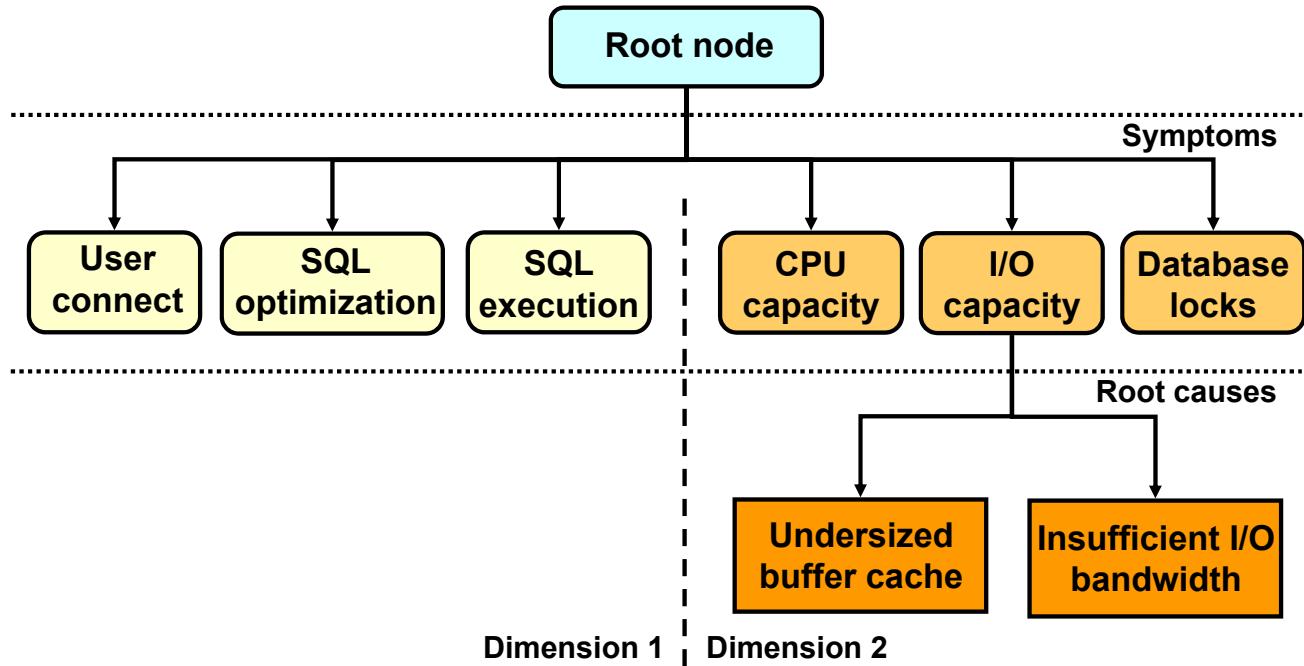
*Database time* is defined as the sum of the time spent inside the database processing user requests. In the slide, the graphic at the top illustrates a simple scenario of a single user submitting a request. The user's response time is the time interval between the instant the request is sent and the instant the response is received. The database time involved in that user request is only a portion of that user's response time that is spent inside the database.

The graphic at the bottom illustrates the database time as it is summed over multiple users, and each user is performing a series of operations resulting in a series of requests to the database. You can see that the database time is directly proportional to the number and duration of user requests, and can be higher or lower than the corresponding wall-clock time (elapsed time).

Using the database time as a measure, you can gauge the performance impact of any entity of the database. For example, the performance impact of an undersized buffer cache would be measured as the total database time spent in performing additional I/O requests that could have been avoided if the buffer cache were larger.

Database time is simply a measurement of the total amount of work done by the database server. The rate at which the database time is consumed is the database load average, measured as database time per second. The objective of ADDM is to reduce the amount of database time spent on a given workload, which is analogous to consuming less energy to perform the same task.

# DB Time-Graph and ADDM Methodology



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

Identifying the component contributing the most database time is equivalent to finding the single database component that provides the greatest benefit when tuned. ADDM uses database time to identify database components that require investigation and also to quantify performance bottlenecks. The first step in automatic performance tuning is to correctly identify the causes of performance problems. Only when the root cause of the performance problem is correctly identified, is it possible to explore effective tuning recommendations to solve or alleviate the issue.

ADDM looks at the database time spent in two independent dimensions:

- The first dimension looks at the database time spent in various phases of processing user requests. This dimension includes categories such as “connecting to the database,” “optimizing SQL statements,” and “executing SQL statements.”
- The second dimension looks at the database time spent using or waiting for various database resources used in processing user requests. The database resources considered in this dimension include both hardware resources, such as CPU and I/O devices, and software resources, such as database locks and application locks.

To perform automatic diagnosis, ADDM looks at the database time spent in each category under both these dimensions and drills down to the categories that had consumed significant database time. This two-dimensional drill-down process can be represented using the DB Time-graph.

Performance problems often distribute the database time across many categories in one dimension but not in the other. For example, a database with insufficient CPU capacity slows down all phases involved in processing user requests, which is in the first dimension of the ADDM analysis. However, it would be evident from the second dimension that the top performance problem affecting the database is insufficient CPU capacity. This two-dimensional view of determining where the database time is consumed gives ADDM very good judgment in zooming in to the more significant performance issues.

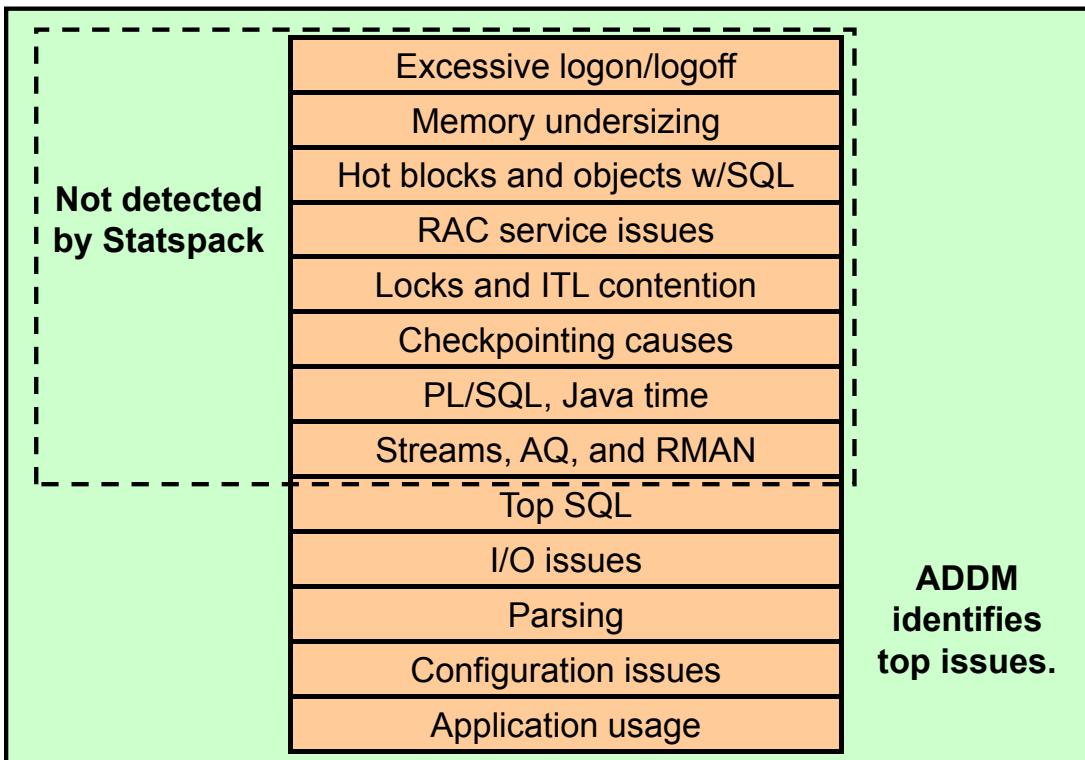
ADDM explores this DBTime-graph starting at the root node and all the children of any node where the database time consumed is significant. Branch nodes in this graph identify the performance impact of what is usually a symptom of some performance bottleneck. The terminal nodes identify particular root causes that can explain all the symptoms that were significant along the path in which the terminal node was reached. For example, in the slide, the I/O Capacity branch node measures the database time spent in all I/O requests, which is significant due to various bottlenecks. Whenever significant database time is spent in I/O requests, all the children of the I/O Capacity node are explored. They correspond to the two terminal nodes in the slide. The Undersized Buffer Cache node points to a particular root cause: The data-block buffer cache is undersized causing excessive number of I/O requests. The Insufficient I/O Bandwidth node looks for hardware issues that can slow down all I/O requests. After a terminal node identifies a root cause, ADDM measures the impact of the root cause in database time. It then explores ways that can solve or alleviate the problem identified. ADDM uses the various metrics and statistical measurements collected by AWR to come up with tuning recommendations. The system maintains measurements at various granularities, so an ADDM analysis can go from the symptoms (for example, commit operations consumed significant database time) to the root cause (for example, write I/O to one of the log files was very slow—possibly a hardware issue). The nodes also allow ADDM to estimate the maximum possible database time that can be saved by the suggested tuning recommendations, which is not necessarily equal to the database time attributed to the root cause.

In addition to identifying bottlenecks, ADDM also identifies key components that are not experiencing any performance bottlenecks. The idea is to prevent you from tuning components that have marginal effect on the total database throughput.

It is interesting to note that ADDM need not traverse the entire DBTime-graph. It can prune the uninteresting subgraphs. This can be achieved only because the DBTime-graph is constructed in a way that a node's database time is contained in the database time attributed to its parents. By pruning and not traversing uninteresting subgraphs, which represent database components that are not consuming significant database time, the cost of an ADDM analysis depends only on the number of actual performance problems that were affecting the database. The cost of the analysis does not depend on the actual load on the database or on the number of issues ADDM could potentially diagnose.

At the end of the analysis, ADDM reports the top root causes identified, ranked by the performance impact attributed with each root cause along with the respective tuning recommendations.

## Top Performance Issues Detected



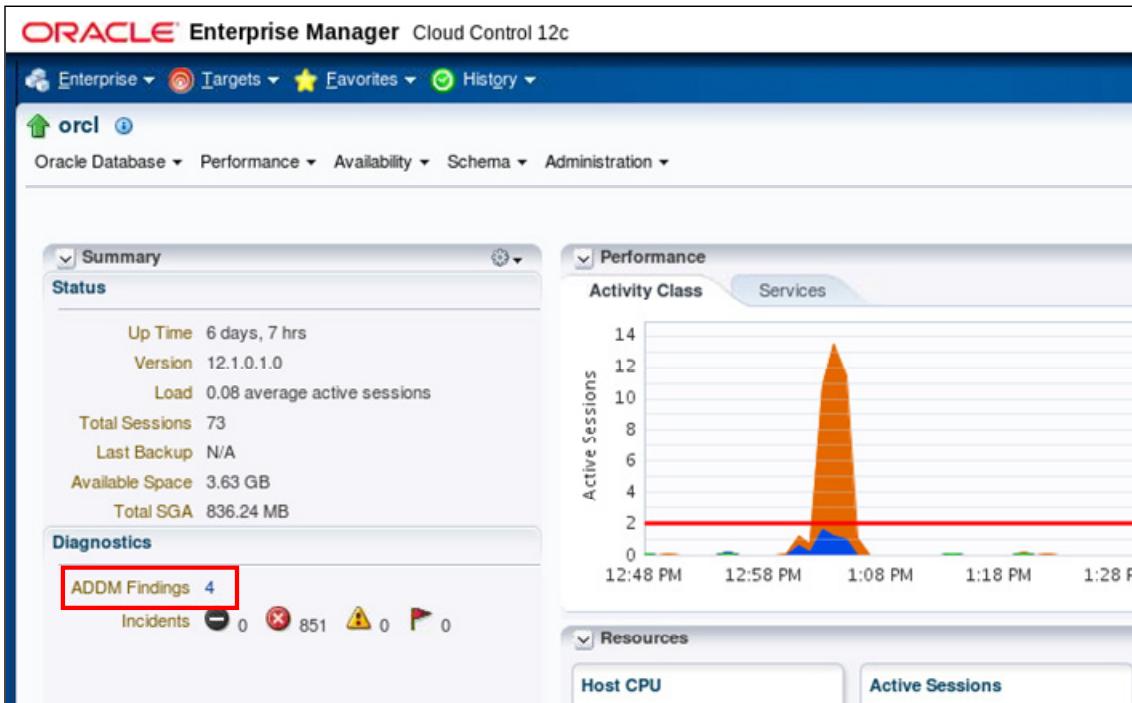
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Statspack is unable to identify some of the problems listed in the slide because of lack of granularity in the statistics. With the wait-and-time statistics model, ADDM is able to identify the top performance issues listed. Another benefit of ADDM over Statspack is that ADDM concentrates its analysis on top problems (the ones that impact the system most).

In addition, ADDM can make analyses in areas related to CPU, paging, and integrated cache. The scope of ADDM includes server components such as Streams, Advanced Queuing (AQ), and RMAN.

**Note:** The list of performance issues in the slide is not a comprehensive list of issues that are automatically detected by ADDM. In fact, it represents only a subset of the potential issues that ADDM can discover.

# Observing ADDM Findings



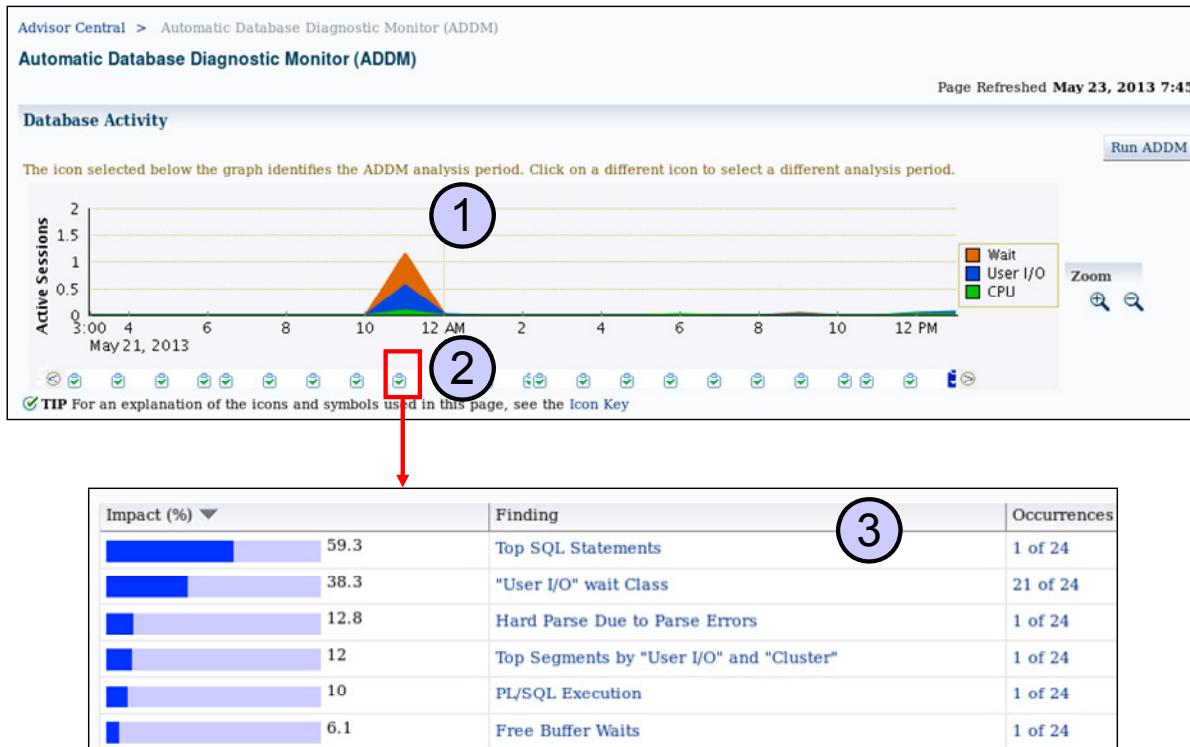
ORACLE

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

The Diagnostics section on the Database Home page shows you the number of ADDM findings from the previous automatic run.

By clicking the ADDM Findings link, you are directed to the Automatic Database Diagnostic Monitor (ADDM) page, where you can access the details of the latest ADDM run.

# ADDM Analysis Results



**ORACLE**

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

On the Automatic Database Diagnostic Monitor (ADDM) page, you can see the detailed findings for the latest ADDM run. Database time represents the sum of the non-idle time spent by sessions in the database for the analysis period. A specific impact percentage is given for each finding. The impact represents the time consumed by the corresponding issue compared with the database time for the analysis period.

The following information relates to the numbers in the slide:

1. The graphic shows the number of average active users. Also, the major problem was a wait problem.
2. The icon shows that the ADDM output displayed at the bottom of the page corresponds to this point in time. You can see the history (to view previous analysis) by clicking the other icons.
3. The findings give you a short summary of what ADDM found as performance areas in the instance that could be tuned. By clicking a particular issue, you are directed to the Performance Finding Details page.

You can click the View Report button to get details of the performance analysis in the form of a text report.

# ADDM Recommendations

The screenshot shows the ADDM Performance Finding Details page for a finding titled "Buffer Busy - Hot Objects". The finding details include:

- Finding:** Read and write contention on database blocks was consuming significant database time.
- Impact (Active Sessions):** 5.11
- Percentage of Finding's Impact (%):** 79.4
- Period Start Time:** Aug 1, 2013 7:58:14 AM
- End Time:** Aug 1, 2013 8:02:57 AM
- Filtered:** No Filters

**Recommendations:**

Details	Category	Benefit (%)
Hide Schema	Schema	79.4
Action	Consider using ORACLE's recommended solution of automatic segment space management in a locally managed tablespace for the tablespace "TBSSPC" containing the TABLE "SPC.SPCT" with object ID 95240. Alternatively, you can move this object to a different tablespace that is locally managed with automatic segment space management. Database Object SPC.SPCT	79.4
Rationale	There was significant read and write contention on TABLE "SPC.SPCT" with object ID 95240. Database Object SPC.SPCT	79.4
Show Schema	Schema	79.4
Show Schema	Schema	79.4

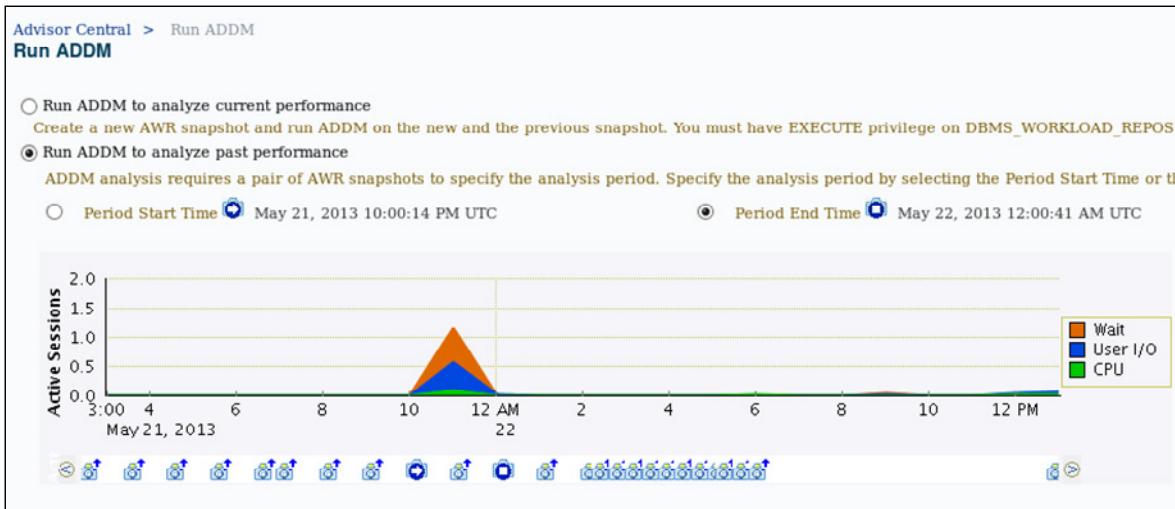
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

On the Performance Finding Details page, you are given some recommendations to solve the corresponding issue. Recommendations are grouped into Schema, SQL Tuning, DB Configuration, and other categories. The Benefit (%) column gives you the maximum reduction in database elapsed time if the recommendation is implemented.

ADDM considers a variety of changes to a system, and its recommendations can include:

- Hardware changes:** Adding CPUs or changing the I/O subsystem configuration
- Database configuration:** Changing initialization parameter settings
- Schema changes:** Hash-partitioning a table or index, or using Automatic Segment-Space Management (ASSM)
- Application changes:** Using the cache option for sequences or using bind variables
- Using other advisors:** Running the SQL Tuning Advisor on high-load SQL or running the Segment Advisor on hot objects

# Creating a Manual ADDM Task



**ORACLE**

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

By default, ADDM tasks are run for every Oracle database snapshot that is stored in the workload repository. However, you can create a custom ADDM task to analyze a period of time that you identify with a starting snapshot and an ending snapshot.

To create an ADDM task by using Enterprise Manager Cloud Control, expand the Performance menu, select Advisors Home, and click the ADDM link on the Advisor Central page.

Choose the Period Start Time option, and then click the snapshot you want to use as the start point of the period of time. Then choose the End Time option and click the snapshot to use as the terminating point of the time period.

Click the OK button. This displays the Automatic Database Diagnostic Monitor (ADDM) page, on which you get the confirmation that a new task has been created.

In the Performance Analysis section of this page, you get the result of your manually created task.

**Note:** The results of a manually created ADDM task are also accessible from the Advisor Central page. You can search for your task by using the Search section of this page.

# Changing ADDM Attributes

1. Ensure that STATISTICS\_LEVEL is set to TYPICAL or ALL.
2. ADDM analysis of I/O performance depends on the expected speed of the I/O subsystem:
  - a. Measure your I/O subsystem speed.
  - b. Set the expected speed.

```
SQL> exec DBMS_ADVISOR.SET_DEFAULT_TASK_PARAMETER(
      'ADDM', 'DBIO_EXPECTED', 8000);
```

```
SELECT parameter_value, is_default
  FROM dba_advisor_def_parameters
 WHERE advisor_name = 'ADDM' AND
       parameter_name = 'DBIO_EXPECTED';
```



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

ADDM is enabled by default and is controlled by the STATISTICS\_LEVEL initialization parameter. ADDM does not run automatically, if STATISTICS\_LEVEL is set to BASIC. The default setting for STATISTICS\_LEVEL is TYPICAL.

ADDM analysis of I/O performance partially depends on the DBIO\_EXPECTED ADDM parameter. This parameter describes the expected performance of the I/O subsystem. The value of DBIO\_EXPECTED represents the average time to read a single database block in microseconds. ADDM uses the default value of 10,000 microseconds (10 milliseconds), which is an appropriate value for most modern hard drives. If your hardware is significantly different, consider using a different value. To determine the correct setting for DBIO\_EXPECTED, perform the following steps:

1. Measure the average read time of a single database block read for your hardware. Note that this measurement is for random I/O, which includes seek time if you use standard hard drives. Typical values for hard drives are between 5,000 and 20,000 microseconds.
2. Set the DBIO\_EXPECTED value. For example, if the measured value is 8,000 microseconds, you should execute the first command shown in the slide connected as the SYS user. The query shown in the slide gives you the current value of this parameter.

## Retrieving ADDM Reports by Using SQL

```

SELECT dbms_advisor.GET_TASK_REPORT(task_name)
FROM   dba_advisor_tasks
WHERE  task_id = (
    SELECT max(t.task_id)
    FROM   dba_advisor_tasks t,
           dba_advisor_log l
    WHERE  t.task_id = l.task_id      AND
           t.advisor_name = 'ADDM'      AND
           l.status = 'COMPLETED');
  
```

```

SQL> @?/rdbms/admin/addmrpt
Instance      DB Name      Snap Id      Snap Started      Level
-----        -----
orcl          ORCL         434 06 Feb 2010 00:00          1
...
Enter value for begin_snap: 434
Enter value for end_snap: 436
...
Enter value for report_name:
Generating the ADDM report for this analysis ...
  
```



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

The first example in the slide shows you how to display the most recent ADDM report by using a SQL command.

To diagnose database performance issues, ADDM analysis can be performed across any two AWR snapshots if the following requirements are met:

- Both snapshots did not encounter any errors during creation, and both have not yet been purged.
- There were no shutdown and startup actions between the two snapshots.

The second example uses the `addmrpt.sql` script. This SQL\*Plus script can be used to run ADDM on any two AWR snapshots that are provided. The two snapshots must have been taken by the same instance.

The script identifies your `DBID` and lists the snapshot identifiers for the last three days. This can help you determine the pair of snapshots on which you want to perform the analysis.

# Quiz

The DB time%, also labeled Impact% or Benefit% in the ADDM report, is a percentage of:

- a. Elapsed time related to response time
- b. Total CPU time used by the item reported
- c. Total time in database calls by all processes
- d. Total elapsed time used by database calls



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

## Answer: c

DB Time is the sum of all the time spent in database calls by all the sessions. It has a theoretical maximum of elapsed time times the number of sessions. Many sessions can be in database calls concurrently, most waiting on various operations.

## AWR Compare Periods Report: Review

Only AWR statistics reported in both cases:

- Comparison between two time periods
- Comparison between DB Replay capture and replay or two replays



Missing intelligent reporting with analysis:

- Changes that happened
- Mapping root causes to performance degradation effects



ORACLE®

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

The AWR Compare Periods report is interesting but you still have to perform an analysis on these metrics to find the effects that were mapped to the root causes of the performance degradation or improvement.

Diligence and experience is needed to find the changes to the environment, workload, and results from the AWR Compare Periods report.

## Compare Periods ADDM: Analysis

Causes Identify system changes	Effects Identify performance difference	Map effects to causes with a rule set
<ul style="list-style-type: none"> <li>• Configuration changes           <ul style="list-style-type: none"> <li>– DB version</li> </ul> </li> <li>• Workload changes           <ul style="list-style-type: none"> <li>– Change in SQLs</li> <li>– Database time consumed by these SQLs</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• Computes problem impacts with ADDM methodology           <ul style="list-style-type: none"> <li>– In base period</li> <li>– In compare period</li> </ul> </li> <li>• Measures the difference</li> </ul>	<ul style="list-style-type: none"> <li>• Rules triggered by performance changes           <ul style="list-style-type: none"> <li>– SGA_TARGET decrease can cause I/O increase</li> </ul> </li> </ul>



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

The Compare Period ADDM report, as opposed to AWR Compare Periods report, performs a cause-to-effect analysis.

1. It first identifies the system changes that may have caused the performance change. For example, it detects a configuration change in DB version, or a workload change with SQL changes. These causes can cause performance difference.
2. Then it identifies the effects of these particular changes. For that purpose, it runs an ADDM analysis for the base period and one for the compare period and then measures the differences between both periods.
3. Finally, it maps the effects to the causes with rule sets. For example, an SGA\_TARGET increase can cause an I/O increase.

Compare Period ADDM provides more than what you had before with former methods, the identification of changes and an intelligent cause to effect analysis.

Is the change due to changes in the workload? Did some new application start running? Were there some changes at the OS level? Was an instance parameter changed?

The report displays changes in the resource demand at the hardware and software levels.

These and other questions are more easily answered using this advisor than with the previous AWR Compare Periods reports.

## Workload Compatibility

Comparison “current” Period

to

Base “earlier” Period

- The test time period
- The “current” production period
- The 1<sup>st</sup> RAT replay
- The 2<sup>nd</sup> RAT replay

- Baseline
- DB Replay capture period
- 1<sup>st</sup> RAT replay before modifications

### Workload Compatibility

- Is the same application running in both periods?
- Gauge the workloads’ similarity, taking into account SQL statements and load.
- Is ideally 100% compatible between DB Replay capture and replay

**ORACLE**

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

To generate a Compare Period ADDM report, you have to perform the following steps:

1. Define the Base Period as being the normal one
2. Define the Reference Period against which you compare the problematic period. The problematic period can cover situations such as a period in a production database, a period when tests are performed in a test database, a period when using new optimizer enhancements, or another type of storage, or adding more memory, more CPU, or new nodes in a RAC environment.

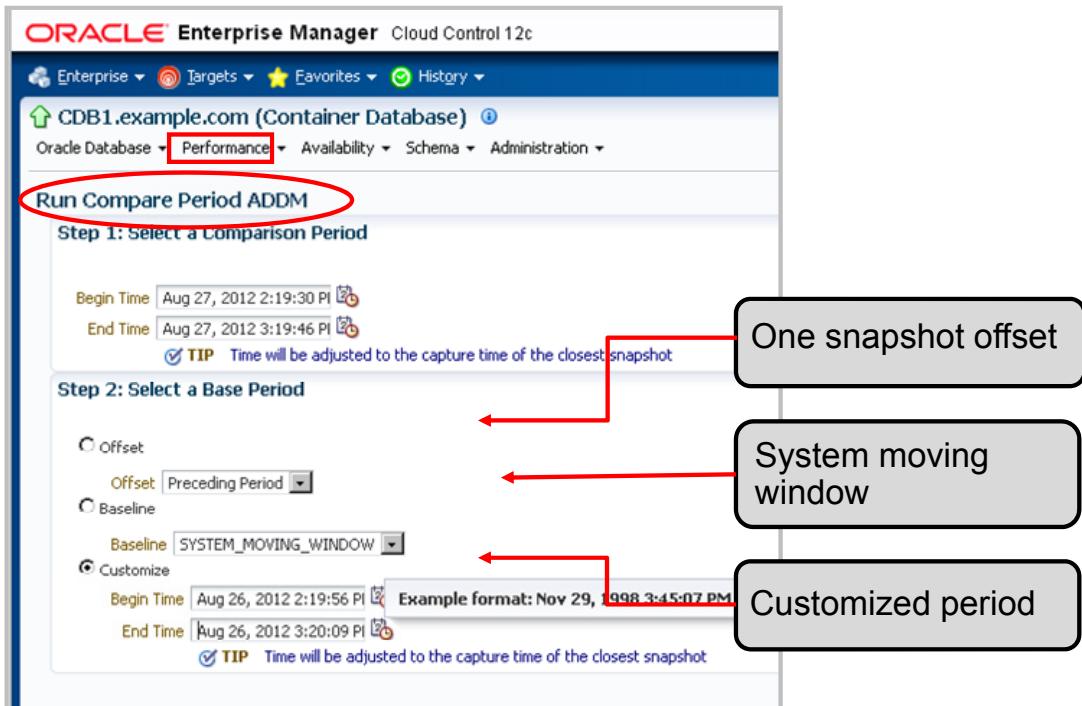
In some cases, you use DB Replay, capturing and replaying where the Base Period would be the capture period, and the Compare Period the replay one.

A new factor comes into play during the comparison, the workload compatibility between two periods. Do you compare similar things? Are the SQL statements similar in both periods? Is the same application running in both periods? This is called “SQL Commonality”.

This is a good way to gauge the workloads’ similarity, taking into account SQL statements and their respective load. In a live system, it is not 100% certain that DBAs compare exact or even similar application periods. If the result of the report shows an 80 or 90% compatibility level, then the DBA can rely on the findings provided by Compare Period ADDM.

Testing with Real Application Testing, you ideally get a 100% compatibility between a DB Replay capture and a replay or between two replays because the workload is packaged by DB Replay.

# Comparison Modes



**ORACLE**

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

1. First, select the Comparison Period that you want to inspect where you noticed a performance degradation.
2. Compare it to the Base Period where the performance was acceptable and that reflects a similar workload.

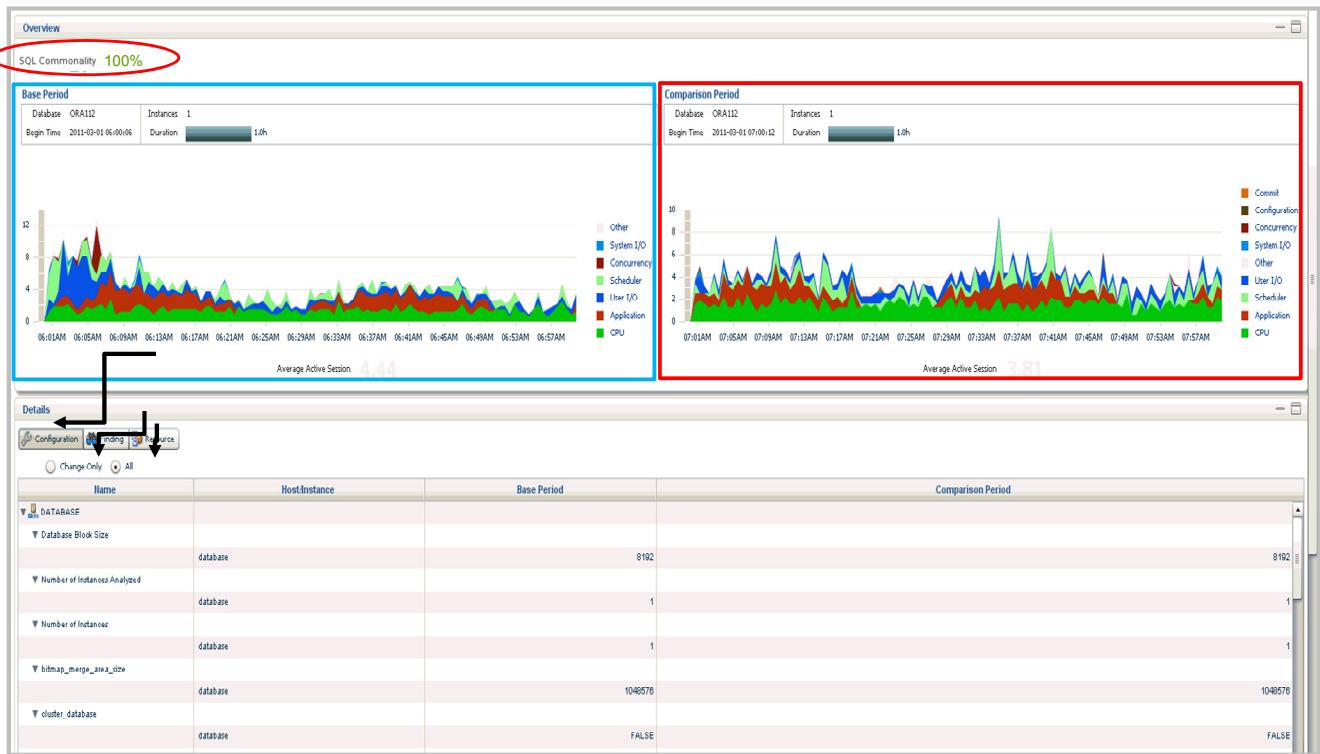
So you have three options to define the Base Period.

- The first option allows you to select an offset of one snapshot: in this example, the comparison period starts at 2 and ends at 3; therefore, the base period will start at 1 and end at 2. In this same option, you can select an offset of one day, so it will use the previous day, same time as the comparison period. You can even choose a week back to compare with the performance of the previous week at the same time.
- The second option allows you to select a baseline, either the out-of-box `system_moving_window` baseline or any other baseline you created to reflect your normal application.
- The third option allows you to choose any other period of time.

In any case, it will automatically adjust the base period to a range of AWR snapshots that are as close as possible to the selected period of time.

3. And then run the report. It should come back with the details of the differences between both periods.

# Report: Configuration



**ORACLE**

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

The report shows various types of information. It shows a list of statistics as with AWR Compare Periods report, and it displays graphics that are easier and faster to interpret.

The overview at the top of screen shows the two periods.

Between the two periods being compared, the report shows that they are 100% identical in terms of SQL compatibility. If the report shows a compatibility under 80%, you should not rely on this comparison. So if you want a good, reliable comparison, check that the SQL commonality is at least 80 or 90%. Even if the compatibility is less than 80%, it is still worthwhile to take a look at the findings. It will provide a list of new SQLs that were run in the system and how they affected the performance.

Now with this intelligent reporting, you get the changes that occurred between the two periods. In the details below the overview, you can choose to view only the configuration changes that occurred between the two periods, and not necessarily all the configuration information that is identical between the two periods.

If there was an instance parameter change, it will appear with the base period value and the comparison period value.

This information informs you about the existing changes but it does not explain why the performance degraded (or improved) and if this change had an impact on the performance degradation (or improvement). It is possible that this change did not impact the performance at all. It can be the contrary. It helped in performance. So another change might be the culprit.

# Report: Finding



ORACLE

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

You need to get the findings that explain what caused the performance degradation or improvement.

Here again you can choose to view only the areas where it degraded or improved, or all of them, and also set the percentage of impact for which you need to know the findings.

In the example in the slide, one of the reasons why the performance degraded is that the buffer cache size was undersized from 1.52 MB to 2 MB which was not sufficient. For each of the performance differences analyzed, you can get a detailed description of each finding and the SQL statements that were impacted positively or negatively.

# Report: Resource CPU and I/O



ORACLE

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

In the example in the slide, the CPU page on the Resource tab shows a warning. There was more CPU consumed by Oracle and Oracle Run Queue.

On the Resource tab, the I/O page shows the relative I/Os in both periods and tells you that neither period was I/O bound.

# Report: Resource Memory

The screenshot shows a software interface titled 'Resource' with a red circle highlighting the 'Memory' tab. Below the tabs, there are two sections: 'Virtual Memory Paging' (Base) and 'Virtual Memory Paging' (Comparison). A table displays memory usage details for a host named 'host01.example.co'. The table has four columns: Host Name, Total Physical Mem..., Total Memory Paging Out, and Memory Used by Oracle Database. The 'Total Memory Paging Out' column shows values of 56.6MByte and 50.3MByte for the base and comparison periods respectively, while the 'Memory Used by Oracle Database' column shows values of 898.1Mbyte and 899.2Mbyte.

Host Name	Total Physical Mem...	Total Memory Paging Out		Memory Used by Oracle Database	
		Base Period	Comparison Period	Base Period	Comparison Period
host01.example.co	4,006.5MByte	56.6MByte	50.3MByte	898.1Mbyte	899.2Mbyte

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

On the Resource tab, the Memory page shows a warning. There is some swapping in both periods, but the values are very similar.

# Using the DBMS\_ADDM Package

Compare two periods within the same instance:

Comparison “current” Period  
Snap\_ID **123** to **124**

to

Base “earlier” Period  
Snap\_ID **121** to **122**

```
SQL> SELECT dbms_adm.compare_instances (
  2      base_dbid          => 1319927350,
  3      base_instance_id   => 1,
  4      base_begin_snap_id => 121,
  5      base_end_snap_id   => 122,
  6      comp_dbid          => 1319927350,
  7      comp_instance_id   => 1,
  8      comp_begin_snap_id => 123,
  9      comp_end_snap_id   => 124,
 10      report_type        => 'XML')
11  FROM dual;
```



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

You can use new functions of the DBMS\_ADDM package. The DBMS\_ADDM.COMPARE\_INSTANCES returns a CLOB containing a compare period ADDM report comparing the performance of a single instance over two different time periods.

The parameters used are the following:

- BASE\_DBID: DB ID of the base period
- BASE\_INSTANCE\_ID: Instance ID of the base period
- BASE\_BEGIN\_SNAP\_ID: Snapshot ID of the beginning of the base period
- BASE\_END\_SNAP\_ID: Snapshot ID of the end of the base period
- COMP\_DBID: DB ID of the comparison period
- COMP\_INSTANCE\_ID: Instance ID of the comparison period
- COMP\_BEGIN\_SNAP\_ID: Snapshot ID of the beginning of the comparison period
- COMP\_END\_SNAP\_ID: Snapshot ID of the end of the comparison period
- REPORT\_TYPE: Output type for the report—XML or HTML (Default is HTML.)

In the slide, you see an ADDM comparison between two periods of time, each delimited in time by a begin and end snapshot.

## Using the DBMS\_ADDM Package

Functions to compare periods:

- COMPARE\_INSTANCES
- COMPARE\_DATABASES
- COMPARE\_CAPTURE\_REPLY\_REPORT
- COMPARE\_REPLY\_REPLY\_REPORT



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

Additional PL/SQL functions:

- COMPARE\_INSTANCES: Generates a Compare Period ADDM report for an instance-level performance comparison
- COMPARE\_DATABASES: Generates a Compare Period ADDM report for a database-wide performance comparison
- COMPARE\_CAPTURE\_REPLY\_REPORT: Generates a Compare Period ADDM report to compare performance for a Workload Capture to a Workload Replay
- COMPARE\_REPLY\_REPLY\_REPORT: Generates a Compare Period ADDM report to compare performance for one Workload Replay with another Workload Replay

All the preceding functions return a CLOB.

**Note:** For a complete description of the available procedures, see the *Oracle Database 12c PL/SQL References and Types* documentation.

## Quiz

The difference between a Compare Period ADDM report and an AWR Compare Periods report resides in the output format.

- a. True
- b. False



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

### Answer: b

The Compare Period ADDM report when compared with the AWR Compare Periods report performs a cause-to-effect analysis.

## Active Session History: Overview

- Stores the history of database time
- Samples session activity in the system including:
  - SQL identifier of a SQL statement
  - Object number, file number, and block number
  - Wait event identifier and parameters
  - Session identifier and session serial number
  - Module and action name
  - Client identifier of the session
  - Service hash identifier
  - Blocking session
- Is always on for first fault analysis
- Replaying the workload is not needed



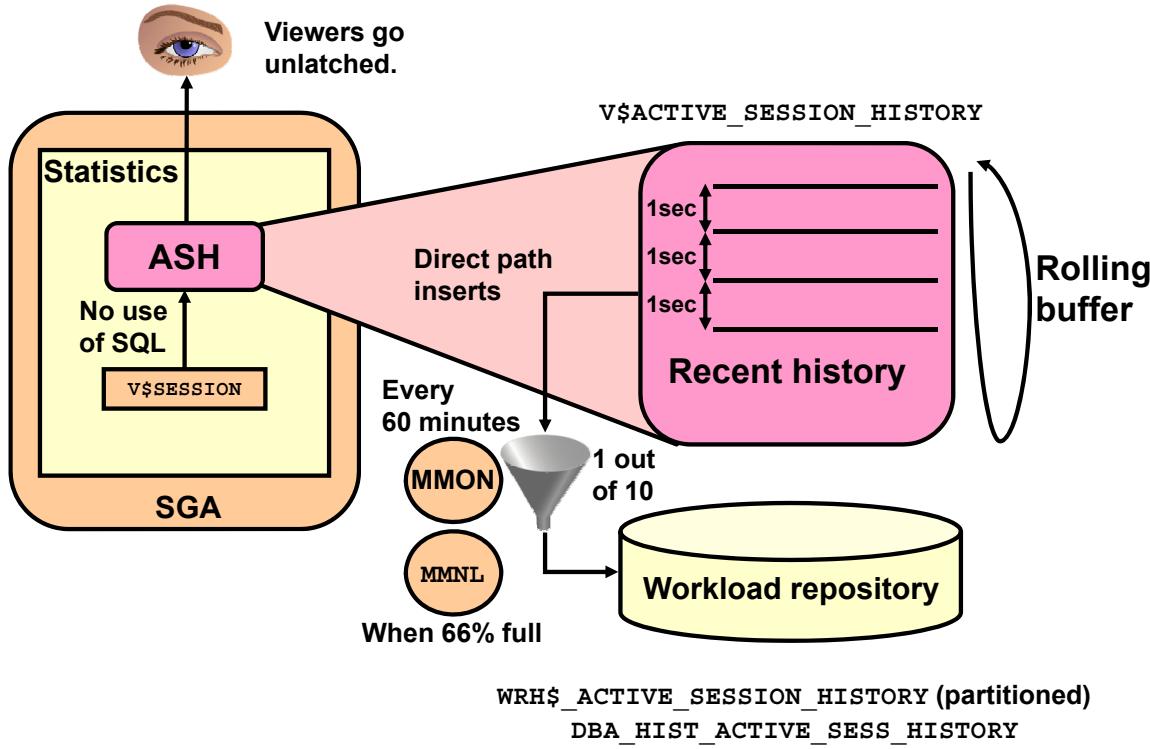
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The V\$ACTIVE\_SESSION\_HISTORY view provides sampled session activity in the instance. This can be used as a first fault system analysis. Any session that is connected to the database and is waiting for an event that does not belong to the Idle wait class is considered as an active session. This includes any session that was on the CPU at the time of sampling.

Active session samples are stored in a circular buffer in SGA. As the system activity increases, the number of seconds of session activity that can be stored in the circular buffer decreases. The time of a session sample is retained in the v\$ view. The number of seconds of session activity displayed in the v\$ view is completely dependent on database activity.

When Automatic Workload Repository (AWR) snapshots are created, the content of V\$ACTIVE\_SESSION\_HISTORY is flushed to disk. By capturing only active sessions, a manageable set of data is represented and its size is directly related to the work being performed rather than the number of sessions allowed on the system. You can examine the current Active Session History (ASH) data in the V\$ACTIVE\_SESSION\_HISTORY view and historical data in the DBA\_HIST\_ACTIVE\_SESS\_HISTORY view. You can also perform detailed analysis on this data, often avoiding the need to replay the workload to gather additional performance tracing information. The data present in the ASH can be rolled up on the various dimensions that it captures.

# Active Session History: Mechanics



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

ORACLE

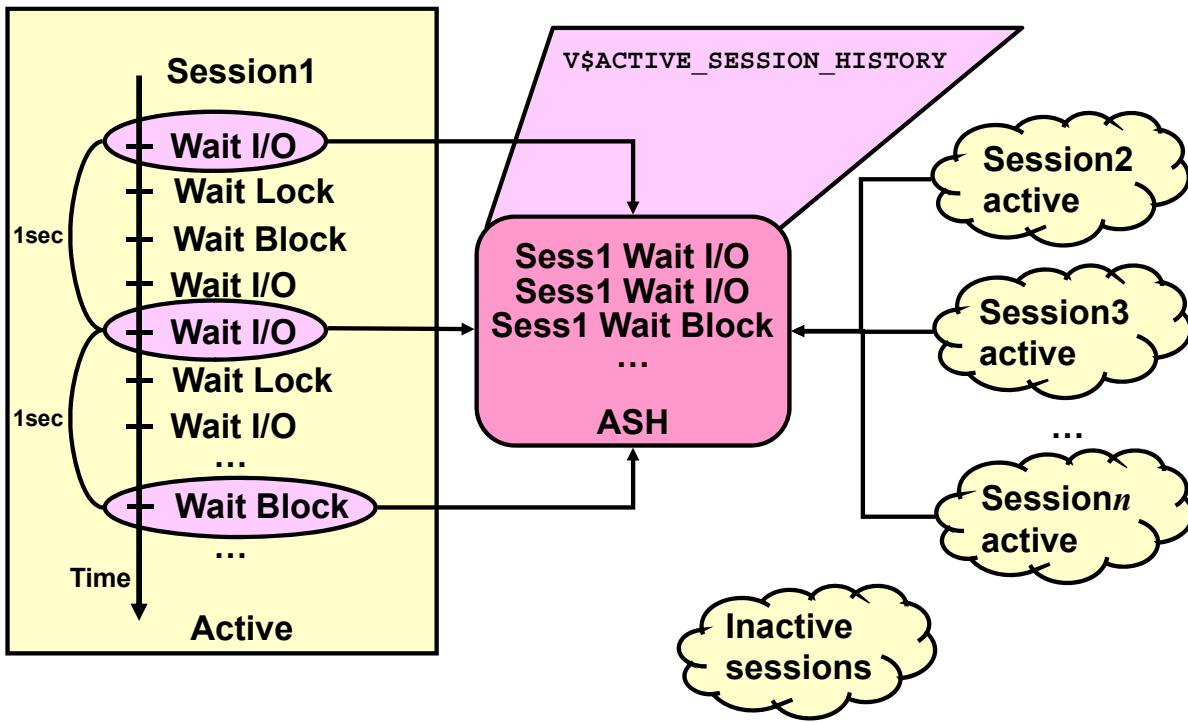
It would be very expensive to record all activities of all sessions. The ASH extracts only samples of information from V\$SESSION. One sample is extracted every second. This is done efficiently without using SQL.

ASH is designed as a rolling buffer in memory, and previous information is overwritten when needed. The volume of the data in the ASH buffer can be very large, and flushing it all to disk is unacceptable. A more efficient approach is to filter the history data while flushing it to the workload repository. This is done automatically by the manageability monitor (MMON) process every 60 minutes, and by the manageability monitor light (MMNL) process whenever the buffer is full.

The slide shows all of the nonintrusive implementation features used to capture ASH data. To further reduce possible contention, processes that only read the data (viewers), do not acquire latches.

**Note:** The memory for the ASH comes from the System Global Area (SGA), and it is fixed for the lifetime of the instance. It represents 2 MB of memory per CPU. The ASH cannot exceed a maximum bound of five percent of the shared pool size, or five percent of SGA\_TARGET.

## ASH Sampling: Example



**ORACLE**

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

As explained earlier, ASH represents the history of recent sessions activity. The diagram shows you how sessions are sampled when active. Each second, the Oracle Database server looks at active sessions, and records the events these sessions are waiting for. Inactive sessions are not sampled. The sampling facility is very efficient because it directly accesses the Oracle database internal structures. The following is the sampled information:

- SQL identifier of SQL statement
- Object number, file number, and block number
- Wait event identifier and parameters
- Session identifier and session serial number
- Module and action name
- Client identifier of the session
- Service hash identifier

ASH statistics are available through the `V$ACTIVE_SESSION_HISTORY` fixed view. This view contains one row for each active session per sample. All columns that describe the session in the ASH are present in the `V$SESSION` view.

## Accessing ASH Data

- V\$ACTIVE\_SESSION\_HISTORY
- DBA\_HIST\_ACTIVE\_SESS\_HISTORY
- ASH report
- Enterprise Manager Diagnostic Pack performance pages



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

The ASH data can be accessed in a variety of ways. You can use SQL\*Plus to access V\$ACTIVE\_SESSION\_HISTORY and DBA\_HIST\_ACTIVE\_SESS\_HISTORY. You can also generate an ASH report from SQL\*Plus by using the \$ORACLE\_HOME/rdbms/admin/ashrpt.sql script. In Enterprise Manager, you can generate an Active Session History report or you can use the Top Activity pages, accessed from the Enterprise Manager Performance page, to display the ASH data.

## Analyzing the ASH Data

- GROUP BY and COUNT
  - Proxy for non-idle elapsed time
  - Proportions of actual time spent
- Can analyze any time slice
- Example: Returns most active SQL in the past minute

```
SELECT    sql_id, count(*),
          round(count(*)/sum(count(*)) over (), 2) pctload
FROM      v$active_session_history
WHERE     sample_time > sysdate -1/24/60 and
          session_type <> 'BACKGROUND'
GROUP BY sql_id
ORDER BY count(*) desc;
```



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

You can use SQL statements to analyze any time slice of your choice using V\$ACTIVE\_SESSION\_HISTORY. However, the more samples you include in your analysis, the more accurate the results you get. As already seen, you can extract your data from the ASH by using different dimensions, such as a SQL\_ID, an ACTION, or an object number.

The SQL statement shown in the slide returns the list of the most active SQL statements executed on your instance for the last minute.

The counts in ASH are directly related to the DB Time shown in the AWR and ADDM reports. Each ASH count takes some DB Time, even though ASH data is a sample, statistically the items that take the most DB time will have the highest counts in the ASH data. The advantage of the ASH report is that you can easily see whether the wait is accumulating over time, many waits over a long period, or a few waits in a short period of time. ADDM and AWR reports usually look at a period ranging from 30 minutes to 24 hours; the ASH report allows you to examine periods measured in minutes.

**Note:** You can also use the DBA\_HIST\_ACTIVE\_SESS\_HISTORY view from the workload repository whenever you want to access data that is no longer in memory. However, the number of samples stored in the ASH on-disk version is less than its corresponding in-memory version: one sample every 10 seconds.

# Generating ASH Reports



**ORACLE**

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

The ASH Report is a digest of the ASH samples that were taken during a time period. Some of the information it shows are top wait events, top SQL, top SQL command types, and top sessions, among others. The ASH Report, which is run against collected ASH data, can be focused on a time period of days, hours, or minutes. The start and end times are not restricted to when the AWR snapshots were taken; the period can span snapshot boundaries. This makes it possible for you to focus your analysis on a very small time period; even as small as just a few minutes.

Using Enterprise Manager, you can generate the ASH Report by navigating to the Performance Home page and then clicking Run ASH Report. You can also select Top Activity in the Performance menu. On the Top Activity page, you can choose a 5-minute interval by dragging the shaded area. Then click Run ASH Report.

## Executing the ASH Report Script

```
SQL> define dbid      = '';
SQL> define inst_num   = '';
SQL> define report_type = 'html';
SQL> define begin_time  = '09:00';
SQL> define duration    = 480;
SQL> define report_name = '/tmp/sql_ashrpt.txt';
SQL> define slot_width  = '';
SQL> define target_session_id = '';
SQL> define target_sql_id   = 'abcdefghijklm';
SQL> define target_wait_class = '';
SQL> define target_service_hash = '';
SQL> define target_module_name = '';
SQL> define target_action_name = '';
SQL> define target_client_id  = '';
SQL> @?/rdbms/admin/ashrpti
```



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

Another way to generate the ASH Report is to execute the \$ORACLE\_HOME/rdbms/admin/ashrpt.sql SQL\*Plus script.

By pressing the Enter key when prompted for each variable value, you can quickly generate an ASH Report that covers the last 15 minutes of active history. Alternatively, you can define SQL\*Plus variables before invoking the script.

As shown in the slide, you can target the report for a particular ASH dimension with the ash rpti.sql script. You can define variables or enter them when prompted.

To target a specific ASH dimension, you define any number of the variables shown in the slide to the value you want reported on. This allows you to focus analysis on a single item or combination of items during a particular time span.

## ASH Report: General Section

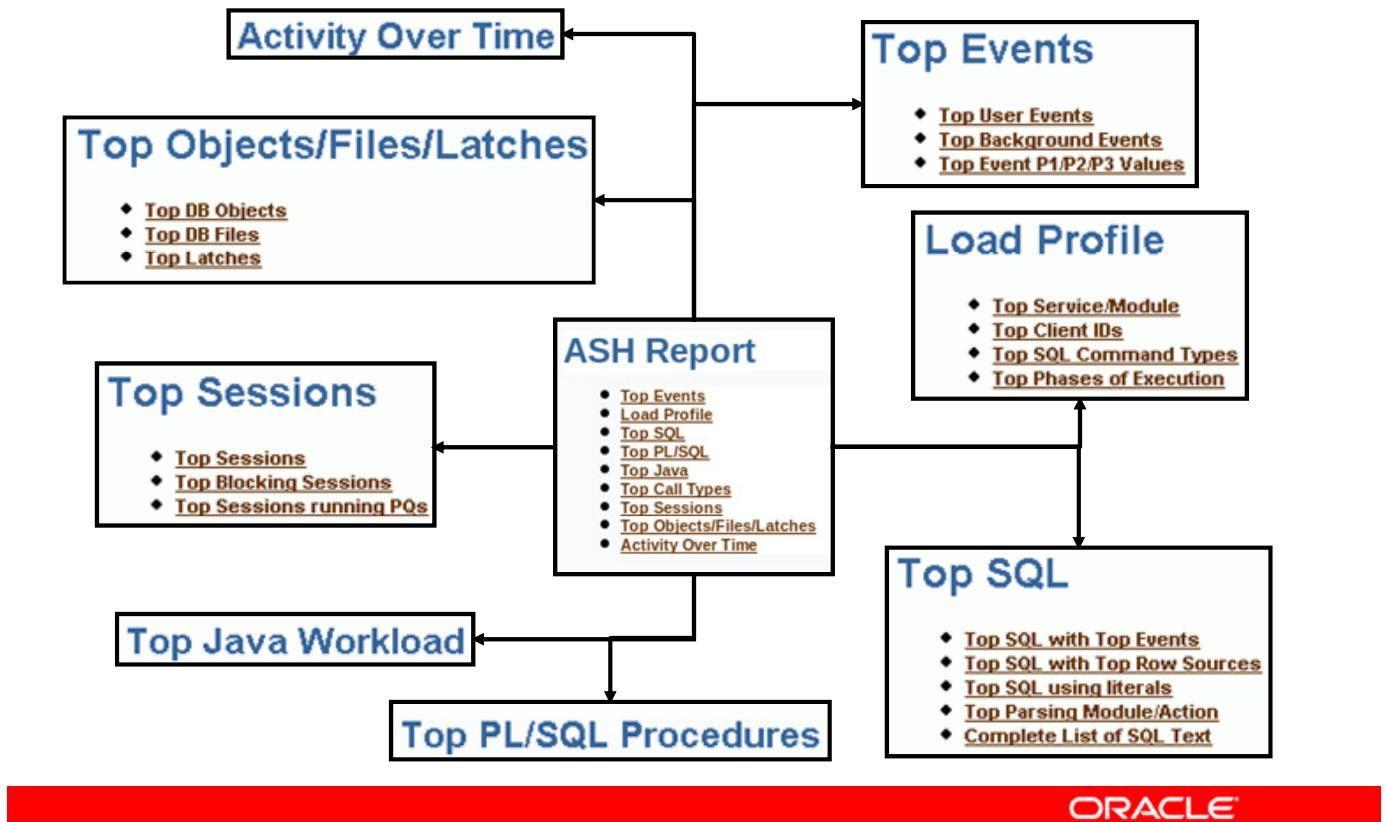
ASH Report For ORCL/orcl								
DB Name	DB Id	Instance	Inst num	Release	RAC	Host		
ORCL	1341081972	orcl	1	12.1.0.1.0	NO	EDRSR11P1		
CPUs		SGA Size		Buffer Cache		ASH Buffer Size		
2		836M (100%)		220M (26.3%)		4.0M (0.5%)		
Sample Time			Data Source					
Analysis Begin Time:	23-May-13 08:52:56		V\$ACTIVE_SESSION_HISTORY					
Analysis End Time:	23-May-13 08:57:56		VSACTIVE_SESSION_HISTORY					
Elapsed Time:	5.0 (mins)							
Sample Count:	98							
Average Active Sessions:	0.33							
Avg. Active Session per CPU:	0.16							
Report Target:	None specified							

V\$ACTIVE\_SESSION\_HISTORY ←  
DBA\_HIST\_ACTIVE\_SESS\_HISTORY ←

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

The Data Source column of the report summary shows where the data came from. This can have two different values because the data from V\$ACTIVE\_SESSION\_HISTORY, which is in-memory data, regularly gets migrated to DBA\_HIST\_ACTIVE\_SESS\_HISTORY, which is on-disk data. This migration helps to relieve the memory requirements. During migration, the data is reduced from 1-second samples to 10-second samples, thus taking up one-tenth the space on disk. Because of that difference in granularity, data that comes from memory can possibly be more accurate than that from the history view.

# ASH Report Structure



ORACLE

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

The slide shows you the various sections of the ASH Report. The ASH Report follows the pattern of the AWR report. Starting from the upper right of the slide, the report sections are as follows:

- **Top Events:** Reports the user events, background events, and the event parameter values
- **Load Profile:** Reports the top service/module and top clients; identifies the type of SQL commands and top phases of execution
- **Top SQL:** Reports the top SQL statements associated with the top events, SQL associated with the top row sources, top SQL using literals, and the SQL text for these SQL statements
- **Top PL/SQL Procedures:** Lists the PL/SQL procedures that accounted for the highest percentages of sampled session activity
- **Top Java Workload:** Describes the top Java programs in the sampled session activity
- **Top Sessions:** Reports the top sessions found waiting, top blocking sessions, and aggregates for PQ sessions
- **Top Objects/Files/Latches:** Reports the top objects, files, or latches that were involved in a wait
- **Activity Over Time:** Reports the top three wait events for 10 equally sized time periods during the report period

This report allows you to see very detailed activity over the last hour.

# ASH Report: Activity Over Time

Activity Over Time				
Slot Time (Duration)	Slot Count	Event	Event Count	% Event
08:16:00 (1.0 min)	2	control file parallel write	1	0.03
		db file sequential read	1	0.03
08:18:00 (1.0 min)	2	CPU + Wait for CPU	1	0.03
		control file parallel write	1	0.03
08:19:00 (1.0 min)	52	CPU + Wait for CPU	40	1.38
		db file async I/O submit	3	0.10
		log file parallel write	3	0.10
08:20:00 (1.0 min)	1,554	buffer busy waits	557	19.21
		free buffer waits	552	19.04
		CPU + Wait for CPU	255	8.80
08:21:00 (1.0 min)	1,162	buffer busy waits	440	15.18
		free buffer waits	352	12.14
		log file switch (checkpoint incomplete)	123	4.24
08:22:00 (1.0 min)	1	CPU + Wait for CPU	1	0.03

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

One of the most informative sections of the ASH Report is the Activity Over Time section. In this section, the ASH report time span is divided into 10 time slots. If the time period is too short or the data too sparse, the resulting report has fewer slots. Slots 2 through 9 are defined as an integer number of minutes of all the same size for easy comparison. So, it is best to compare the inner slots to one another.

Using the Activity Over Time section, you can perform skew analysis by looking for spikes in the Event Count column. This would indicate an increase in the number of processes waiting for a particular event. A spike in the Slot Count indicates an increase in active sessions because ASH data is sampled from active sessions only. In the pictured example, note the circled event. The number of active session samples has increased, and the number of sessions associated with the buffer busy waits event has also spiked. This kind of skew in a slot possibly represents the root cause of the problem being investigated.

## Additional Automatic Workload Repository Views

- DBA\_HIST\_DB\_CACHE\_ADVICE
- DBA\_HIST\_DISPATCHER
- DBA\_HIST\_DYN\_REMASTER\_STATS
- DBA\_HIST\_IOSTAT\_DETAIL
- DBA\_HIST\_SHARED\_SERVER\_SUMMARY



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

The following AWR statistics views are also available:

- DBA\_HIST\_DB\_CACHE\_ADVICE: Displays historical predictions of the number of physical reads for the cache size corresponding to each row
- DBA\_HIST\_DISPATCHER: Displays historical information for each dispatcher process at the time of the snapshot
- DBA\_HIST\_DYN\_REMASTER\_STATS: Displays statistical information about the dynamic remastering process
- DBA\_HIST\_IOSTAT\_DETAIL: Displays historical I/O statistics aggregated by file type and function
- DBA\_HIST\_SHARED\_SERVER\_SUMMARY: Displays historical information for shared servers, such as shared server activity, common queues, and dispatcher queues

# Quiz

In the Active Session History report, you can see only the last hour of data, because only one hour of data is held in memory.

- a. True
- b. False



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

## Answer: b

It is true that only an hour of ASH data is held in memory, but the memory data is sampled and placed in the `DBA_HIST_ACTIVE_SESS_HISTORY` table in the automatic workload repository, and this data is available for as long as the AWR snapshots are retained.

# Emergency Monitoring: Challenges

- Systems are sick.
- Database is slow:
  - All users' queries are very slow.
  - Performance screens show slow data refresh rates.
  - There is a significant reduction in throughput .
- Database is hung due to internal contention for resources:
  - Database is totally unresponsive; no logon is allowed.
  - Users' queries are hanging.
  - Performance screens do not refresh.



## Solution: Shut the database instance down?

ORACLE®

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

The DBA detects that the system is sick based on different symptoms:

- Users complain that their queries do not respond.
- The refresh rate of the EM Performance Page is slower and slower.
- The throughput is abnormally degrading.

You expect to be able to perform a regular performance analysis with ADDM, but there might be serious limitations:

- On slowly performing systems, the snapshots may not be available for that time period.
- Depending on severity of the performance issue, it may not be advisable or even possible to take AWR snapshots.
- You may just not be able to connect to the database because it is hanging.

Is a database instance shutdown the only solution? There might be another solution that is less drastic than bouncing the database instance.

To perform this analysis, you need to be able to connect and have a quick, lightweight analysis in order to check who is blocking the database, why it is hanging, and this analysis should not require I/O nor global resources such as enqueues or latches.

## Emergency Monitoring: Goals

Running Emergency Monitoring should be the final resort before bouncing the database.

1. Switching to Emergency Monitoring allows you to:
  - Connect to the instance in diagnostic mode
  - View the Emergency Performance page with data collected refreshed in real time
  - View ASH data and Hang Analysis table of top blocking and blocked sessions and deadlocks, refreshed in real time
2. Viewing Hang Analysis data helps you in:
  - Killing the root blocker responsible for hangs/deadlocks
  - Shutting down and starting up the instance



**ORACLE**

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

Before shutting down the instance, you can launch the Emergency Monitoring. It allows you to have a quick look and perform a performance analysis even when you cannot log to the instance with a normal connection.

Emergency Monitoring allows DBAs with SYSDBA credentials to connect in diagnostic mode and have a quick, lightweight analysis to check who is blocking the database and why it is hanging. This type of connection does not require I/O or global resources.

In Enterprise Manager 11g, “Memory Access Mode” is enabled and disabled explicitly with click of a button. This is done to start/stop the collector process, which reads performance data from SGA.

In the new approach there is no collector process. You do not have to switch between modes. When you navigate to Emergency Monitoring, the agent connects directly to the SGA and starts collecting SGA data bypassing SQL retrieval layer to get performance statistics. It will stop collecting after you return back to the regular performance monitoring.

It shows data refreshed in real time, and ASH data and Hang Analysis table of top blocking and blocked sessions, deadlocks, refreshed in real time.

It uses historical data obtained from ASH buffers to populate the performance pages.

ASH buffers generally contain history of wait data over the past 60 minutes, but this is not guaranteed. ASH buffers may not have enough history in the following cases:

- If there is too much active session activity then ASH buffer may get filled up earlier than 1 hour and flushed to the disk.
- If there is high resource contention, the process responsible for writing data into ASH buffers may get stuck and, therefore, result in loss of data.

It will, on some occasions, reduce the amount of history provided to the user from usual 30 minutes to something lesser.

When Emergency Monitoring shows the session that blocks other users, you can kill that session. Otherwise, you can either shut down the database instance or attempt a deeper analysis.

## Real-Time ADDM: Challenges

- Systems are sick.
- Database is slow:
  - All users' queries are still very slow.
  - Performance screens still show slow data refresh rates.
  - There is still a significant reduction in throughput.
- Database is hung due to other contention for resources:
  - Database might still be unresponsive; logon is allowed or is not allowed.
  - Users' queries are still waiting.
  - Performance screens do not refresh faster.
  - *You did not find any blocking session to kill.*
  - *Emergency Monitoring does not provide the root cause.*



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

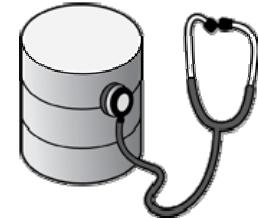
You did not find any blocking session and the database is still slow. Nevertheless, you do not want to bounce the database instance before trying another analysis.

Real-Time ADDM offers you the possibility to perform a complementary analysis, a root-cause analysis that Emergency Monitoring does not provide.

## Real-Time ADDM: Goals

1. Switch to Real-Time ADDM before bouncing the instance:
  - Starts collecting performance data from all database instances
  - Analyzes recent data for systems paralyzed because of severe contention on local or global resources
  - Provides holistic analysis for systems experiencing unusually high (though not severe) database activity
  - Detects findings for the recent activity (past 10 minutes)
  - Offers actionable recommendations
2. Use the recommendations to solve.
3. Return back to regular Performance Monitoring.

**Note:** Can be invoked for a RAC environment



**ORACLE**

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

Real-Time ADDM has two components: one in EMCC which is described here, and another in the database, which is automatic when triggered by certain conditions.

Real-Time ADDM in EMCC works in a very similar fashion as ADDM to analyze performance.

Regular ADDM runs using AWR snapshots and provides findings and recommendations to let you know what needs to be changed to improve the performance.

Real-Time ADDM avoids I/O by accessing ASH recent activity from SGA data.

You connect in either mode with the `SYSDBA` privilege, depending on the state of the instance:

- **Diagnostic mode:** If you cannot get a connection
- **Normal mode:** If you can still connect

Then the analysis starts, collecting recent performance data from all database instances from SGA, analyzing data for systems that are paralyzed because of severe contention on local or global resources.

It provides holistic analysis for systems experiencing unusually high (though not severe) database activity. After the analysis is complete, you can ask for the findings.

Real-Time ADDM can find that audit tracing cannot be performed due to lack of disk space preventing connections, or that a session needs to be killed because a logon trigger attempts to lock a table already locked, thereby preventing any further connections. Real-Time ADDM identifies long-running queries that consume all resources.

In a case when a session blocks other users, the Hang Data tab presents the Hang Analysis page with the Final Blockers and the Blocked Sessions. You will see the details of the blocker session and thus be able to get the process ID killed.

The Top Activity Snapshot presents the top activity of the instance for the past 10 minutes. Because Real-Time ADDM focuses on the last recent period, it is useful only for clarifying current performance issues.

Always review the recommendations to get help on different solutions. Then return back to regular performance monitoring.

Real-Time ADDM can be invoked for a RAC environment in the same way as single instances.

## Real-Time ADDM in the Database

- Analyzes and helps to resolve problems in hung and unresponsive Oracle Database systems
- Runs automatically every three seconds
- Employs controls to ensure that it does not adversely impact the database management system
- ADDM recommends possible solutions as a set of findings.



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

Real-Time ADDM helps you analyze and resolve problems such as deadlocks and shared pool connections in unresponsive or hung databases. Real-Time ADDM uses a set of predefined criteria to analyze the current performance of the database. After analyzing the problem, Real-Time ADDM helps you to resolve the identified issue without having to restart the database.

Real-Time ADDM in the database proactively detects transient database performance issues by running automatically every three seconds, and using in-memory data to diagnose performance spikes in the database.

The following issues and conditions trigger a Real-Time ADDM analysis:

- **High load:** Average active sessions are greater than three times the number of CPU cores.
- **I/O bound:** I/O impact on active sessions based on single block read performance
- **CPU bound:** Active sessions are greater than 10% of the total load and CPU utilization is greater than 50%.
- **Over-allocated memory:** Memory allocations of more than 95% of physical memory
- **Interconnect bound:** Based on single block interconnect transfer time
- **Session limit:** Is close to 100%

- **Process limit:** Is close to 100%
- **Hung sessions:** are greater than 10% of total sessions.
- **Deadlock detected:** Any deadlock is detected.

To ensure that Real-Time ADDM automatic triggers do not impact the system in an adverse manner, the following controls are employed:

- If a Real-Time ADDM report was created in the past 5 minutes by the automatic trigger, no new reports will be generated.
- In an Oracle RAC configuration, only one database instance can create a Real-Time ADDM report at a given time.
- A new report is generated if a new issue is detected and the same issue was not previously detected within the past 45 minutes.

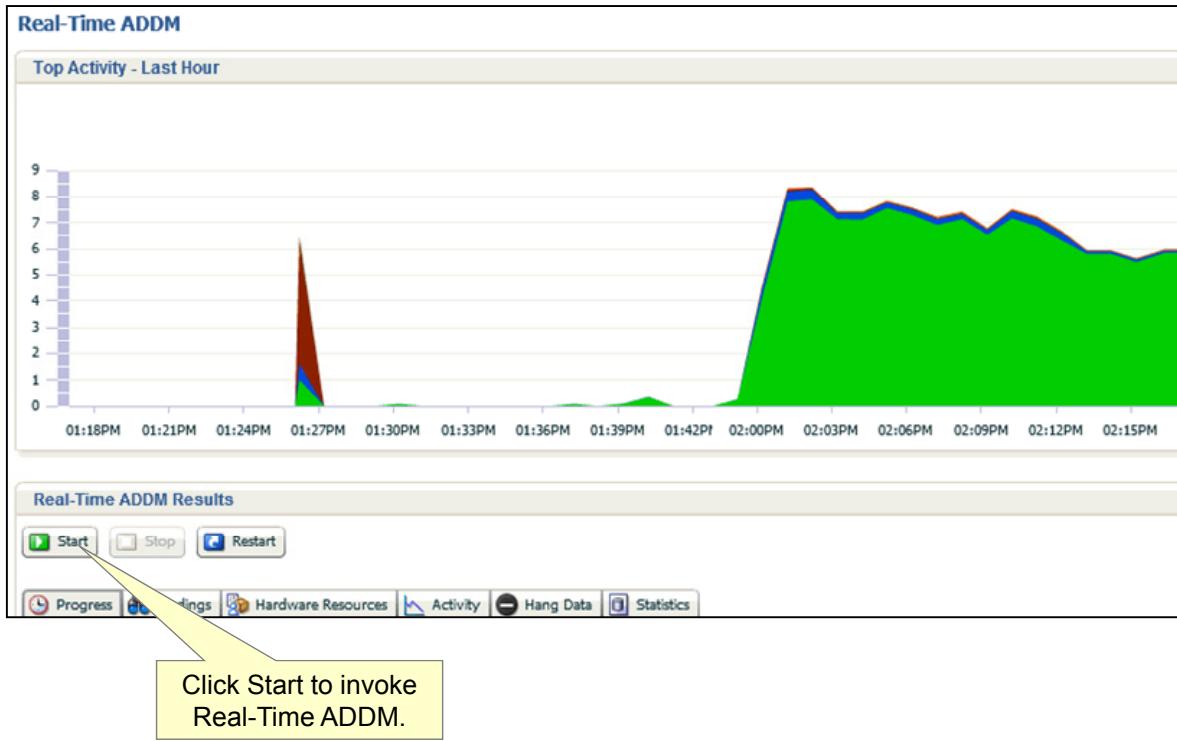
In addition to diagnosing problems, possible solutions are presented as a set of findings.

Types of ADDM findings are as follows:

- **Problem findings:** Describe the root cause of a database performance problem
- **Symptom findings:** Contain information that may lead to one or more problem findings
- **Information findings:** Provide information that is relevant to understanding the performance of the database, but do not constitute a performance problem
- **Warning findings:** Contain information about problems that may affect the completeness or accuracy of the ADDM analysis

A problem finding can be associated with a list of recommendations that will reduce the impact of the performance problem. Each recommendation includes an estimate of the portion of DB time that can be saved if the recommendation is implemented.

# Using Real-Time ADDM

**ORACLE**

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

Real-Time ADDM can be invoked in Enterprise Manager Cloud Control. Select Real-Time ADDM in the Performance menu. On the Real-Time ADDM page, click Start.

The system then collects performance data from the target database, analyzing data for problem diagnosis and resolutions, and displaying the results of the analysis.

# Viewing Real-Time ADDM Results

The screenshot shows the 'Real-Time ADDM Results' window. At the top, there are 'Start', 'Stop', and 'Restart' buttons. Below them is a navigation bar with tabs: 'Progress' (selected), 'Findings' (highlighted with a yellow arrow), 'Hardware Resources', 'Activity', 'Hang Data', and 'Statistics'. The status is shown as 'FINISHED'. The start time is 'Thu Aug 15, 2013 9:17:00 AM' and the end time is 'Thu Aug 15, 2013 9:22:49 AM'. The number of findings is '0'. Two sections are displayed: 'Normal Connection' and 'Diagnostic Connection', each listing various metrics checked off with green checkmarks.

Normal Connection	Diagnostic Connection
JDBC Connection to the Database	Acquire SYSDBA Credentials
Basic Meta-data (database version and number of i	Session Activity by Wait Classes
Database and Instance Meta Data	Hang Analysis Data
Hang Analysis Data	I/O Metrics
Database Metrics	Host Metrics
Raw ASH Data	
Host Hardware Information	
Instance Memory Information	
Shared Pool Resize Operations	
Database Initialization Parameters	
Session Activity by Wait Classes	
Top SQL from ASH	

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

The analysis results is displayed, including an indication of the number of findings. Click the Findings tab for a summary of the findings the analysis has detected and to view recommendations.

When you run Real-Time ADDM in EMCC, the report is not persisted in the database, but you can save it in the file system.

When Real-Time ADDM is run automatically because of a triggering event, the report is persisted and can be accessed from the Advisors Home page.

# Quiz

The difference between Real-Time ADDM and regular ADDM resides in the source used for analysis.

- a. Real-Time ADDM uses the AWR snapshots of the last 10 minutes.
- b. Real-Time ADDM uses the ASH recent activity from SGA data.
- c. Regular ADDM uses the AWR snapshots that are not yet purged.



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

## Answer: b, c

Real-Time ADDM works in a very similar fashion as regular ADDM to analyze performance; Real-Time ADDM does not access the AWR snapshots but accesses Active Session History recent activity from SGA data.

## Summary

In this lesson, you should have learned how to:

- Tune Automatic Maintenance Tasks
- Generate ADDM reports
- Generate ASH reports



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

## **Practice 7 Overview: Using AWR-Based Tools**

This practice covers the following topics:

- Generating an AWR report
- Creating a preserved set on snapshots
- Generating an ADDM report
- Generating an ASH Report



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



# Real-Time Database Operation Monitoring

8

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 database operations
- Implement Real-Time Database Operation Monitoring

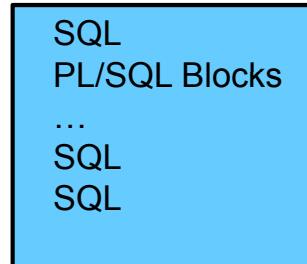


Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

# Overview

- A database operation is:
  - One or more SQL or PL/SQL statements
  - In one or more sessions
- A database operation can:
  - Be monitored
  - Produce active reports

**BEGIN\_OPERATION**



**END\_OPERATION**

**ORACLE**

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

Real-Time Database Operation Monitoring extends and generalizes Real-Time SQL Monitoring.

Real-Time SQL Monitoring helps to determine where a currently executing SQL statement is in its execution plan and where the statement is spending its time.

You can use Real-Time Database Operation Monitoring for performance monitoring of active SQL statements and PL/SQL procedures and functions. You can also see the breakdown of time and resources used for recently completed statements.

In databases where batch jobs run regularly, it can be very difficult to determine the batch job to which an offending SQL statement belongs. In decision support systems (DSS), it is not uncommon for many batch jobs to be running simultaneously. In some systems, hundreds of concurrent jobs could be running. With Real-Time Database Operation Monitoring, you can monitor these jobs while they are running.

Real-Time Database Operation Monitoring identifies an operation with a tag, determines what to monitor, and generates reports. It monitors the progress of the operation.

# Use Cases

- Monitor batch jobs:
  - Set expected times; alert when exceeded.
  - Identify and tune expensive statements.
- Diagnose background processes:
  - Identify and tune expensive statements.
- Diagnose changes to jobs over time:
  - Compare previous executions of an operation.
  - Compare changes to jobs after an upgrade.



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

In which situations would a DBA use Real-Time Database Operation Monitoring?

A large retailer has a DSS with hundreds of batch jobs for extraction, transformation, and loading (ETL). You want to monitor these jobs in real time. As an Enterprise Manager user, you experience a very long wait on a page load. You want to easily identify what is running on behalf of the page load. At times a PL/SQL job that is required to finish in 24 hours is taking up to 80 hours to run. You need to identify and tune the expensive statements in the job. You want to be alerted if the job is expected to run too long, so that you can fix it and restart it before it uses too much time.

After an upgrade, a batch job went from 4.5 hours to 8 hours. You want to capture the performance statistics of the job steps before and after the upgrade, and then compare the two sets and identify the changes, for example, SQL execution plan, resource consumption, degree of parallelism.

# Defining a DB Operation

- Any set of database operations between two points in time identified by the user or application
- Types of operations:
  - Simple:
    - One SQL statement
    - One PL/SQL function/procedure
  - Composite:
    - Multiple SQL statements and/or PL/SQL procedures and functions between two points in time
    - Single session
    - Multiple concurrent sessions



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

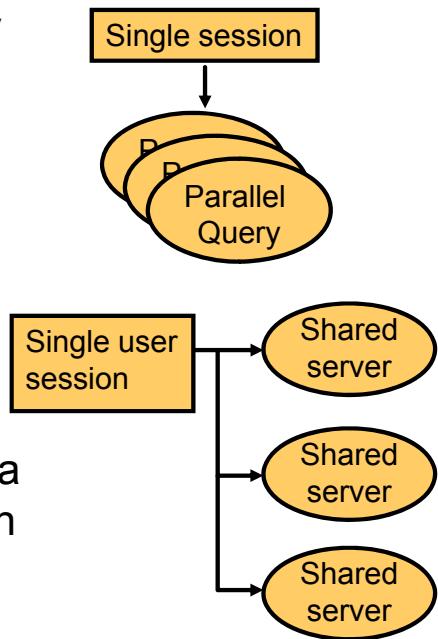
A DB operation is an operation that the database server runs to accomplish tasks. It can be simple or composite:

- **Simple database operations:** Consist of a single SQL statement or a single PL/SQL function or procedure.
- **Composite database operations:** Consist of the activity of one or more sessions between two points in time. SQL statements or PL/SQL procedures running in these sessions are part of the composite operations:
  - **Single sessions:** Single-session operations fall into two categories. In one category, exactly one session exists for the duration of the database operation. In the other category, multiple sessions exist in the database operation, but no more than one session runs at any given time. The application jumps from session to session.
  - **Multiple concurrent sessions:** In ETL, it is common for a job to involve multiple sessions running at the same time. You can define the entire ETL job as a single database operation.

# Scope of a Composite DB Operation

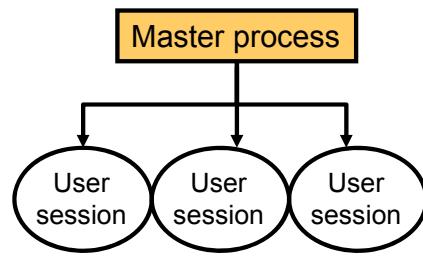
## Single Session

- Parallel query with single coordinator
- Single-user session connected to a server session sequentially



## Multiple Sessions

An operation using multiple-user sessions concurrently



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

A composite operation in the scope of a single session comprises one of the following:

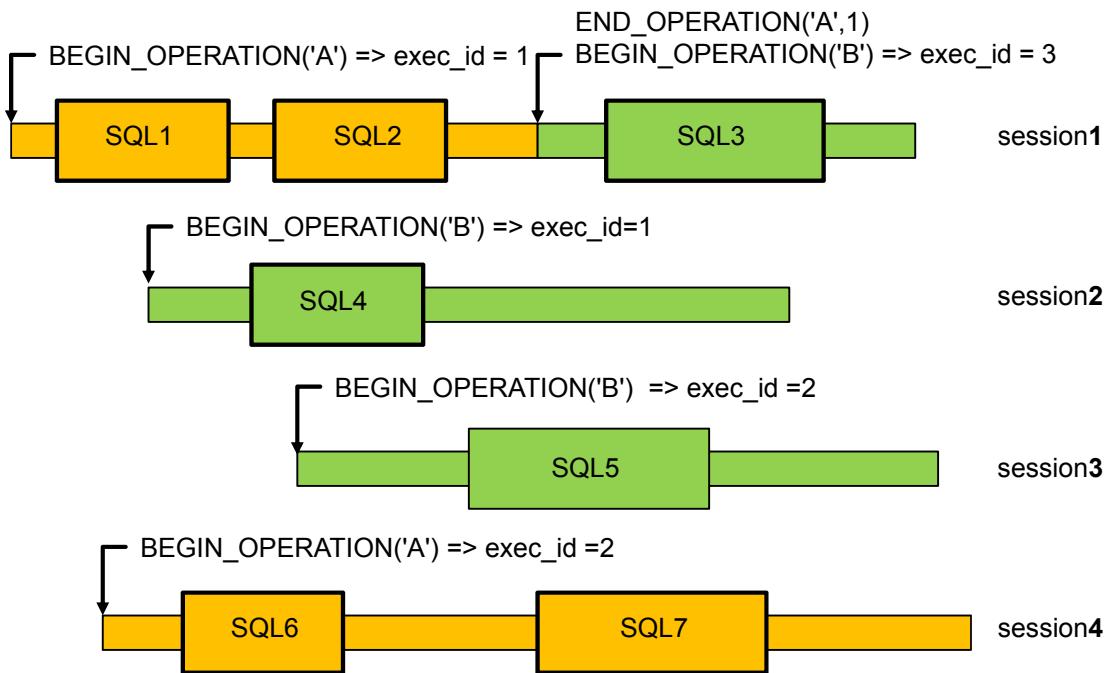
- SQL statements coordinated by the single session, such as parallel operations
- The active session, such as a user session in an application pool or shared server

Diagrams illustrating single-session and single-user session examples of a database operation are shown on the left side of the slide.

**Note:** Also in the scope of a single session are statements that are parallel using one coordinator session or statements that use shared servers with a single-user session spanning multiple servers with only one active at a time.

A composite operation in the scope of multiple sessions comprises SQL statements running concurrently, such as during an ETL operation. This example is shown in the diagram on the right side of the slide.

# Database Operation Concepts



**ORACLE**

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

This illustration shows four sessions. The first (upper) session shows that a session can belong to at most one database operation at a time. It belongs to database operation A at first, and then belongs to database operation B. The `END_OPERATION` and then `BEGIN_OPERATION` commands cause the session to change database operations.

Sessions 1, 2, and 3 show that the name and execution ID are taken together to uniquely identify database operation B.

The time that a session belongs to a database operation, but is not executing a SQL or PL/SQL statement, is idle time.

# Identifying a Database Operation

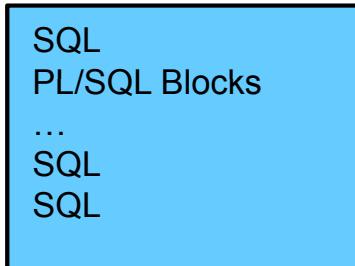
- Naming

**DBOP (Name)**



- Bracketing

**DBMS\_SQL\_MONITOR.BEGIN\_OPERATION**



**DBMS\_SQL\_MONITOR.END\_OPERATION**

**ORACLE**

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

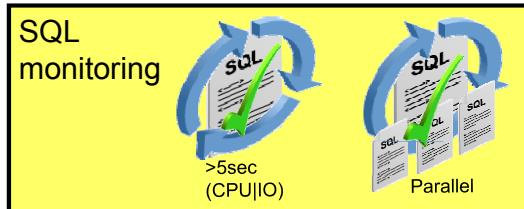
To monitor a database operation, you must give the database operation a name, a begin point, and an end point. You can name a database operation in one of two ways either:

- Insert the following into the application: a start point by using the `DBMS_SQL_MONITOR.BEGIN_OPERATION` function and an end point by using the `DBMS_SQL_MONITOR.END_OPERATION` procedure calls.
- Or set a tag for Java and Oracle Call Interface (OCI) applications by using the `OCIAttrSet` and `OCIAppCtxSet` OCI calls, the `setClientInfo` Java call, or the `ORA_DBOP` OS environment variable. The tag in Java or OCI piggybacks the next database call so that the begin point is easily distinguished.

The explicit calls provide a clear beginning and ending of a database operation to provide the bracketing, but applications using tagging do not have an explicit method for ending an operation. Because starting a new operation ends the previous operation, you should set the tag to `NULL` and send it to the database to force the end of an operation.

# Enabling Monitoring of Database Operations

- At the system level



- At the database operation level
- At the statement level



**ORACLE®**

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

The SQL monitoring feature is enabled by default when you set the `STATISTICS_LEVEL` initialization parameter either to `TYPICAL` (the default value) or to `ALL`. You must also set the `CONTROL_MANAGEMENT_PACK_ACCESS` parameter to `DIAGNOSTIC+TUNING` (the default value) because SQL monitoring is a feature of the Oracle Tuning Pack. By default, SQL monitoring starts when a SQL command runs in parallel or when it has consumed at least five seconds of the CPU or I/O time in a single execution. A SQL statement is also monitored when the `MONITOR` hint is explicitly declared in the statement.

When starting a database operation, you can force the operation to be tracked when it starts by using the `FORCED_TRACKING` parameter. When you set this parameter to the default value, '`N`', the database operation is tracked only when it is sufficiently expensive. To force tracking, set this parameter to '`Y`' when you start the database operation. The specification for `DBMS_SQL_MONITOR.BEGIN_OPERATION` function is:

```
DBMS_MONITOR.BEGIN_OPERATION (
    dbop_name IN VARCHAR2,
    dbop_eid IN NUMBER :=NULL,
    forced_tracking IN VARCHAR2 := 'N',
    attribute_list IN VARCHAR2 :=NULL)
RETURN NUMBER;
```

# Identifying, Starting, and Completing a Database Operation

1. Identify the database operation by operation name and execution ID.
2. Start the database operation with DBMS\_SQL\_MONITOR.BEGIN\_OPERATION.
3. Complete the database operation with DBMS\_SQL\_MONITOR.END\_OPERATION.

```
SQL> VAR dbop_eid NUMBER;
SQL> EXEC :dbop_eid := DBMS_SQL_MONITOR.BEGIN_OPERATION
          ('ORA.MV.refresh', FORCED_TRACKING => 'Y')
SQL> SELECT ...
SQL> SELECT ...
SQL> EXEC DBMS_SQL_MONITOR.END_OPERATION('ORA.MV.refresh'
                                         :dbop_eid)
```



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

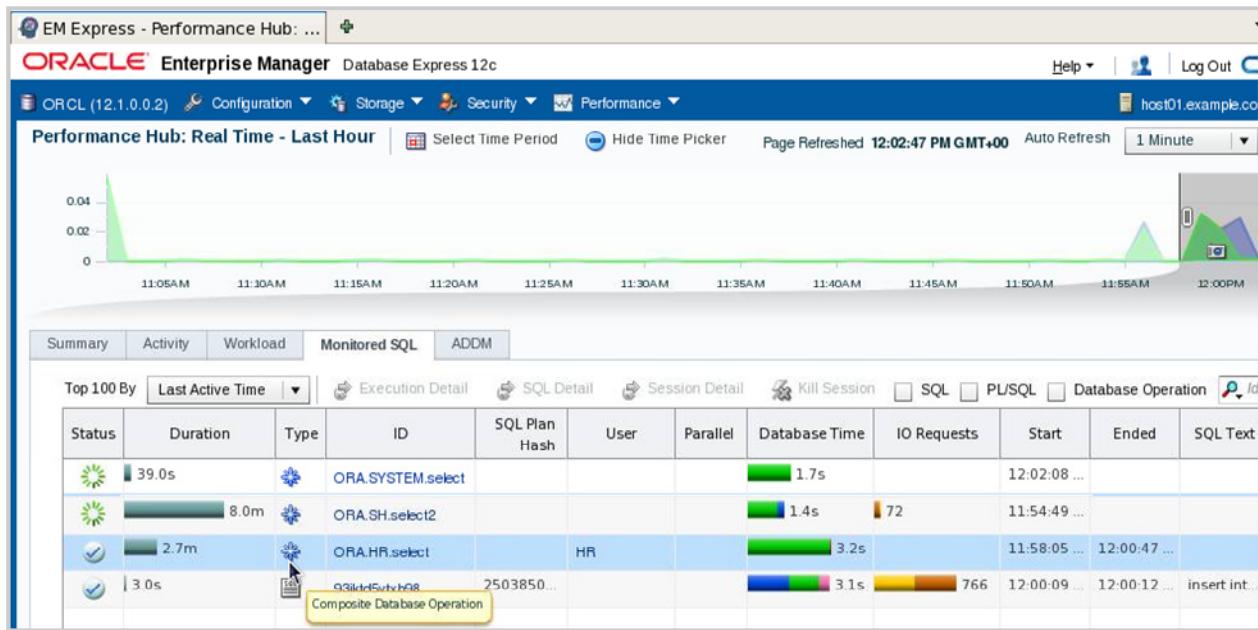
When monitoring, identify each DB operation by the operation name and execution ID, which together uniquely identify a specific execution of the operation. The operation name is specific to a piece of code. The same code may be executed by multiple sessions or users at the same time. Each invocation has the same operation name but different executions IDs. The execution ID is not unique between operations; that is, two DB operations with different names could have the same execution ID.

The BEGIN\_OPERATION call returns the execution ID. If an execution ID is supplied to the call, that ID is assigned and returned. If the execution ID is not supplied, a unique execution ID is returned. If a master process spawns several sessions that are associated with the same DB operation, an execution ID is generated for the entire operation, and each session sets the execution ID with a BEGIN\_OPERATION call.

The database operation names are in a single namespace. A recommendation to prevent collisions is to use a format of <component>.<subcomponent>.<operation\_name>. A suggestion is to use a component name of ORA for operations inside the database. The example in the slide shows an operation that performs a materialized view refresh.

If different code paths are taken through conditions in PL/SQL, the different executions of the same database operation (with the same name) can have different SQL statements or PL/SQL functions. There is also nothing to prevent one operation name from being assigned to multiple sets of code, which could cause confusion.

# Monitoring the Progress of a Database Operation



**ORACLE®**

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

With Enterprise Manager Database Express and Enterprise Manager Cloud Control, you can view database operations that are active and recently finished.

On the Database Home page of Enterprise Manager Database Express, the list of monitored SQL and database operations appears at the bottom of the page, as shown in the slide.

In the Performance menu, select the Performance Hub option to get the list of monitored SQL and database operations. For details, click the database operation name. The execution ID is shown on the Details page. By selecting the Database Operations filter above the list (SQL, PL/SQL, and Database Operations), you ensure that only database operations are shown. Use the search box on the right above the list to filter further on the operation name.

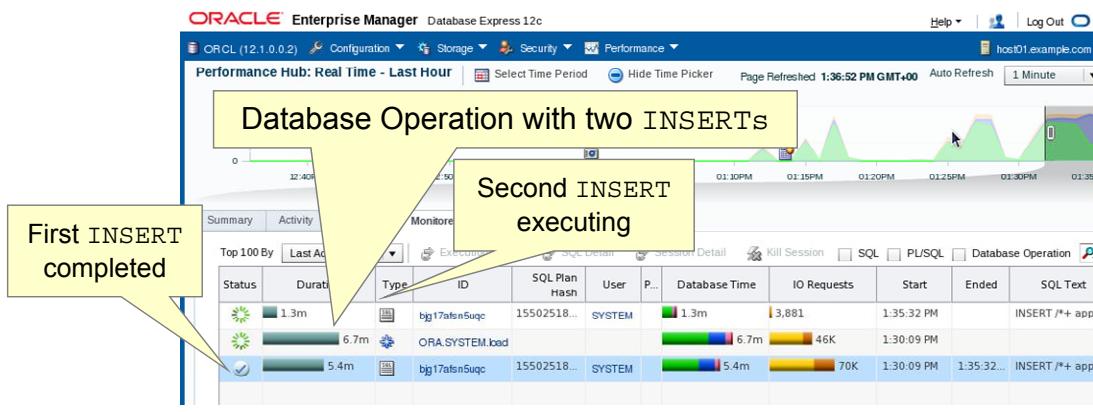
To navigate to the Monitored SQL page by using Enterprise Manager Cloud Control, log in to the target database, click the Performance menu, and then select the SQL Monitoring option.

# Monitoring Load Database Operations

```

SQL> VAR op_eid NUMBER;
SQL> EXEC :op_eid := DBMS_SQL_MONITOR.BEGIN_OPERATION
      ('ORA.SYSTEM.load', FORCED_TRACKING => 'Y')
SQL> INSERT /*+ APPEND */ * INTO tab1 SELECT ...;
SQL> INSERT /*+ APPEND */ * INTO tab2 SELECT ...;
SQL> EXEC DBMS_SQL_MONITOR.END_OPERATION
      ('ORA.SYSTEM.load', :op_eid)
    
```

One load is completed and the DB operation is still executing.



ORACLE

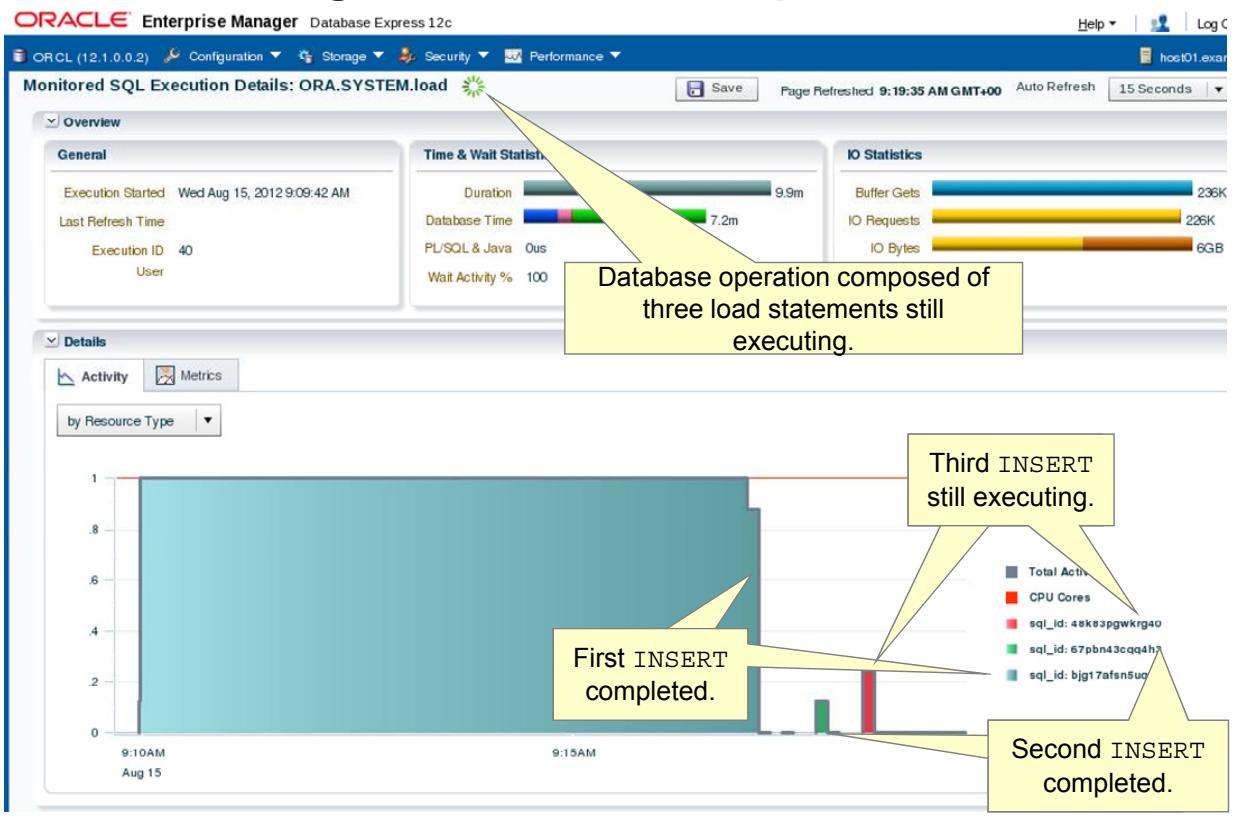
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Data load operations can be monitored. For the example in the slide, the database operation consists of two bulk load statements. You can see three rows, one for the database operation as a whole and one for each of the two insert statements.

On the Performance Hub page and the Monitored SQL tab in Enterprise Manager Database Express, you can see that the database load operation is still executing. You can see that the first bulk load statement is already completed, and the second bulk load statement is still executing.

In Enterprise Manager Cloud Control, if you log in to a target database, you can see the same execution of the database load operation by selecting SQL Monitoring from the Performance menu.

# Monitoring Load Database Operation Details



**ORACLE**

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

The execution details page shows a summary at the top of the screen and either the activity graph (shown) or the metrics graphs (not shown) at the bottom of the screen. The DBMS\_SQL\_MONITOR.REPORT\_SQL\_MONITOR and DBMS\_SQL\_MONITOR.REPORT\_SQL\_MONITOR\_LIST functions are used to report execution details of real-time database operations.

## Database Operation View: V\$SQL\_MONITOR

The V\$SQL\_MONITOR view shows the list of executing and completed database operations.

```
SQL> SELECT DBOP_NAME, DBOP_EXEC_ID, STATUS
  2  FROM V$SQL_MONITOR
  3 WHERE DBOP_NAME IS NOT NULL;

DBOP_NAME          DBOP_EXEC_ID STATUS
-----  -----
ORA.HR.select      8 EXECUTING
ORA.SH.select2    10 EXECUTING
ORA.SH.select     3 DONE
```



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

The V\$SQL\_MONITOR view reports a list of executing and completed database operations, including global, high-level information about the top SQL statements in a database operation. Each monitored SQL statement has an entry in this view. Each row contains a SQL statement whose statistics are accumulated from multiple sessions and all of its operation executions.

The primary key is the combination of the DBOP\_NAME, DBOP\_EXEC\_ID, and SQL\_ID columns.

## Database Operation Views

- Enterprise Manager Cloud Control and Enterprise Manager Database Express use multiple views.
- AWR automatically captures SQL monitoring reports and stores active session history.

```
SQL> SELECT session_id, DBOP_NAME, DBOP_EXEC_ID
  2  FROM DBA_HIST_ACTIVE_SESS_HISTORY
  3  where DBOP_NAME is not NULL;

SESSION_ID DBOP_NAME          DBOP_EXEC_ID
-----  -----
      30 ORA.SYSTEM.load        8
      45 ORA.HR.select         11
```



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

Enterprise Manager Cloud Control and Enterprise Manager Database Express summarize the activity for monitored statements and database operations collected from multiple views.

- Recently completed and active database operations from:
  - V\$SQL\_MONITOR, V\$ACTIVE\_SESSION\_HISTORY by using the new columns DBOP\_NAME and DBOP\_EXEC\_ID.
  - V\$SQL\_PLAN\_MONITOR, V\$SQL\_MONITOR\_SESSTAT, V\$SQL, V\$SQL\_PLAN, V\$SESSION\_LONGOPS
- Completed database operations and reports are stored in AWR and can be reported in the following views:
  - DBA\_HIST\_REPORTS, DBA\_HIST\_REPORTS\_DETAILS
  - DBA\_HIST\_ACTIVE\_SESS\_HISTORY

# Reporting Database Operations by Using Functions

The DBMS\_SQL\_MONITOR package:

- REPORT\_SQL\_MONITOR\_LIST\_XML and REPORT\_SQL\_MONITOR\_XML functions report the list and details of database operations in XML format.
- REPORT\_SQL\_MONITOR\_LIST and REPORT\_SQL\_MONITOR functions report the list and details of database operations in a character large object (CLOB).

```
SQL> SELECT DBMS_SQL_MONITOR.REPORT_SQL_MONITOR_LIST_XML()
  2  FROM dual;
```



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

These reporting functions provide a simple way to produce reports in your preferred format.

# Database Operation Tuning

- ADDM
- SQL Tuning Advisor
- SQL Performance Analyzer



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

The goal of tuning is to improve the response time of a specific execution. At this time, to tune a database operation, you will tune the top SQL statements one by one. Database operation tuning could involve more than just tuning the SQL statements in the operation.

Here are the tools that you can use for SQL tuning:

- **Automatic Database Diagnostic Monitor (ADDM):** ADDM analyzes the information collected by the AWR for possible performance problems with Oracle Database, including high-load SQL statements.
- **SQL Tuning Advisor and SQL Performance Analyzer:** These tools can take SQL tuning sets or top SQL statements as targets.

# Quiz

Monitoring database operations is allowed only for sets of SELECT statements.

- a. True
- b. False



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

**Answer: b**

## Quiz

You can define any set of SQL and PL/SQL statements (including multiple, concurrent sessions) between two points in time as a database operation to be monitored.

- 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 database operations
- Implement Real-Time Database Operation Monitoring



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

## **Practice 8 Overview: Real-Time Database Operation Monitoring**

- Creating DB operations:
  - Modify a batch script with multiple SQL statements to be treated as an operation.
- Monitoring DB operations:
  - Execute a script that has been marked as multiple operations.
  - View the SQL Monitoring results.



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



# 9

## Monitoring Applications

ORACLE®

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

# Objectives

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

- Configure and manage services
- Use services with client applications
- Use services with the Database Resource Manager
- Use services with the Scheduler
- Set performance-metric thresholds on services
- Configure services aggregation and tracing



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

## What Is a Service?

- A means of grouping sessions by type of work
- An abstraction that presents data independent of the instance
- A part of the regular administration tasks that provide dynamic service-to-instance allocation
- The base for high availability of connections
- An additional performance tuning dimension



**ORACLE**

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

Services organize work execution within the database to make that more manageable, measurable, tunable, and recoverable. A service is a group of related tasks within the database with common functionality, quality expectations, and priority relative to other services.

Services are an abstraction layer permitting clients and middle tiers to access required data from the database independent of where the instance or instances reside. Services thus provide a single-system image for managing competing applications running within a single instance and across multiple instances. By using standard interfaces, such as Database Configuration Assistant (DBCA), Enterprise Manager, and Server Control (SRVCTL) services can be configured, administered, enabled, disabled, and measured each as a single entity.

Services provide availability. Following outages, a service is recovered automatically at surviving instances.

Services provide an additional dimension to performance tuning. With services, workloads are visible and measurable. Tuning by “service and SQL” replaces tuning by “session and SQL” in systems where sessions are anonymous and shared.

Services are dynamic. The number of instances a service runs on can be augmented when load increases, and reduced when load declines. This dynamic resource allocation enables a cost-effective solution for meeting demands as they occur.

Services were designed for multiple instance databases, but provide a convenient way to monitor multiple applications on a single instance database.

## Service Attributes

- Global unique name
- Network name
- Load Balancing Advisory goal
- Distributed transactions flag
- Advance queuing notification characteristics for OCI and ODP.NET clients
- Failover characteristics
- Connection load-balancing algorithm
- Threshold
- Priority
- High-availability configuration



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

When you create new services for your database, you should define each service's workload management characteristics. In a single instance database; most of these characteristics do not apply. In a single instance database, service name, network name, and global unique name are all that are required. In addition to the required service name and network name, the characteristics of a service include:

- A service goal that determines whether work requests are made to the service based on best service quality (service response time), or best throughput (how much work is completed in a unit of time), as determined by the Load Balancing Advisory
- The session failover characteristics when using transparent application failover
- The method for load balancing connections (which you define) for each service:
  - SHORT: Using Load Balancing Advisory (open workloads)
  - LONG: Using session count by service (closed workload)
- Services metric thresholds for response time and CPU consumption
- Mapping of services to consumer groups instead of usernames to consumer groups
- How the service is distributed across instances when the system first starts

**Note:** For more information, refer to the *Oracle Clusterware and Oracle Real Application Clusters Administration and Deployment* guide.

# Service Types

- Application services
- Internal services:
  - SYS\$BACKGROUND
  - SYS\$USERS
  - Cannot be deleted or changed
- Limit of 1024 services per database:
  - 1019 application services
  - 2 internal services
  - 3 services by default



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

Oracle Database supports two broad types of services: application services and internal services. Application services are mainly functional maps to workloads. Sessions doing work for a common business function are grouped together. For Oracle Applications, AP, AR, GL, MFG, WIP, BOM, and so on create a functional division of work within the database and can thus be categorized as services.

In addition to application services, the RDBMS also supports two internal services. SYS\$BACKGROUND is used by the background processes only. SYS\$USERS is the default service for user sessions that are not associated with any application service. Both internal services support all workload management features and neither can be stopped or disabled.

There is a limitation of 1024 services per database: 1019 application services, and 2 internal services. There may also be a few services created by DBCA. Three services are created by default, such as the <dbname>XDB service, which is required by most databases. A service name is restricted to 64 characters.

**Note:** Shadow services used by Transparent Application Failover are also included in the application services category. In addition, a service is also created for each Streams Buffered Queue.

# Creating Services

- Services are maintained in the data dictionary.
- Use DBMS\_SERVICE.CREATE to create a service for a single-instance database.
- Services are created automatically based on the SERVICE\_NAMES initialization parameter.
- Create services by using the following:
  - SRVCTL (Oracle Clusterware, Oracle Restart, or Oracle GlobalData Services)
  - Enterprise Manager



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

Like other database objects, services are maintained and tracked through the data dictionary and dynamic performance views.

Each service has a unique name that identifies it locally in the cluster and globally for Data Guard.

For a single-instance environment, services can be created by using the DBMS\_SERVICE package.

Services are also created implicitly at the startup of the instance according to the values set for the SERVICE\_NAMES initialization parameter.

Services can also be maintained by using SRVCTL and Enterprise Manager.

In environments using Oracle Clusterware, Oracle Restart, or Oracle GlobalData Services, you must use SRVCTL to create database services.

**Note:** Oracle Restart is deprecated in Oracle Database 12c.

# Managing Services in a Single-Instance Environment

- Create a new service.

```
exec DBMS_SERVICE.CREATE_SERVICE('SERV1','SERV1.oracle.com');
```

- Start a service.

```
exec DBMS_SERVICE.START_SERVICE('SERV1');
```

- Stop a service.

```
exec DBMS_SERVICE.STOP_SERVICE('SERV1');
```

- Delete a service.

```
exec DBMS_SERVICE.DELETE_SERVICE('SERV1');
```

- Disconnect sessions connected under a service.

```
exec DBMS_SERVICE.DISCONNECT_SESSION('SERV1');
```



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

Use the following procedures in the DBMS\_SERVICE package to manage services:

- CREATE\_SERVICE: This procedure creates a service name in the data dictionary. Services are also created in the data dictionary implicitly when you set the service in the SERVICE\_NAMES initialization parameter or by means of the ALTER SYSTEM SET SERVICE\_NAMES command. When you create a service, you must specify its name, and its network's name. The network name should then be used in the SERVICE\_NAME parameter of your corresponding connect descriptor. Additional parameters may be specified when creating a service.
- START\_SERVICE: This procedure starts a service in a single-instance environment.
- STOP\_SERVICE: This procedure stops a service. In a single-instance environment, this procedure alters the SERVICE\_NAMES to remove this service name.
- DELETE\_SERVICE: This procedure deletes a service from the data dictionary. You must stop a service before you can delete it.
- DISCONNECT\_SESSION: This procedure disconnects sessions with the named service at the current instance. This subprogram does not return until all corresponding sessions are disconnected.

**Note:** For more information about the DBMS\_SERVICE package, refer to the *PL/SQL Packages and Types Reference* guide.

## Where Are Services Used?

- Data dictionary maintains services.
- AWR measures the performance of services.
- The Database Resource Manager can use services in place of users for priorities.
- A service can be specified for Scheduler jobs, parallel execution (PQ, PDML, and PDDL), and Streams queues.



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

Several database features support services. A session is tracked by the service with which it connects. In addition, performance-related statistics and wait events are also tracked by services.

Automatic Workload Repository (AWR) manages the performance of services. It records the service performance, including SQL execution times, wait classes, and resources consumed by service. AWR alerts the DBA when service response time thresholds are exceeded. Specific dynamic performance views report current service status with one hour of history.

The Database Resource Manager is capable of managing services for prioritizing application workloads within an instance, using Resource Plans and Consumer Group Mapping. Job Classes has a service option in the Scheduler to control which service to which a job connects as opposed to a specific instance. Parallel slave processes inherit the service of their coordinator.

# Using Services with Client Applications

```
ERP= (DESCRIPTION=
      (LOAD_BALANCE=on)
      (ADDRESS= (PROTOCOL=TCP) (HOST=node-1vip) (PORT=1521))
      (ADDRESS= (PROTOCOL=TCP) (HOST=node-2vip) (PORT=1521))
      (ADDRESS= (PROTOCOL=TCP) (HOST=node-3vip) (PORT=1521))
      (ADDRESS= (PROTOCOL=TCP) (HOST=node-4vip) (PORT=1521))
      (CONNECT_DATA= (SERVICE_NAME=ERP)))
```

```
url="jdbc:oracle:oci:@ERP"
```

```
url="jdbc:oracle:thin:@(DESCRIPTION=
      (LOAD_BALANCE=on)
      (ADDRESS= (PROTOCOL=TCP) (HOST=node-1vip) (PORT=1521))
      (ADDRESS= (PROTOCOL=TCP) (HOST=node-2vip) (PORT=1521))
      (ADDRESS= (PROTOCOL=TCP) (HOST=node-3vip) (PORT=1521))
      (ADDRESS= (PROTOCOL=TCP) (HOST=node-4vip) (PORT=1521))
      (CONNECT_DATA= (SERVICE_NAME=ERP)))"
```



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

Services have their greatest usefulness when multiple servers offer the same service as in a RAC environment. The examples use a RAC database with four nodes.

Applications and mid-tier connection pools select a service by using the TNS connection descriptor.

The selected service must match the service that has been created.

The address lists in each example in the slide use virtual IP addresses. Using the virtual IP addresses for client communication ensures that connections and SQL statements issued against a node that is down do not result in a TCP/IP timeout.

The first example in the slide shows the TNS connect descriptor that can be used to access the ERP service.

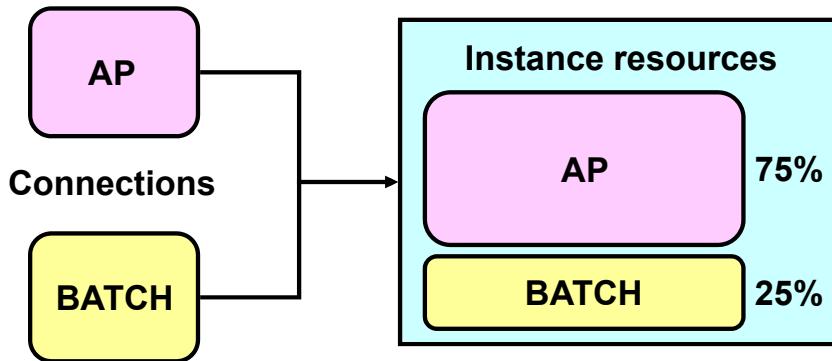
The second example shows the thick JDBC URL using the previously defined TNS connect descriptor.

The third example shows the thin JDBC URL using the same TNS connect descriptor.

**Note:** The `LOAD_BALANCE=ON` clause is used by Oracle Net to randomize its progress through the protocol addresses of the connect descriptor. This feature is called client connection load balancing.

# Using Services with the Resource Manager

- Consumer groups are automatically assigned to sessions based on session services.
- Work is prioritized by service inside one instance.



ORACLE®

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

The Database Resource Manager (also called the Resource Manager) enables you to identify work by using services. It manages the relative priority of services within an instance by binding services directly to consumer groups. When a client connects by using a service, the consumer group is assigned transparently at connect time. This enables the Resource Manager to manage the work requests by service in the order of their importance.

For example, you define the AP and BATCH services to run on the same instance, and assign AP to a high-priority consumer group and BATCH to a low-priority consumer group. Sessions that connect to the database with the AP service specified in their TNS connect descriptor get priority over those that connect to the BATCH service.

This offers benefits in managing workloads because priority is given to business functions rather than the sessions that support those business functions.

# Using Enterprise Manager to Manage Consumer Group Mappings

The screenshot shows the 'Consumer Group Mappings' page in Oracle Enterprise Manager. The 'General' tab is active. On the left, there's a 'View Service' dropdown set to 'Service', a 'Value' input field containing 'No Mappings Specified', and a prominent red-bordered 'Add Rule for Selected Type' button. On the right, the breadcrumb shows 'Resource Consumer Group Mappings > Resource Consumer Group Mappings Consumer Group Mappings'. Below it, a message says 'Use a mapping rule to automatically assign sessions to specific consumer groups by Selected Consumer Group' with a dropdown set to 'BATCH\_GROUP'. The main area has two columns: 'Available Service' (containing ORCL, ORCL.ORACLE.COM, ORCLXDB, SYS\$BACKGROUND, SYS\$USERS) and 'Selected Service' (empty). Between them are four buttons: 'Move', 'Move All', 'Remove', and 'Remove All'.

**ORACLE**

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

Enterprise Manager Cloud Control includes an interface through the Resource Consumer Group Mapping page to automatically map sessions to consumer groups. You can access this page by expanding the Administration menu, selecting Resource Manager, and then selecting Consumer Group Mappings.

Using the General tabbed page of the Resource Consumer Group Mappings page, you can set up a mapping of sessions connecting with a service name to consumer groups. Also on this page, there is an option for a module name and action mapping.

With the ability to map sessions to consumer groups by service, module, and action, you have greater flexibility when it comes to managing the performance of different application workloads.

**Note:** Using the Priorities tabbed page of the Resource Consumer Group Mappings page, you can set priorities for the mappings that you set up on the General tabbed page. The mapping options correspond to columns in V\$SESSION. When multiple mapping columns have values, the priorities you set determine the precedence for assigning sessions to consumer groups.

## Services and the Resource Manager: Example

```

exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA;
exec DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(
    CONSUMER_GROUP => 'HIGH_PRIORITY',
    COMMENT => 'High priority consumer group');
exec DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING(
    ATTRIBUTE => DBMS_RESOURCE_MANAGER.SERVICE_NAME,
    VALUE => 'AP',
    CONSUMER_GROUP => 'HIGH_PRIORITY');
exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA;

```

```

exec -
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP(
    GRANTEE_NAME => 'PUBLIC',
    CONSUMER_GROUP => 'HIGH_PRIORITY',
    GRANT_OPTION => FALSE);

```



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

Assume that your site has two consumer groups called HIGH\_PRIORITY and LOW\_PRIORITY. These consumer groups map to a resource plan for the database that reflects either the intended ratios or the intended resource consumption.

Before mapping services to consumer groups, you must first create the consumer groups and the resource plan for these consumer groups. The resource plan can be priority based or ratio based. The PL/SQL calls shown in the slide are used to create the HIGH\_PRIORITY consumer group, and map the AP service to the HIGH\_PRIORITY consumer group. You can use similar calls to create the LOW\_PRIORITY consumer groups and map the BATCH service to the LOW\_PRIORITY consumer group.

The last PL/SQL call in the example in the slide is executed because sessions are automatically assigned only to consumer groups for which they have been granted switch privileges. A similar call should be executed for the LOW\_PRIORITY consumer group.

**Note:** For more information about the Database Resource Manager, refer to the *Oracle Database Administrator's Guide* and *PL/SQL Packages and Types Reference*.

# Using Enterprise Manager to Create a Job Class

Scheduler Job Classes > Create Job Class  
Create Job Class

\* Name SERVER\_BATCH

Description

Logging Level Log job runs only (RUNS)

Log Retention Period (Days)

Resource Consumer Group

Service Name SERV1

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

The Scheduler uses services. When you create a job class, you define the service that the job class uses. You assign jobs to job classes and job classes run within services. Using services with job classes ensures that the work of the Scheduler is identified for workload management and performance tuning.

For example, jobs inherit server-generated alerts and performance thresholds for the service they run under.

To configure a job to run under a specific service, expand the Administration menu, expand the Oracle Scheduler menu, and select Job Classes. On the Scheduler Job Classes page, you can see services assigned to job classes.

When you click the Create button on the Scheduler Job Classes page, the Create Job Class page is displayed. On this page, you can enter details of a new job class, including which service it must run under.

# Using Enterprise Manager to Create a Job

Scheduler Jobs > Create Job  
**Create Job**

**General**   **Schedule**   **Options**

\* Name  \* Schema

Enabled  Yes  No

Description

Logging Level   Specify logging requirements for the job

Job Class

Auto Drop  Specify whether the job should be dropped after completion

Restartable  Specify whether the job can be restarted manually or in the event of failure

Destination Destination and Credential Name only apply for jobs of type executable. For Destination specify the host:port of the Name specify the credential to use to run the external job. Credential Name

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

After your job class is set up with the service that you want it to run under, you can create the job.

To create the job, expand the Administration menu, expand the Oracle Scheduler menu, and select Jobs. On the Scheduler Jobs page, click the Create button to create a new job. When you click the Create button, the Create Job page is displayed. This page has different tabs: General, Schedule, and Options. Use the General tabbed page to assign your job to a job class.

## Services and the Scheduler: Example

```
DBMS_SCHEDULER.CREATE_JOB_CLASS(
    JOB_CLASS_NAME          => 'HOT_BATCH_CLASS',
    RESOURCE_CONSUMER_GROUP => NULL,
    SERVICE                 => 'HOT_BATCH_SERV',
    LOGGING_LEVEL            => DBMS_SCHEDULER.LOGGING_RUNS,
    LOG_HISTORY              => 30, COMMENTS => 'P1 batch');
```

```
DBMS_SCHEDULER.CREATE_JOB (
    JOB_NAME    => 'my_report_job',
    JOB_TYPE    => 'stored_procedure',
    JOB_ACTION   => 'my_name.my_proc();',
    NUMBER_OF_ARGUMENTS => 4, START_DATE => SYSDATE+1,
    REPEAT_INTERVAL => 5, END_DATE => SYSDATE+30,
    JOB_CLASS    => 'HOT_BATCH_CLASS', ENABLED => TRUE,
    AUTO_DROP    => false, COMMENTS => 'daily status');
```



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

In this PL/SQL example, you define a batch queue, HOT\_BATCH\_CLASS, managed by the Scheduler. You associate the HOT\_BATCH\_SERV service to the HOT\_BATCH\_CLASS queue. It is assumed that you had already defined the HOT\_BATCH\_SERV service.

After the class is defined, you can define your job. In this example, the MY\_REPORT\_JOB job executes in the HOT\_BATCH\_CLASS job class at instances offering the HOT\_BATCH\_SERV service.

In this example, you do not assign a resource consumer group to the HOT\_BATCH\_CLASS job class. However, it is possible to assign a consumer group to a class. Regarding services, this allows you to combine Scheduler jobs and service prioritization by using the Database Resource Manager.

**Note:** For more information about the Scheduler, refer to the *Oracle Database Administrator's Guide* and *PL/SQL Packages and Types Reference*.

# Using Services with Metric Thresholds

- You can define service-level thresholds:
  - ELAPSED\_TIME\_PER\_CALL
  - CPU\_TIME\_PER\_CALL
- Server-generated alerts are triggered on threshold violations.
- You can react on generated alerts:
  - Change priority
  - Relocate services
  - Add instances for services

```
SELECT service_name, elapsedpercall, cpupercall  
FROM v$servicemetric;
```



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

Service-level thresholds permit the comparison of actual service levels against the accepted minimum required level. This provides accountability with respect to delivery or failure to deliver an agreed service level. You can explicitly specify two metric thresholds for each service on a particular instance:

- **The response time for calls:** ELAPSED\_TIME\_PER\_CALL. The alert on this threshold is activated when the elapsed time (wall-clock time) exceeds the threshold value. This is a fundamental measure that reflects all delays and faults the call experiences.
- **CPU time for calls:** CPU\_TIME\_PER\_CALL

AWR monitors the service time and publishes AWR alerts when the performance exceeds the thresholds. You can respond to these alerts by changing the priority of a job, stopping overloaded processes, or relocating, expanding, shrinking, starting, or stopping a service. You can use automated tasks to respond to these alerts. These alerts enable you to maintain service quality despite changes in demand.

**Note:** The SELECT statement shown in the slide gives you the accumulated instance statistics for elapsed time and for CPU-used metrics for each service for the most recent 60-second interval. For the last hour history, look at V\$SERVICEMETRIC\_HISTORY.

# Using Enterprise Manager to Change Service Thresholds

Database Instance: orcl > Metric and Collection Settings > Edit Advanced Settings: Service Response Time (per user call) (microseconds)

**Edit Advanced Settings: Service Response Time (per user call) (microseconds)**

**Monitored Objects**

The table lists all Service Name objects monitored for this metric. You can specify different threshold settings for each Service Name object.

Select	Service Name	Comparison Operator	Warning Threshold	Critical Threshold	Corrective Action
<input checked="" type="radio"/>	SERV1	>	40000000	100000000	None
<input type="radio"/>	All others	>			None

**TIP** Empty Thresholds will disable alerts for that metric.  
**TIP** Wildcards for object names are not supported as this is a Server Managed Metric. (Example: Key 'A%' is not allowed, but you can use 'A\' exact phrase 'A%').

ORACLE

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

The Metric and Collection Settings page is displayed in the slide. The screenshot shows a portion of the page where you can see the Service CPU Time (per user call) and Service Response Time (per user call) metrics.

To access the Metric and Collection Settings page, expand the Oracle Database menu. Then expand the Monitoring menu and select Metric and Collection Settings. Using the Metric and Policy Settings page, you can change the critical and warning values for the service metrics. If you modify the critical and warning values on this page, the thresholds apply to all services of the instance.

If you want different thresholds for different services, click the Edit button on the right. Another page appears, where you can set critical and warning thresholds for individual services.

## Services and Metric Thresholds: Example

```
exec DBMS_SERVER_ALERT.SET_THRESHOLD(-
  METRICS_ID => dbms_server_alert.elapsed_time_per_call,
  WARNING_OPERATOR => dbms_server_alert.operator_ge,
  WARNING_VALUE => '500000',
  CRITICAL_OPERATOR => dbms_server_alert.operator_ge,
  CRITICAL_VALUE => '750000',
  OBSERVATION_PERIOD => 15,
  CONSECUTIVE_OCCURRENCES => 3,
  INSTANCE_NAME => 'IOn',
  OBJECT_TYPE => dbms_server_alert.object_type_service,
  OBJECT_NAME => 'ERP');
```



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

In this example, thresholds are added for the ERP service for the ELAPSED\_TIME\_PER\_CALL metric. This metric measures the elapsed time for each user call for the corresponding service. The time must be expressed in microseconds.

A warning alert is raised by the server whenever the average elapsed time per call for the ERP service over a 15-minute period exceeds 0.5 seconds three consecutive times.

A critical alert is raised by the server whenever the average elapsed time per call for the ERP service over a 15-minute period exceeds 0.75 seconds three consecutive times.

# Service Aggregation and Tracing

- Statistics are always aggregated by service to measure workloads for performance tuning.
- Statistics can be aggregated at finer levels:
  - MODULE
  - ACTION
  - Combination of SERVICE\_NAME, MODULE, ACTION
- Tracing can be done at various levels:
  - SERVICE\_NAME
  - MODULE
  - ACTION
  - Combination of SERVICE\_NAME, MODULE, ACTION
- Aggregation and tracing are useful for tuning systems using shared sessions.



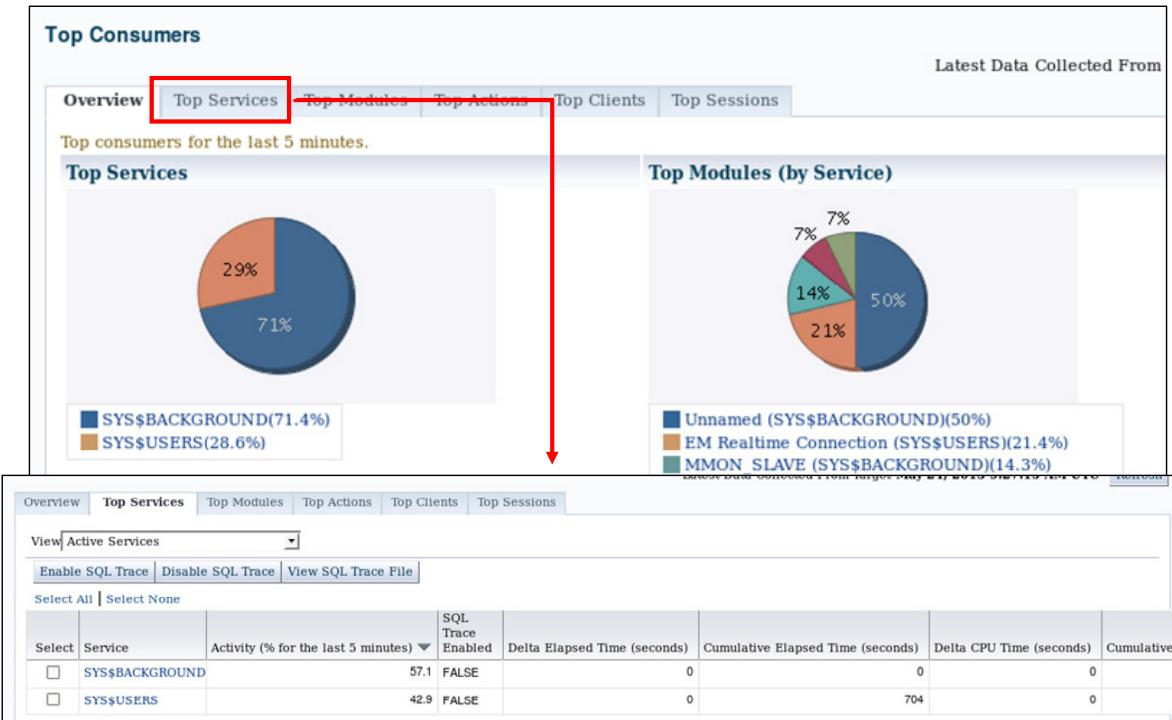
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Important statistics and wait events are collected for the work attributed to every service by default. An application can qualify a service by MODULE and ACTION names to identify the important transactions within the service. This enables you to locate exactly the poorly performing transactions. In systems where the sessions are shared, accountability is difficult. For example, in systems where connection pools or transaction processing monitors are used, the sessions are shared.

SERVICE\_NAME, MODULE, and ACTION are columns in V\$SESSION. SERVICE\_NAME is set automatically at login time for the user. MODULE and ACTION names are set by the application by using the DBMS\_APPLICATION\_INFO PL/SQL package or special OCI calls. MODULE should be set to a user-recognizable name for the program that is currently executing. Likewise, ACTION should be set to a specific action or task that a user is performing within a module (for example, entering a new customer).

Workload aggregation also enables tracing by service. The traditional method of tracing each session produces trace files with SQL commands that can span workloads. This results in a hit-or-miss approach to diagnose problematic SQL. You can produce a single output trace file that contains SQL that is relevant to a specific workload by using the SERVICE\_NAME, MODULE, or ACTION criteria.

# Top Services Performance Page



**ORACLE**

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

From the Performance Home page, you can access the Top Consumers page by clicking the Top Consumers link.

The Top Consumers page has several tabs for displaying your database as a single-system image. The Overview tabbed page contains four pie charts: Top Clients, Top Services, Top Modules, and Top Actions. Each chart provides a different perspective regarding the top resource consumers in your database.

The Top Services tabbed page displays performance-related information for the services that are defined in your database. Using this page, you can enable or disable tracing at the service level, as well as view the resulting SQL trace file.

# Service Aggregation Configuration

- Automatic service aggregation level of statistics
- DBMS\_MONITOR used for finer granularity of service aggregations:
  - SERV\_MOD\_ACT\_STAT\_ENABLE
  - SERV\_MOD\_ACT\_STAT\_DISABLE
- Possible additional aggregation levels:
  - SERVICE\_NAME/MODULE
  - SERVICE\_NAME/MODULE/ACTION
- Tracing services, modules, and actions:
  - SERV\_MOD\_ACT\_TRACE\_ENABLE
  - SERV\_MOD\_ACT\_TRACE\_DISABLE
- Database settings persist across instance restarts.



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

On each instance, important statistics and wait events are automatically aggregated and collected by service. This level of aggregation is automatic if SERVICE\_NAME is specified in the connect string. Each connect string can be associated with a different service. You can achieve a finer level of granularity of statistics collection for services by using the SERV\_MOD\_ACT\_STAT\_ENABLE procedure in the DBMS\_MONITOR package. This procedure enables statistics gathering for additional hierarchical combinations of SERVICE\_NAME/MODULE and SERVICE\_NAME/MODULE/ACTION. The SERV\_MOD\_ACT\_STAT\_DISABLE procedure stops the statistics gathering that was turned on. The enabling and disabling of statistics aggregation within the service applies to every instance accessing the database. Furthermore, these settings are persistent across instance restarts.

The SERV\_MOD\_ACT\_TRACE\_ENABLE procedure enables tracing for services with three hierarchical possibilities: SERVICE\_NAME, SERVICE\_NAME/MODULE, and SERVICE\_NAME/MODULE/ACTION. The default is to trace for all instances that access the database. A parameter is provided that restricts tracing to specified instances where poor performance is known to exist. This procedure also gives you the option of capturing relevant waits and bind variable values in the generated trace files.

SERV\_MOD\_ACT\_TRACE\_DISABLE disables the tracing at all enabled instances for a given combination of service, module, and action. Like the statistics gathering mentioned previously, service tracing persists across instance restarts.

## Service Aggregation: Example

- Collect statistics on service and module:

```
exec DBMS_MONITOR.SERV_MOD_ACT_STAT_ENABLE(-
    'AP', 'PAYMENTS');
```

- Collect statistics on service, module, and action:

```
exec DBMS_MONITOR.SERV_MOD_ACT_STAT_ENABLE(-
    'AP', 'PAYMENTS', 'QUERY_DELINQUENT');
```

- Trace all sessions of an entire service:

```
exec DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE('AP');
```

- Trace on service, module, and action:

```
exec DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE(-
    'AP', 'PAYMENTS', 'QUERY_DELINQUENT');
```



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

The first piece of sample code begins collecting statistics for the PAYMENTS module within the AP service. The second example collects statistics only for the QUERY\_DELINQUENT program that runs in the PAYMENTS module under the AP service. This enables statistics collection on specific tasks that run in the database.

In the third code box, all sessions that log in under the AP service are traced. A trace file is created for each session that uses the service, regardless of the module and action.

You can also trace specific tasks within a service. This is illustrated in the last example, where all sessions of the AP service that execute the QUERY\_DELINQUENT action within the PAYMENTS module are traced.

Tracing by service, module, and action enables you to focus your tuning efforts on specific SQL, rather than sifting through trace files with SQL from different programs. Only the SQL statements that define this task are recorded in the trace file. This complements collecting statistics by service, module, and action because relevant wait events for an action can be identified.

**Note:** For more information about the DBMS\_MONITOR package, refer to the *PL/SQL Packages and Types Reference*.

# Client Identifier Aggregation and Tracing

- Collect statistics on client identifier:

```
exec DBMS_MONITOR.CLIENT_ID_STAT_ENABLE('HR.HR');
```

- View collected data:

```
SELECT * FROM V$CLIENT_STATS;
```

- Disable statistics collection:

```
exec DBMS_MONITOR.CLIENT_ID_STAT_DISABLE('HR.HR');
```

- Trace client identifiers:

```
exec DBMS_MONITOR.CLIENT_ID_TRACE_ENABLE(
  client_id => 'HR.HR', waits => TRUE, binds => FALSE);
```



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

It is also possible to use the DBMS\_MONITOR package to enable and disable statistics aggregation for a particular client identifier. The client identifier is set using the DBMS\_SESSION.SET\_IDENTIFIER procedure, and is visible through the CLIENT\_IDENTIFIER column from V\$SESSION.

V\$CLIENT\_STATS displays the resulting measures for all sessions that are active for the client identifier per instance. Similar to the aggregated statistics available for service aggregation, the statistics published in V\$CLIENT\_STATS are a subset of those available in V\$SESSTAT and V\$SESS\_TIME\_MODEL.

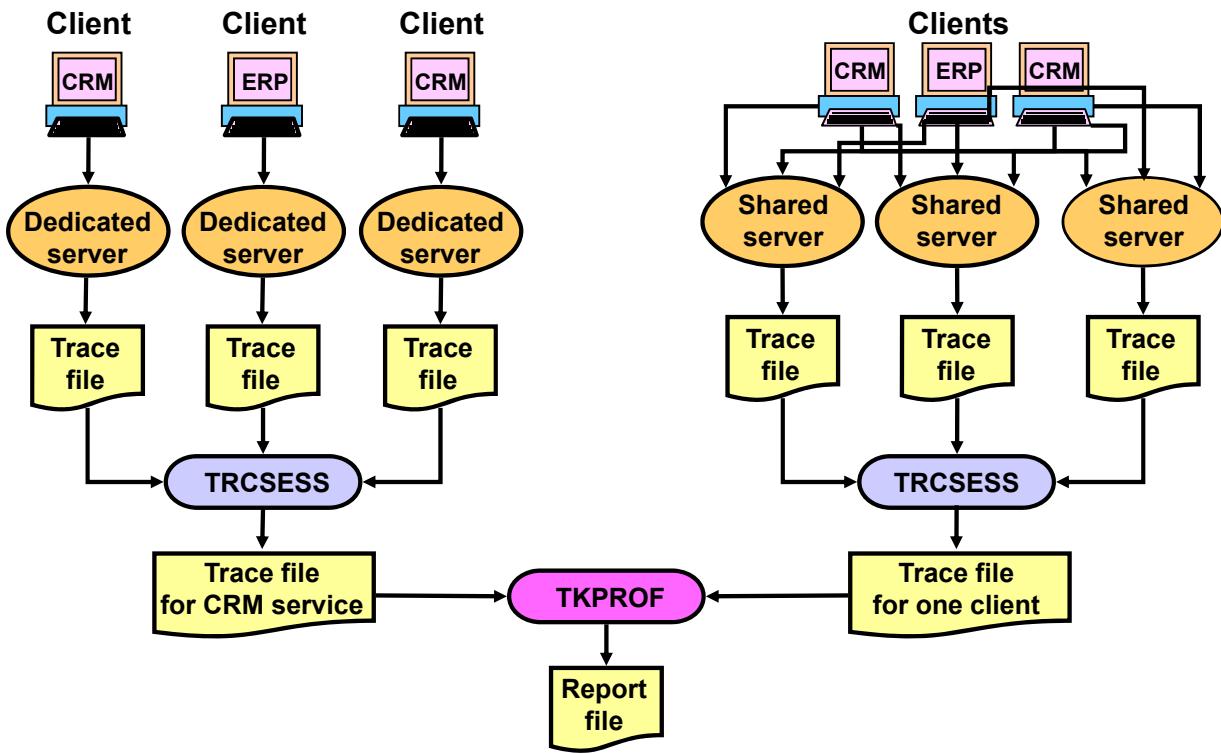
Using the CLIENT\_ID\_STAT\_DISABLE procedure, you can disable accumulation of wait model statistics for the specified client identifier.

The CLIENT\_ID\_TRACE\_ENABLE procedure enables tracing globally for the database for a given client identifier. As you can see from the example, you can also request to dump wait events and bind variables to the trace files.

Use the CLIENT\_ID\_TRACE\_DISABLE procedure to disable the generation of the trace files for a specified client identifier.

**Note:** You can use the Top Clients page accessible from the Top Consumers page of Enterprise Manager to graphically do the same thing as with the PL/SQL interface.

## trcsess Utility



ORACLE

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

The `trcsess` utility consolidates trace output from selected trace files based on several criteria: session ID, client ID, service name, action name, and module name. After `trcsess` merges the trace information into a single output file, the output file can be processed by `tkprof`. When using the `DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE` procedure, tracing information is present in multiple trace files and you must use the `trcsess` tool to collect it into a single file. The `trcsess` utility is useful for consolidating the tracing of a particular session or service for performance or debugging purposes.

Tracing a specific session is usually not a problem in the dedicated server model because a single dedicated process serves a session during its lifetime. All the trace information for the session can be seen from the trace file belonging to the dedicated server serving it. However, tracing a service might become a complex task even in the dedicated server model.

Moreover, in a shared-server configuration, a user session is serviced by different processes from time to time. The trace pertaining to the user session is scattered across different trace files belonging to different processes. This makes it difficult to get a complete picture of the life cycle of a session.

# Service Performance Views

- Service, module, and action information in:
  - V\$SESSION
  - V\$ACTIVE\_SESSION\_HISTORY
- Service performance in:
  - V\$SERVICE\_STATS
  - V\$SERVICE\_EVENT
  - V\$SERVICE\_WAIT\_CLASS
  - V\$SERVICEMETRIC
  - V\$SERVICEMETRIC\_HISTORY
  - V\$SERV\_MOD\_ACT\_STATS
  - DBA\_ENABLED\_AGGREGATIONS
  - DBA\_ENABLED\_TRACES
- Twenty-eight statistics for services



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

The service, module, and action information are visible in V\$SESSION and V\$ACTIVE\_SESSION\_HISTORY.

The call times and performance statistics are visible in V\$SERVICE\_STATS, V\$SERVICE\_EVENT, V\$SERVICE\_WAIT\_CLASS, V\$SERVICEMETRIC, and V\$SERVICEMETRIC\_HISTORY. These views show the statistics and metrics gathered at the service level. Examples are:

```
SQL> SELECT service_name, stat_name, value
  2  FROM V$SERVICE_STATS
  3 WHERE service_name = 'SERV1';
```

SERVICE_NAME	STAT_NAME	VALUE
SERV1	user calls	37
SERV1	DB time	225407870
SERV1	DB CPU	216824843
SERV1	parse count (total)	134
SERV1	parse time elapsed	975732
...		

```
SQL> SELECT service_name, elapsedpercall, cpupercall,
  2      dbtimepercall, dbtimepersec
  3  FROM V$SERVICEMETRIC
 4* WHERE service_name = 'SERV1';
```

SERVICE_NAME	ELAPSEDPERCALL	CPUPERCALL	DBTIMEPERCALL	DBTIMEPERSEC
SERV1	0	0	0	0
SERV1	58593980	55399177	58593980	97.6078294

When statistics collection for specific modules and actions is enabled, performance measures are visible at each instance in V\$SERV\_MOD\_ACT\_STATS.

Of the over 300 performance-related statistics that are tracked and visible in V\$SYSSTAT, 28 statistics are tracked for services. To see the statistics measured for services, run the following query:

```
SELECT DISTINCT stat_name FROM V$SERVICE_STATS
```

Of the 28 statistics, DB time and DB CPU are worth mentioning. DB time is a statistic that measures the average response time per call. It represents the actual wall-clock time for a call to complete. DB CPU is an average of the actual CPU time spent per call. The difference between response time and CPU time is the wait time for the service. After the wait time is known, and if it consumes a large percentage of response time, then you can trace at the action level to identify the waits.

**Note:** DBA\_ENABLED\_AGGREGATIONS displays information about enabled on-demand statistic aggregation. DBA\_ENABLED\_TRACES displays information about enabled traces.

# Quiz

A service (choose all that apply):

- a. Is a representation of an instance
- b. Is a grouping of sessions doing similar work
- c. Replaces tnsnames to connect clients to instances
- d. Is a way to collect performance statistics for an application
- e. Is an abstraction of a data set independent of the instance providing it



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

**Answer: b, d, e**

## Summary

In this lesson, you should have learned how to:

- Configure and manage services
- Use services with client applications
- Use services with the Database Resource Manager
- Use services with the Scheduler
- Set performance-metric thresholds on services
- Configure services aggregation and tracing



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

## **Practice 9 Overview: Using Services**

This practice covers the following topics:

- Using services in a single-instance environment
- Tracing services in a single-instance environment



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



# 10

## Identifying Problem SQL Statements

ORACLE®

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

# Objectives

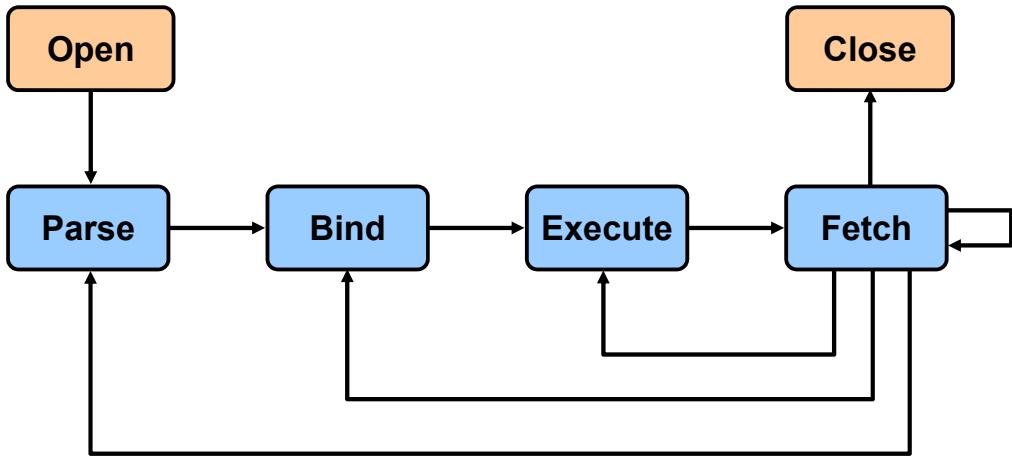
After completing this lesson, you should be able to do the following:

- Describe SQL statement processing
- Describe the role of the optimizer
- View the SQL statement statistics
- Identify the SQL statements that perform poorly
- Generate and view an execution plan
- Generate a TKPROF report
- Generate an optimizer trace



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

# SQL Statement Processing Phases



ORACLE®

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

A good understanding of SQL processing is helpful for understanding the SQL statistics. In SQL statement processing, there are four important phases: parsing, binding, executing, and fetching.

The reverse arrows indicate processing scenarios (for example, fetch—(re)bind—execute—fetch).

The fetch phase applies only to queries and DML statements with a returning clause.

**Note:** A detailed description of SQL statement processing can be found in the *Oracle Database Development Guide* and *Oracle Database Concepts*.

# Understanding Parsing

- Parse phase checks:
  - Syntax
  - Semantics and privileges
- Types of parses:
  - Soft parse:
    - Searches for the statement in the shared pool
  - Hard parse:
    - Merges view definitions and subqueries
    - Determines execution plan



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

Parsing is one of the stages in the processing of a SQL statement. When an application issues a SQL statement, the application makes a parse call to the Oracle Database server. During the parse call, the Oracle Database server:

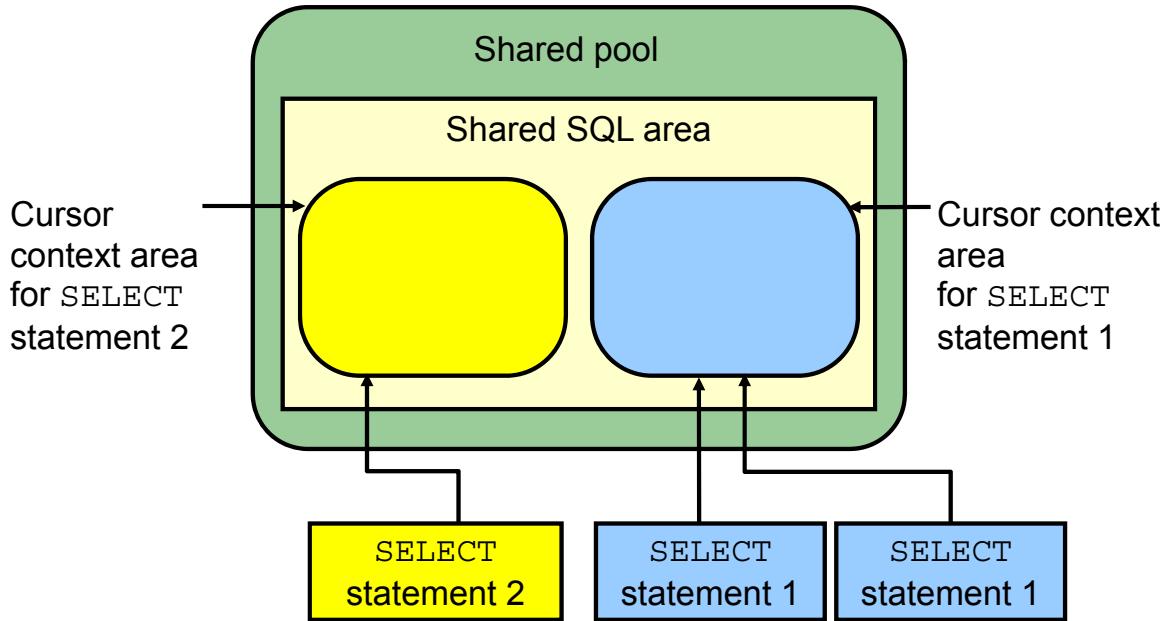
- Checks the statement for syntactic and semantic validity
- Determines whether the process issuing the statement has privileges to run it
- Searches for a shareable match of the statement in the shared pool
- Allocates a private SQL area for the statement

There are two types of parse operations:

- **Soft parsing:** A SQL statement is submitted, and a match *is* found in the shared pool. The match can be the result of a previous execution by another user. The SQL statement is shared, which is good for performance. However, soft parses still require syntax and security checking, which consume system resources.
- **Hard parsing:** A SQL statement is submitted for the first time, and no shareable match is found in the shared pool. Hard parses are the most resource-intensive and unscalable, because they perform all the operations involved in a parse.

When bind variables are used properly, more soft parses are possible, thereby reducing hard parses and keeping parsed statements in the library cache for a longer period.

## SQL Cursor Storage



**ORACLE**

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

The Oracle server uses the library cache and the shared SQL area to store SQL statements and PL/SQL blocks. When a statement is stored in the cache, the Oracle server:

- Reduces the statement to the numeric value of the ASCII text
- Uses a hash function of this number
- Places the cursor for this statement on a hash chain

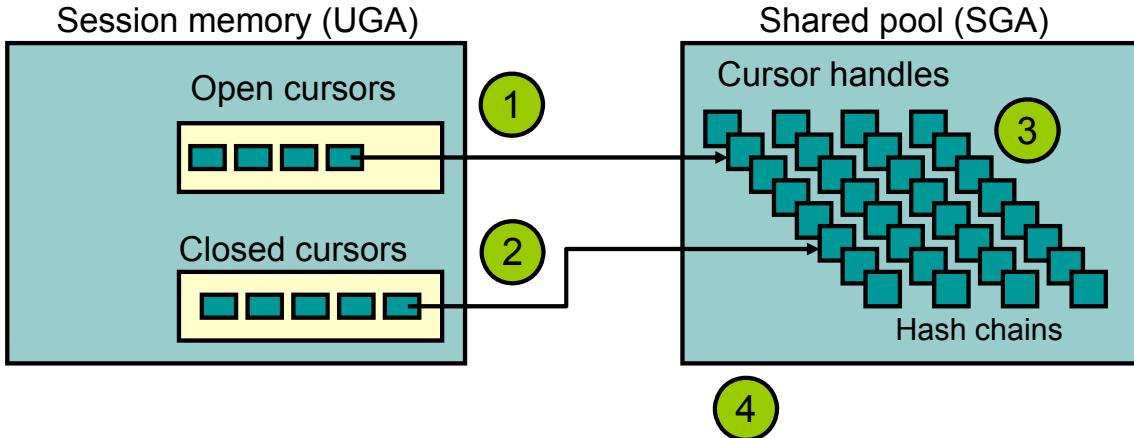
The hash value is not a unique value, and several statements may hash to the same value. The cursor contexts for these statements are all stored in the same hash chain. The hash chain is searched for the correct statement. Any time that a statement is submitted, the cache is searched. If the cursor handle is not found, the cursor is constructed from the statement.

If the statement has already been parsed and executed, and the cursor handle is still in the client cache, the cursor may be called and executed without searching the shared pool for the statement. The parse count statistic is still incremented whenever the parse request is made, but finding the statement in the session cache has a much lower overhead.

This is the basic behavior and it is modified by the `CURSOR_SHARING` parameter and the adaptive cursor sharing feature described in the lesson titled "Tuning the Shared Pool."

**Note:** Ideally the SQL statement gets one hard parse the first time that it is submitted, and one soft parse for each additional session that uses the statement. This depends on sufficient memory both in the session cache, and the shared pool to retain the cursor information.

# Cursor Usage and Parsing



Parse procedure:

1. Find and execute an open cursor.
2. Find a closed cursor in the session cache.
3. Search the hash chains (soft parse).
4. Construct the cursor (hard parse).

**ORACLE**

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

Every developer wants his or her code to run as fast as possible. For code with SQL statements, this means that cursor access must be fast. The fastest possible access of a cursor is through the open cursor cache in the session memory of the server session. Every open cursor in the open cursor cache has a pointer to the SGA memory location of that cursor handle. To execute the cursor, the pointer is used; parsing is not required. An open cursor has already been parsed and the cursor handle is in the library cache.

When a cursor is closed, the cursor information is moved into the session's closed cursor cache, if the `SESSION_CACHED_CURSORS` parameter has been set to some value. (The default is 50.)

When a cursor is opened, the session hashes the SQL statement and performs a hash lookup in the closed cursor cache. If the cursor is found, it is moved to the open cursor cache, then the pointer to the cursor handle in the shared pool is used to execute the cursor. No parse is required.

If the cursor is not found in the session, then the hash value is used to search the hash chains in the shared pool for the cursor handle. The search is registered as a parse. If the cursor handle is found and the rest of the cursor has not aged out, the cursor is executed. This is a soft parse.

If the cursor has aged out of the shared pool, or the cursor does not exist in the shared pool, then the cursor is constructed. This is a hard parse. The cursor construction requires lookups of the metadata for dependent objects such as tables, indexes, extents, and sequences. If the metadata for these objects is not already cached in the shared pool, then recursive SQL is generated to fetch the information from the data dictionary.

In some cases where a large number of cursors is submitted to the shared pool and the shared pool has limited memory allocated, it is possible for a cursor to age out of the shared pool very quickly, even between fetches. This situation will lead to a large number of hard parses.

**Note:** Details for tuning the shared pool to optimize cursor handling are presented in the *Tuning the Shared Pool* lesson

# SQL Statement Processing Phases: Bind

- Bind phase:
  - Checks the statement for bind variables
  - Assigns or reassigns a value to the bind variable
- Bind variables impact performance when:
  - Parsing is reduced by using a shared cursor
  - A different execution plan might benefit performance with different bind values



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

During the bind phase, the Oracle Database server:

- Checks the statement for references to bind variables
- Assigns or reassigns a value to each variable

When bind variables are used in a statement, the optimizer assumes that cursor sharing is intended and that different invocations should use the same execution plan. This helps performance by reducing hard parses.

When a histogram is present, the optimizer assumes that the data distribution does not match the default assumptions of the optimizer. Therefore multiple invocations of the cursor with different bind variables might significantly benefit from different execution plans. In this case adaptive cursor sharing will create new plans. If new plans are not attempted, performance could degrade for certain bind variable values.

Cursor sharing is affected by database initialization parameters and the Adaptive Cursor Sharing feature of Oracle Database. For more details, see the lesson titled “Tuning the Shared Pool.”

If the bind variable does not match the type of the column, an implicit conversion will take place possibly preventing the optimizer in choosing fast access indexes. Use the DBA\_HIST\_SQLBIND view to find the actual types used.

# SQL Statement Processing Phases: Execute and Fetch

- Execute phase:
  - Executes the SQL statement
  - Performs necessary I/O and sorts for data manipulation language (DML) statements
- Fetch phase:
  - Retrieves rows for a query
  - Sorts for queries when needed
  - Uses an array fetch mechanism



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

## Execute Phase

The execution plan is a series of steps that the server process uses to access and to identify the required rows of data from the data buffers. Multiple users can share the same execution plan. The Oracle Database performs physical reads or logical reads/writes for DML statements and also sorts the data when needed.

**Note:** Physical reads are disk reads; logical reads are blocks already in memory in the database buffer cache. Physical reads use more resources and time because they require I/O from disk.

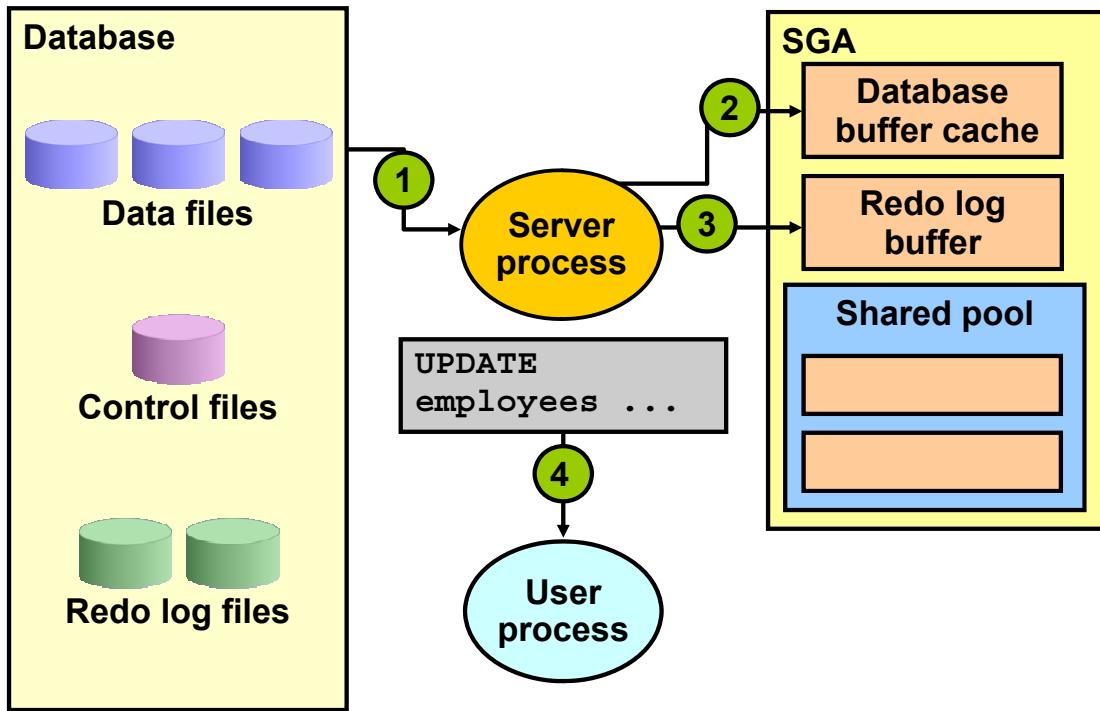
## Fetch Phase

The Oracle Database server retrieves rows for a SELECT statement during the fetch phase. Each fetch typically retrieves multiple rows, using an array fetch. Array fetches can improve performance by reducing network round trips. Each Oracle tool offers its own ways of influencing the array size—for example, in SQL\*Plus, you can change the fetch size by using the `ARRAYSIZE` setting:

```
SQL> show arraysizes
arraysize 15
SQL> set arraysizes 50
```

SQL\*Plus processes 15 rows at a time by default. Very high array sizes provide little or no advantage.

# Processing a DML Statement



ORACLE

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

## DML Processing Steps

A data manipulation language (DML) statement requires only two phases of processing:

- Parse is the same as the parse phase used for processing a query.
- Execute requires additional processing to make data changes.

## DML Execute Phase

To execute a DML statement:

- If the data and rollback blocks are not already in the buffer cache, the server process reads them from the data files into the buffer cache. The server process locks the rows that are to be modified.
- The server process records the changes to be made to the data buffers, as well as the undo changes. These changes are written to the redo log buffer before the in-memory data and rollback buffers are modified. This is called “write-ahead logging.”
- The rollback buffers contain values of the data before it is modified. The rollback buffers are used to store the before image of the data so that the DML statements can be rolled back if necessary. The data buffers record the new values of the data.
- The user gets the feedback from the DML operation (such as how many rows were affected by the operation).

All in-memory data blocks and undo blocks (in the buffer cache) that changed due to the DML are marked as dirty buffers. These buffers are not the same as the corresponding blocks on the disk. These buffers are not immediately written to disk by the Database Writer (DBW $n$ ) process. When a transaction is committed, the redo change records of the changes made to the blocks are immediately recorded in the redo log files by the Log Writer process and the dirty blocks are eventually written to disk by DBWR as determined by the incremental checkpoint algorithm or required by space pressure in the database buffer cache.

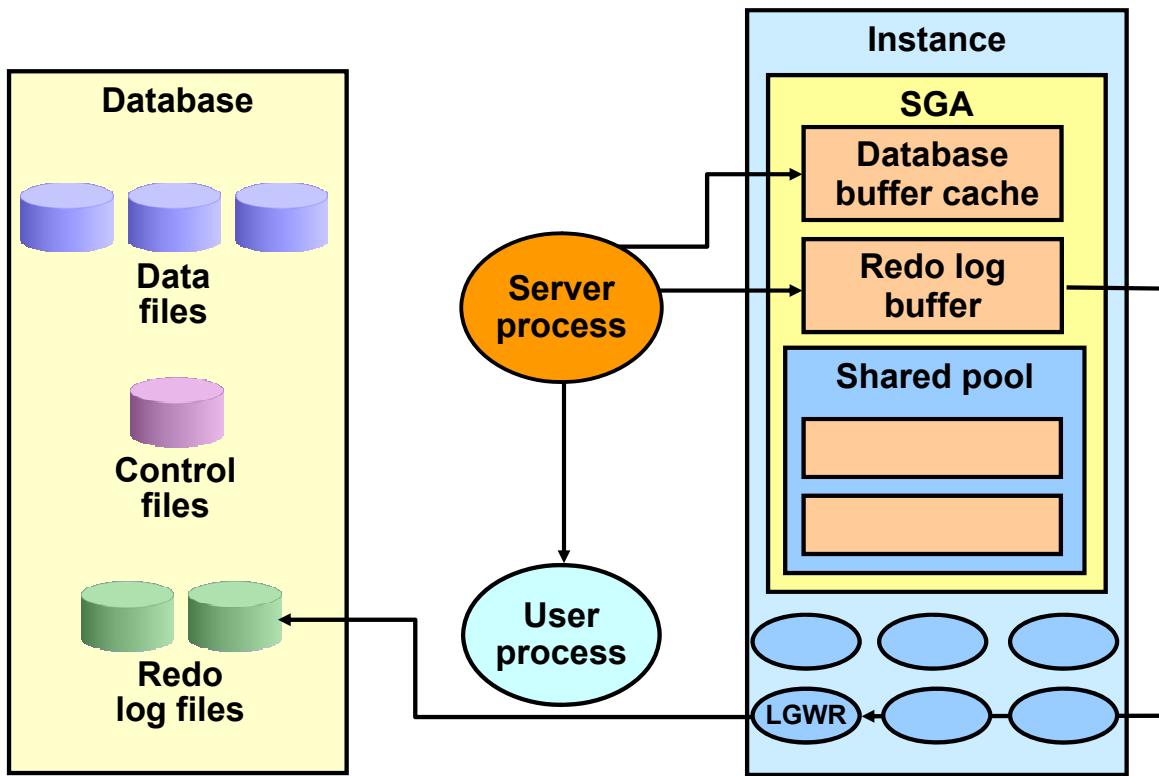
**Note:** The redo change records for a dirty block must be written to the redo log file before DBWR is allowed to write the dirty block to disk.

The processing of UPDATE, DELETE, or INSERT commands all use similar steps. The before image for a DELETE contains the column values in the deleted row, and the before image of an INSERT contains only the row location information.

Until a transaction is committed, the changes made to the blocks are only recorded in memory structures and are not written immediately to disk. The instance processes are following a lazy write algorithm that improves the overall performance. After a transaction is committed, it is permanent. The “committed” message is not issued until the LWGR process has recorded the redo information to disk ensuring complete recoverability. The DBW $n$  process writes the data blocks out to disk according to the checkpoint algorithm. A computer failure that causes the loss of the SGA before a transaction is committed will also lose these changes. The rule is that no transaction is permanent until it is committed.

For more details on the processing of database buffer cache, see the lesson titled “Tuning the Buffer Cache.”

# Commit Processing



ORACLE

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

## Fast Commit

The Oracle Database server uses a fast commit mechanism that guarantees the committed changes can be recovered in case of instance failure.

## System Change Number

Whenever a transaction commits, the Oracle Database server assigns a unique system change number (SCN) to the transaction. It is used by the Oracle Database server as an internal time stamp to synchronize data and to provide read consistency when data is retrieved from the data files. The SCN enables the instance to perform consistency checks without depending on the date and time of the operating system.

When a `COMMIT` is issued, the following steps are performed:

- The server process places a commit record, with the SCN, in the redo log buffer.
- The Log Writer background process (`LGWR`) performs a contiguous write of all the redo log buffer entries up to and including the commit record to the redo log files. This guarantees that the changes will not be lost even if there is an instance failure.
- The server process sends a transaction completion message to the user process.

`DBWN` eventually writes the actual data block changes back to disk based on its own internal timing mechanism and the incremental checkpoint settings.

## Role of the Oracle Optimizer

- The Oracle query optimizer determines the most efficient execution plan and is the most important step in the processing of any SQL statement.
- The optimizer:
  - Evaluates expressions and conditions
  - Uses object and system statistics
  - Decides how to access the data
  - Decides how to join tables
  - Decides which access path is most efficient



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

The optimizer is the part of the Oracle Database server that creates the execution plan for a SQL statement. The determination of the execution plan is an important step in the processing of any SQL statement and can greatly affect execution time.

The execution plan is a series of operations that are performed in sequence to execute the statement. The details of the various steps are shown in the lesson titled “Influencing the Optimizer.” The optimizer considers many factors related to the objects referenced and the conditions specified in the query. The information needed by the optimizer includes:

- Statistics gathered for the system (I/O, CPU, and so on) as well as schema objects (number of rows, index, and so on)
- Information in the dictionary
- WHERE clause qualifiers
- Hints supplied by the developer

When you use diagnostic tools such as Enterprise Manager, EXPLAIN PLAN, and SQL\*Plus AUTOTRACE, you can see the execution plan that the optimizer chooses.

**Note:** The optimizer has two names based on its functionality: the query optimizer or run-time optimizer and the Automatic Tuning Optimizer (ATO). The value of ATO depends on sharing SQL cursors. Cursor sharing is affected by the use of literals, the setting of the CURSOR\_SHARING parameter, and histograms.

**Optimizer operations:** For any SQL statement processed by the Oracle Database server, the optimizer performs the following operations:

- **Evaluation of expressions and conditions:** The optimizer first evaluates expressions and conditions containing constants as fully as possible.
- **Statement transformation:** For complex statements involving, for example, correlated subqueries or views, the optimizer might transform the original statement into an equivalent join statement.
- **Choice of optimizer approaches:** The optimizer determines the goal of the optimization.
- **Choice of access paths:** For each table accessed by the statement, the optimizer chooses one or more of the available access paths to obtain table data. The optimizer will skip certain access paths if there are no statistics available such as using bitmap indexes.
- **Choice of join orders:** For a join statement that joins more than two tables, the optimizer chooses which pair of tables is joined first, then which table is joined to the result, and so on.
- **Choice of join methods:** For any join statement, the optimizer chooses an operation to use to perform the join.

**Note:** The optimizer may not make the same decisions from one version of Oracle Database to the next. In recent versions because more information is available, the optimizer may make different decisions.

The optimizer works in two modes, the first and usual mode is the run-time optimizer that creates the execution plan at run time. In this mode, the optimizer is time limited; it can only consider a limited number of alternatives. The second mode is called the Automatic Tuning Optimizer (ATO). In this mode, the optimizer is given a much longer time to consider more options and gather statistics. The ATO can produce a better plan and create a SQL profile that will influence the optimizer to choose the better plan whenever the SQL statement is submitted in the future.

# Quiz

In the parse phase, the data dictionary is used to find the properties and statistics of objects named or implied in a SQL statement. The optimizer uses this information to build an execution plan. The optimizer is always invoked for a soft parse.

- a. True
- b. False



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

## Answer: b

The optimizer is always invoked for a hard parse, because some part of the cursor and associated execution plan is not found in the SQL area. A soft parse would not generally invoke the optimizer, because a soft parse finds the SQL cursor in the SQL area.

# Identifying Bad SQL

## Bad SQL:

- Uses more resources than necessary
- Has the following characteristics:
  - Long parse time
  - Excessive I/O (physical reads and writes)
  - Excessive buffer gets
  - Excessive CPU time
  - Excessive waits



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

One of the advantages of SQL is that it is possible to write different SQL statements that produce the same result. Any SQL statement that produces the correct result is a correct SQL statement. However, different SQL can require different amounts of resources. Bad SQL can be correct, but bad SQL is inefficient, taking more resources than necessary.

The symptoms of bad SQL can be any one of the characteristics listed in the slide. The Top SQL Report shown in the next slide provides a way to find those SQL statements that are consuming the most resources on your system.

Bad SQL can result from a bad design, poor coding, or from the optimizer choosing an inefficient execution plan. As a DBA, you seldom have control over the design or code, but you can influence the optimizer to produce a better execution plan.

Conceptually there is an optimum execution plan for any given result set from a given set of relational data. The optimizer attempts to find this optimum execution plan given the constraints of time and resources. The optimum plan may take a long time to find. For example, you would not be willing to wait 5 minutes for the optimizer to produce a plan that reduces the runtime by 5 seconds. The order that trial execution plans are evaluated by the optimizer, is influenced by many factors, including the way the SQL is written.

# TOP SQL Reports

## SQL ordered by CPU Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- %Total - CPU Time as a percentage of Total DB CPU
- %CPU - CPU Time as a percentage of Elapsed Time
- %IO - User I/O Time as a percentage of Elapsed Time
- Captured SQL account for 118.1% of Total CPU Time (s): 10
- Captured PL/SQL account for 8.6% of Total CPU Time (s): 10

CPU Time (s)	Executions	CPU per Exec (s)	%Total	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Module	SQL Text
3.61	4	0.90	36.37	3.65	98.83	0.00	057kbmvld1qqlk	EM Realtime Connection	select dbms_sqlline.report_sql...
1.98	4	0.50	20.00	2.00	99.10	0.00	09p3vf74541p5	EM Realtime Connection	SELECT /*+ no_monitor leading(...
1.30	1	1.30	13.11	1.33	98.05	0.01	27asrt9dxd17z	emagent_SQL_oracle_database	select a.capture_name capture...
0.83	1	0.83	8.33	0.85	97.70	1.52	f5agknuaraz7	emagent_SQL_oracle_database	select capture_name streams_n...
0.76	10	0.08	7.66	0.77	98.37	0.00	abt26ipdfxwhv	MMON_SLAVE	SELECT APPENDCHILDXML(:B2 , '...
0.62	1	0.62	6.28	6.18	10.07	45.55	8u809k64x3nzd	Admin Connection	begin DBMS_WORKLOAD_REPOSITORY...
0.51	1	0.51	5.17	0.53	97.02	2.34	1xa4scdlk8nab	emagent_SQL_oracle_database	select r.apply_name apply_nam...
0.50	1	0.50	5.08	0.54	93.08	4.08	cqf802wa2fy65	emagent_SQL_oracle_database	select r.apply_name apply_nam...

## SQL ordered by Gets

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- %Total - Buffer Gets as a percentage of Total Buffer Gets
- %CPU - CPU Time as a percentage of Elapsed Time
- %IO - User I/O Time as a percentage of Elapsed Time
- Total Buffer Gets: 653,363
- Captured SQL account for 147.5% of Total

Buffer Gets	Executions	Gets per Exec	%Total	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Module	SQL Text
496,634	4	124,158.50	76.01	3.65	98.8	0	057kbmvld1qqlk	EM Realtime Connection	select dbms_sqlline.report_sql...
234,866	4	58,716.50	35.95	2.00	99.1	0	09p3vf74541p5	EM Realtime Connection	SELECT /*+ no_monitor leading(...
103,698	10	10,369.80	15.87	0.77	98.4	0	abt26ipdfxwhv	MMON_SLAVE	SELECT APPENDCHILDXML(:B2 , '...
89,116	4	22,279.00	13.64	0.41	97.6	15.9	f58fb5n0yvr7c	EM Realtime Connection	select 'uptime' stat_type, rou...
21,816	1	21,816.00	3.34	6.18	10.1	45.5	8u809k64x3nzd	Admin Connection	begin DBMS_WORKLOAD_REPOSITORY...



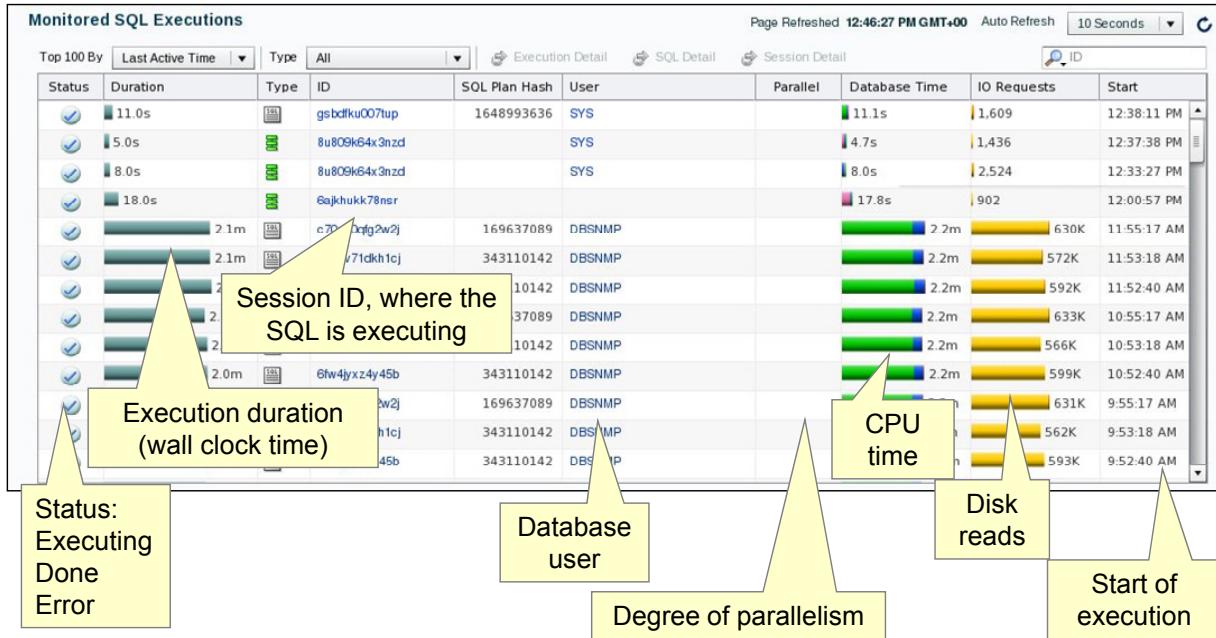
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The greatest return on investment in the tuning arena is SQL tuning. The Top SQL reports are very useful for identifying the statements that consume the most resources on your system. Studies have shown that typically 20% of the SQL statements consume 80% of the resources and 10% of the statements consume 50% of the resources. This means that by identifying and tuning the top SQL statements, you can improve the performance for the entire system.

Finding the top resource consuming SQL statements is simplified by using the Top SQL reports. Both AWR and Statspack reports include a set of Top SQL listings. Each report lists the top SQL statements sorted by resource usage in several categories. The categories are Elapsed Time, CPU Time, Gets, Reads, Executions, Parse Calls, Sharable Memory, and Version Count. The individual reports do not include the full SQL text, but a report of all SQL text by `SQL_ID` follows the individual reports.

Not all SQL statements are included in these reports by default. The number of statements included is controlled by the `topnsql` parameter setting for AWR, and Level and threshold settings in Statspack. For more details on Statspack parameters, see the appendix “Using Statspack.”

# SQL Monitoring



ORACLE®

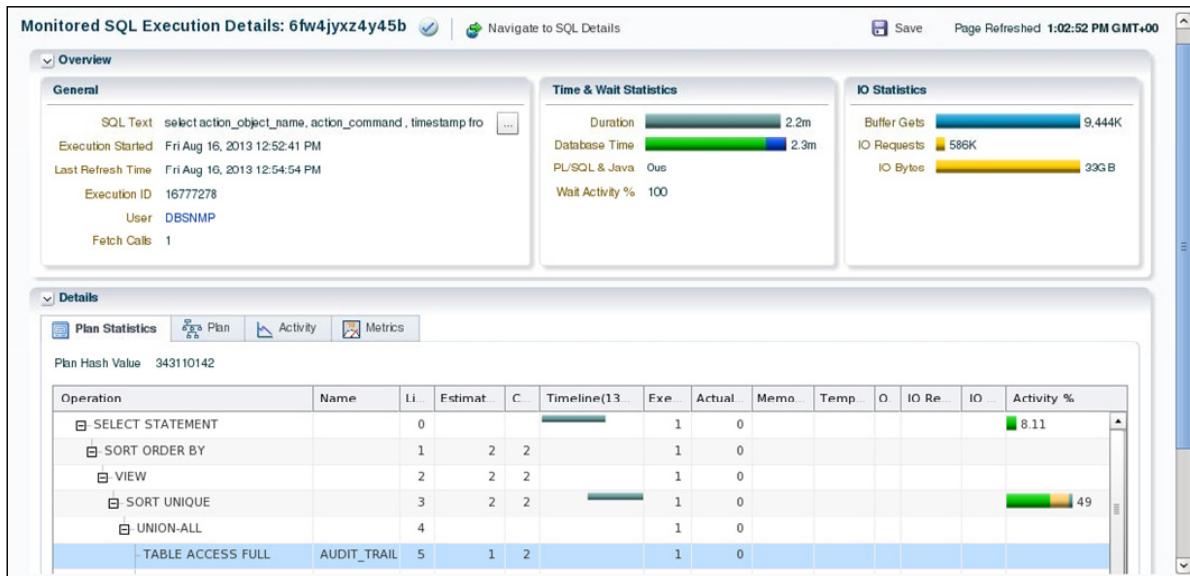
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can access the SQL Monitoring feature in Enterprise Manager Cloud Control by clicking SQL Monitoring in the Performance menu.

When you move the cursor over the values or symbols of each SQL execution, a relevant hint appears. The slide shows the actual SQL command being executed with the cursor on the SQL ID link.

When you click the link that shows the SQL ID, you navigate to the Monitored SQL Execution Details page, as shown in the following slide.

# Monitored SQL Execution Details



**ORACLE®**

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

The Monitored SQL Execution Details page provides additional information about the selected SQL ID. On this page, you can view the execution plan for the statement.

# Quiz

A SQL statement that uses more resources than necessary is considered bad SQL. Which of the following are characteristics of bad SQL?

- a. Does full table scans
- b. Does large amount of physical I/O
- c. Does not use indexes
- d. Has a large number of waits
- e. Has a large number of parses



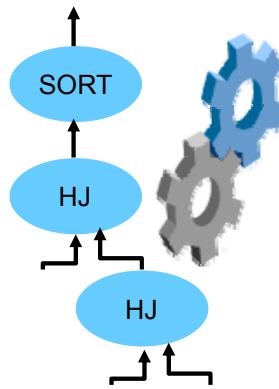
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

## Answer: b, d

Excessive I/O and excessive waits are the characteristics in this list that point to bad SQL. Full table scans and not using indexes could point to excessive I/Os but not necessarily. Large number of parses may point to an insufficient shared pool size rather than bad SQL.

## What Is an Execution Plan?

An execution plan is a set of steps that the optimizer performs when executing a SQL statement and performing an operation.



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

When a statement is executed, the server executes steps of the plan created by the optimizer. Each step either retrieves rows of data physically from the database or prepares them in some way for the user issuing the statement. The combination of steps that are used to run a statement is called an “execution plan.”

An execution plan includes an access method for each table that the statement accesses and an ordering of the tables (the join order). The optimizer also uses different methods to combine the rows from multiple tables (the join method). The steps of the execution plan are not performed in the order in which they are numbered.

The execution plan allows you to look into the methods that the optimizer has chosen. Sometimes the execution plan illustrates clearly why a statement is inefficient; for example, when a full table scan (FTS), involving many I/Os, is chosen over an index lookup. In this case, the question becomes why the optimizer chose the FTS. The lesson titled “Influencing the Optimizer” looks at this type question in more detail.

# Methods for Viewing Execution Plans

To view execution plans, use:

- Enterprise Manager SQL pages
- DBMS\_XPLAN methods to view plans from:
  - Automatic Workload Repository
  - V\$SQL\_PLAN
  - SQL Tuning Sets
  - Plan table
- SQL Trace (event 10046) with TKPROF
- SQL\*Plus AUTOTRACE
- EXPLAIN PLAN



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

Enterprise Manager allows you to examine the execution plans of SQL statements in any of the AWR, V\$SQL\_PLAN view, or a SQL tuning set. Enterprise Manager uses DBMS\_XPLAN to obtain these execution plans. You can access these plans by going through the Top SQL pages, SQL Monitor, or SQL Advisor, depending on which set of SQL plans you want to see.

The DBMS\_XPLAN package has several procedures to obtain and format the execution plan from several sources including the AWR repository, V\$SQL\_PLAN, SQL plan baselines, SQL Tuning sets, and a plan table.

- The Automatic Workload Repository (AWR) is a built-in repository in Oracle Database. At regular intervals, the Oracle Database server makes a snapshot of all of its vital statistics and workload information and stores this in the AWR, including a listing of high-resource SQL statements. The AWR data includes the execution plans.
- The V\$SQL\_PLAN view contains information about executed SQL statements, and their execution plans, that are still in the shared pool. It is also called the SQL cache.
- SQL Tuning sets contain SQL statements and the associated execution plans.
- A plan table can contain the execution plan produced by the EXPLAIN PLAN SQL command.

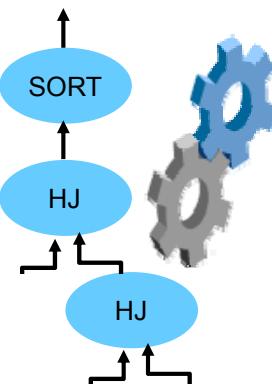
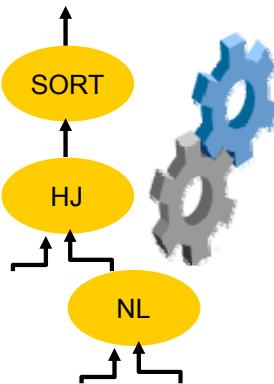
The SQL Trace utility is used to measure timing statistics for a SQL statement. The same trace information is produced by setting event 10046. Both trace outputs can be formatted with the TKPROF utility.

The AUTOTRACE command available in SQL\*Plus generates the PLAN\_TABLE output and statistics about the performance of a query. This command provides many of the same statistics as SQL Trace, such as disk reads and memory reads.

The EXPLAIN PLAN command allows you to view the execution plan that the optimizer uses to execute a SQL statement without executing the SQL statement. The execution plan produced by the EXPLAIN PLAN command may not be the same plan produced by the runtime optimizer.

# Uses of Execution Plans

- Determining the current execution plan
- Identifying the effect of indexes
- Determining access paths
- Verifying the use of indexes
- Verifying which execution plan may be used



**ORACLE**

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

Viewing execution plans is used to:

- Determine the current execution plan
- Identify the effect of creating an index on a table
- Find cursors containing a certain access path (for example, full table scan or index range scan)
- Identify indexes that are, or are not, selected by the optimizer
- Determine whether the optimizer selects the particular execution plan (for example, nested loops join) expected by the developer

Comparing costs and measuring actual execution times and resource usage of one SQL executing various plans are ways to help you decide what execution plan is best for that SQL, but comparing execution plans is insufficient to determine the lowest cost plan.

Execution plans can show how the optimizer is influenced by such things as indexes, statistics on database objects, changes to initialization parameter values, and the migration of the application or the database to a new release. But these items are global in nature. They can affect the entire system. SQL Access Advisor is a better tool for analyzing the impact of global changes.

By default, execution plans are not kept after the SQL ages out of the shared pool. If previously used plans are kept in user-defined tables, or loaded as baseline plans, it is then possible to identify how changes in the performance of a SQL statement can be correlated with changes in the execution plan for that statement.

## **DBMS\_XPLAN Package: Overview**

The DBMS\_XPLAN package:

- Provides an easy way to display the output from:
  - EXPLAIN PLAN command
  - Automatic Workload Repository (AWR)
  - V\$SQL\_PLAN and V\$SQL\_PLAN\_STATISTICS\_ALL fixed views
- Supplies three table functions that can be used to retrieve and display the execution plan:
  - DISPLAY
  - DISPLAY\_AWR
  - DISPLAY\_CURSOR



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

The DBMS\_XPLAN package provides an easy way to display the output of the EXPLAIN PLAN command in several predefined formats. You can also use the DBMS\_XPLAN package to display the plan of a statement stored in the AWR. Furthermore, it provides a way to display the SQL execution plan and SQL execution run-time statistics for cached SQL cursors based on the information stored in the V\$SQL\_PLAN and V\$SQL\_PLAN\_STATISTICS\_ALL fixed views.

The DBMS\_XPLAN package supplies three table functions that can be used to retrieve and display the execution plan:

- DISPLAY formats and displays the contents of a plan table from a PLAN\_TABLE.
- DISPLAY\_AWR formats and displays the contents of the execution plan of a stored SQL statement in the AWR.
- DISPLAY\_CURSOR formats and displays the contents of the execution plan of any loaded cursor from the V\$SQL\_PLAN view.

The methods of this package contain a `FORMAT` parameter that allows you to specify the detail level of displayed plans.

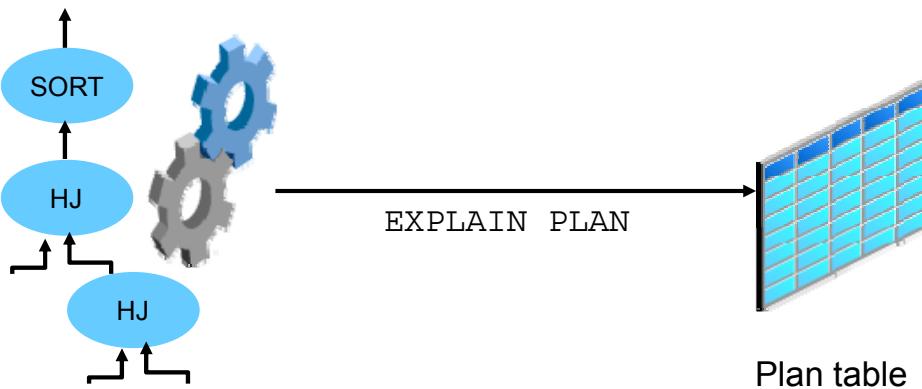
- **BASIC:** Displays the minimum information in the plan (the operation ID, the object name, and the operation option)
- **TYPICAL:** Default; displays the most relevant information in the plan. Partition pruning, parallelism, and predicates are displayed only when available.
- **ALL:** Maximum level; includes information displayed with the `TYPICAL` level and adds projection information as well as SQL statements generated for parallel execution servers (only if parallel)
- **SERIAL:** Similar to `TYPICAL`, except that the parallel information is not displayed, even if the plan executes in parallel

This package runs with the privileges of the calling user, not the package owner (`SYS`). The `DISPLAY_CURSOR` table function requires select privileges on the following fixed views: `V$SQL_PLAN`, `V$SESSION`, and `V$SQL_PLAN_STATISTICS_ALL`. Using the `DISPLAY_AWR` function requires `SELECT` privileges on `DBA_HIST_SQL_PLAN`, `DBA_HIST_SQLTEXT`, and `V$DATABASE`. All these privileges are automatically granted as part of the `SELECT_CATALOG_ROLE`. However, granting this role indiscriminately is not advised as it may cause security concerns.

Both the `DISPLAY_CURSOR` and `DISPLAY_AWR` functions accept `SQL_ID` as an argument (as shown in examples later in this lesson). `SQL_ID` of a statement can be obtained by querying `V$SQL` or `DBA_HIST_SQLTEXT`.

## EXPLAIN PLAN Command

- Generates an optimizer execution plan
- Stores the plan in the `PLAN` table
- Does not execute the statement itself



**ORACLE**

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

The `EXPLAIN PLAN` command is used to generate the execution plan that the optimizer uses to execute a SQL statement. It does not execute the statement but simply produces the plan that may be used and inserts this plan into a table. If you examine the plan, you can see how the Oracle Server executes the statement.

To use `EXPLAIN PLAN`, you must:

- First use the `EXPLAIN PLAN` command to explain a SQL statement
- Retrieve the plan steps by using the methods in the `DBMS_XPLAN` package

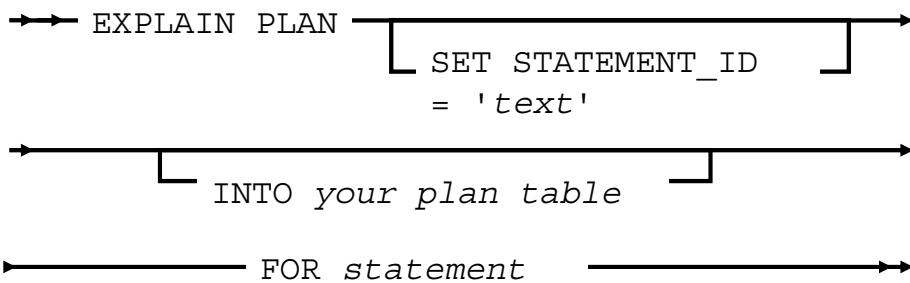
`PLAN_TABLE` is automatically created as a global temporary table to hold the output of an `EXPLAIN PLAN` statement for all users. `PLAN_TABLE` is the default sample output table into which the `EXPLAIN PLAN` statement inserts rows describing execution plans.

**Note:** The `EXPLAIN PLAN` may produce a plan that is different from the actual plan that is used by the optimizer, due to a variety of reasons:

- The `EXPLAIN PLAN` command does not have access to the bind variables.
- The SQL\*Plus session may have a different environment due to logon triggers or session parameter settings.

`V$SQLPLAN` will have the actual plan used.

## EXPLAIN PLAN Command: Example



```

EXPLAIN PLAN
SET STATEMENT_ID = 'demo01' FOR
SELECT e.last_name, d.department_name
FROM hr.employees e, hr.departments d
WHERE e.department_id = d.department_id;
Explained.
  
```

**Note:** The EXPLAIN PLAN command does not actually execute the statement.

ORACLE

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

This command inserts the execution plan of the SQL statement in the plan table and adds the name tag `demo01` for future reference. The tag is optional. You can also use the following syntax:

```

EXPLAIN PLAN
FOR
SELECT e.last_name, d.department_name
FROM hr.employees e, hr.departments d
WHERE e.department_id = d.department_id;
  
```

## EXPLAIN PLAN Command: Output

```
SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY());
```

```
Plan hash value: 1343509718

-----+-----+-----+-----+-----+-----+
| Id | Operation          | Name      | Rows   | Bytes  | Cost (%CPU) |
-----+-----+-----+-----+-----+-----+
| 0  | SELECT STATEMENT    |           | 106   | 2862   | 6  (17)    |
| 1  |  MERGE JOIN          |           | 106   | 2862   | 6  (17)    |
| 2  |    TABLE ACCESS BY INDEX ROWID | DEPARTMENTS | 27   | 432   | 2  (0)    |
| 3  |    INDEX FULL SCAN   | DEPT_ID_PK  | 27   | 432   | 1  (0)    |
* 4  |    SORT JOIN          |           | 107   | 1177   | 4  (25)   |
| 5  |    TABLE ACCESS FULL | EMPLOYEES  | 107   | 1177   | 3  (0)    |
-----+-----+-----+-----+-----+-----+

Predicate Information (identified by operation id):
-----
4 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
   filter("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")

18 rows selected.
```



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

The DISPLAY function of the DBMS\_XPLAN package can be used to format and display the last statement stored in a plan table.

The slide shows the result of using the DBMS\_XPLAN package as shown on the previous page to retrieve the information from the PLAN table in that example.

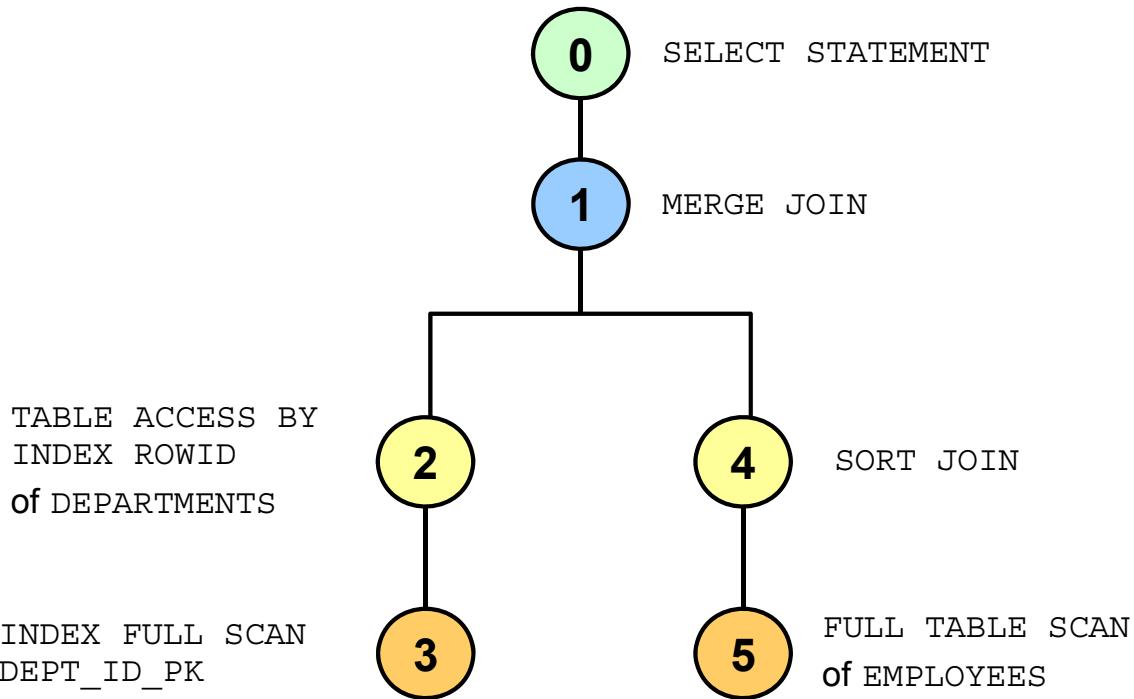
You can also use the following syntax to retrieve information from the PLAN table:

```
SELECT plan_table_output FROM
  TABLE(dbms_xplan.display('plan_table','demo01','serial'));
```

The output is the same as the one shown in the slide. In this example, you can substitute the name of another plan table instead of PLAN\_TABLE, and 'demo01' represents the statement ID.

You can run the utlxpls.sql script (located in the ORACLE\_HOME/rdbms/admin directory) to display the EXPLAIN PLAN of the last statement explained. This script uses the DISPLAY table function from the DBMS\_XPLAN package.

# Reading an Execution Plan



**ORACLE**

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

From the execution plan, you can construct an execution tree (or *parse tree*) to get a better idea of how a statement is processed. To construct the tree, start with step 1. Then find all steps with a parent of step 1 and draw them as children or branches below step 1. Repeat for each step finding all the children of that step until all steps are accounted for. To each step in the execution plan, Oracle Database assigns a number representing the ID column of the `PLAN_TABLE`. Each step is depicted by a “node.” The result of each node’s operation is passed to its parent node, which uses it as input.

The sequence of steps is determined by the parent/child relationships of the steps. Each step of the execution plan either retrieves rows from the database or accepts rows as input from one or more other steps, also known as “row sources.” The child step is performed at least once and feeds the parent. When a parent has multiple children, each child is performed sequentially in order of step position. If the lower child steps are arranged left to right, the plan can be read left to right and bottom to top. In the diagram, the numbers correspond to the ID values in the `PLAN` table (see the previous slide). The optimizer retrieves the rows from the `DEPARTMENTS` table using an index scan by performing a `FULL INDEX SCAN` on the primary key column. It then performs a `FULL TABLE SCAN` and `SORT` operation on the `EMPLOYEES` table. These two result sets are then `MERGED` to get the end result for the query.

**Note:** Step 3 is a full scan of the PK index on the `DEPARTMENTS` table; the operation retrieves the index entries, and then gets the table rows in *sorted order*. A fast Full Index Scan does not return the index entries in sorted order, and so would require a sort operation as well.

## Using the V\$SQL\_PLAN View

- V\$SQL\_PLAN provides a way of examining the execution plan for cursors that were recently executed.
- Information in V\$SQL\_PLAN is very similar to the output of an EXPLAIN PLAN statement:
  - EXPLAIN PLAN shows a theoretical plan that can be used if this statement were to be executed.
  - V\$SQL\_PLAN contains the actual plan used.



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

The V\$SQL\_PLAN view provides a way of examining the execution plan for cursors that were recently executed. The information in this view is very similar to the output from the PLAN\_TABLE. However, EXPLAIN PLAN shows a theoretical plan that can be used if this statement were to be executed, whereas V\$SQL\_PLAN contains the actual plan used. The execution plan obtained by the EXPLAIN PLAN statement can be different from the actual execution plan used, due to bind variable peeking, the setting of the cursor\_sharing parameter.

V\$SQL\_PLAN shows the plan for a specific cursor. Each SQL statement may have multiple associated cursors, with each cursor identified by a CHILD\_NUMBER. For example, the same statement executed by different users has different cursors associated with it if the object being referenced is in a different schema. Different hints or different values of bind variables can cause different cursors. The V\$SQL\_PLAN table can be used to see the different plans for different child cursors of the same statement.

**Note:** Another useful view is V\$SQL\_PLAN\_STATISTICS, which provides the execution statistics of each operation in the execution plan for each cached cursor. Also, the V\$SQL\_PLAN\_STATISTICS\_ALL view combines information from V\$SQL\_PLAN with execution statistics from V\$SQL\_PLAN\_STATISTICS and V\$SQL\_WORKAREA.

## V\$SQL\_PLAN Columns

HASH_VALUE	Hash value of the parent statement in the library cache
ADDRESS	Address of the handle to the parent for this cursor
CHILD_NUMBER	Child cursor number using this execution plan
POSITION	Order of processing for operations that all have the same PARENT_ID
PARENT_ID	ID of the next execution step that operates on the output of the current step
ID	Number assigned to each step in the execution plan

**Note:** This is only a partial listing of the columns.



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

Almost all the columns of the V\$SQL\_PLAN view are displayed in the PLAN\_TABLE columns. The columns that have the same names have the same meanings in both views.

The columns ADDRESS and HASH\_VALUE can be used to join with V\$SQLAREA to add the cursor-specific information.

The ADDRESS, HASH\_VALUE, and CHILD\_NUMBER columns can be used to join with V\$SQL to add the child cursor-specific information.

## Querying V\$SQL\_PLAN

```
SELECT PLAN_TABLE_OUTPUT FROM
TABLE(DBMS_XPLAN.DISPLAY_CURSOR('cfz0cdrukrfdn'));
```

```
SQL_ID cfz0cdrukrfdn, child number 0
-----
SELECT e.last_name, d.department_name
FROM hr.employees e, hr.departments d WHERE
e.department_id =d.department_id

Plan hash value: 1343509718

| Id | Operation          | Name      | Rows | Bytes | Cost (%CPU|
|---|---|---|---|---|---|
| 0 | SELECT STATEMENT   |           |       |       | 6 (100) |
| 1 |  MERGE JOIN         |           |       |       | 6 (17)  |
| 2 |    TABLE ACCESS BY INDEX ROWID | DEPARTMENTS | 27 | 432 | 2 (0)  |
| 3 |    INDEX FULL SCAN  | DEPT_ID_PK  | 27 |       | 1 (0)  |
| * 4 |    SORT JOIN        |           | 107 | 1177 | 4 (25) |
| 5 |    TABLE ACCESS FULL | EMPLOYEES  | 107 | 1177 | 3 (0)  |

Predicate Information (identified by operation id):
-----
4 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
      filter("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")

24 rows selected.
```



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

You can query V\$SQL\_PLAN by using the DBMS\_XPLAN.DISPLAY\_CURSOR() function to display the current or last executed statement (as shown in the example). You can pass the value of the SQL\_ID for the statement as a parameter to get the execution plan for a given statement. To obtain the SQL\_ID:

```
SELECT e.last_name, d.department_name
FROM hr.employees e, hr.departments d
WHERE e.department_id =d.department_id;

SELECT SQL_ID, SQL_TEXT FROM V$SQL
WHERE SQL_TEXT LIKE '%SELECT e.last_name,%' ;

13saxr0mmz1s3 select SQL_id, sql_text from v$SQL ...
cfz0cdrukrfdn SELECT e.last_name, d.department_name ...
```

The FORMAT parameter controls the level of detail for the plan. In addition to the standard values (BASIC, TYPICAL, SERIAL, and ALL), there are two additional supported values to display run-time statistics for the cursor:

- RUNSTATS\_LAST: Displays the run-time statistics for the last execution of the cursor
- RUNSTATS\_TOT: Displays the total aggregated run-time statistics for all executions of a specific SQL statement since the statement was first parsed and executed

## V\$SQL\_PLAN\_STATISTICS View

- V\$SQL\_PLAN\_STATISTICS provides actual execution statistics.
- V\$SQL\_PLAN\_STATISTICS\_ALL enables side-by-side comparisons of the optimizer estimates.



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

The V\$SQL\_PLAN\_STATISTICS view provides the actual execution statistics for every operation in the plan, such as the number of output rows and elapsed time. All statistics, except the number of output rows, are cumulative. For example, the statistics for a join operation also includes the statistics for its two inputs. The statistics in V\$SQL\_PLAN\_STATISTICS are available for cursors that have been compiled with the STATISTICS\_LEVEL initialization parameter set to ALL.

The V\$SQL\_PLAN\_STATISTICS\_ALL view contains memory-usage statistics for row sources that use SQL memory (sort or hash join). This view concatenates information in V\$SQL\_PLAN with execution statistics from V\$SQL\_PLAN\_STATISTICS and V\$SQL\_WORKAREA.

# Querying the AWR

```
SELECT PLAN_TABLE_OUTPUT FROM TABLE
(DBMS_XPLAN.DISPLAY_AWR('454rug2yva18w'));
```

```
PLAN_TABLE_OUTPUT
-----
SQL_ID 454rug2yva18w
-----
select /* example */ * from hr.employees natural join hr.departments

Plan hash value: 2052257371

-----
| Id | Operation          | Name      | Rows | Bytes | Cost (%CPU) | Time      |
|---|---|---|---|---|---|---|
| 0 | SELECT STATEMENT   |           |       |       | 6 (100)    |           |
| 1 | HASH JOIN          |           | 11   | 968   | 6 (17)    | 00:00:01  |
| 2 | TABLE ACCESS FULL  | DEPARTMENTS | 11   | 220   | 2 (0)     | 00:00:01  |
| 3 | TABLE ACCESS FULL  | EMPLOYEES   | 107  | 7276  | 3 (0)     | 00:00:01  |
-----
```



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

You can use the `DBMS_XPLAN.DISPLAY_AWR()` function to display all stored plans in the AWR. In the example shown, you are passing in a `SQL_ID` as an argument. The steps to complete this example are as follows:

1. Execute the SQL statement.

```
SQL> select /* example */ *
  2> from hr.employees natural join hr.departments;
```

2. Query `V$SQL_TEXT` to obtain the `SQL_ID`.

```
SQL> select sql_id, sql_text from v$SQL
  2> where sql_text like '%example%';
```

SQL_ID	SQL_TEXT
F8tc4anpz5cdb	select sql_id, sql_text from v\$SQL ...
454rug2yva18w	select /* example */ * from ...

3. Using the `SQL_ID`, verify that this statement has been captured in the `DBA_HIST_SQLTEXT` dictionary view. If the query does not return rows, then it indicates that the statement has not yet been loaded in the AWR.

```
SQL> SELECT SQL_ID, SQL_TEXT FROM dba_hist_sqltext WHERE SQL_ID = '454rug2yva18w';
no rows selected
```

You can take a manual AWR snapshot rather than wait for the next snapshot (which occurs every hour). The SQL may not be captured unless it is in the `topnsql` range, so you can force all the SQL statements to be captured by changing the `topnsql` range with the `MODIFY_SNAPSHOT_SETTINGS` procedure. Then check to see if it has been captured in `DBA_HIST_SQLTEXT`:

```
SQL> exec -
2> DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS(
3> topnsql => 'MAXIMUM');
```

```
PL/SQL procedure successfully completed.
```

```
SQL> exec DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT();
```

```
PL/SQL procedure successfully completed.
```

```
SQL> exec -
2> DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS(
3> topnsql => 'DEFAULT');
```

```
PL/SQL procedure successfully completed.
```

```
SQL> SELECT SQL_ID, SQL_TEXT FROM dba_hist_sqltext WHERE SQL_ID = '454rug2yva18w';
SQL_ID          SQL_TEXT
-----          -----
454rug2yva18w  select /* example */ * from ...
```

#### 4. Use the `DBMS_XPLAN.DISPLAY_AWR()` function to retrieve the execution plan:

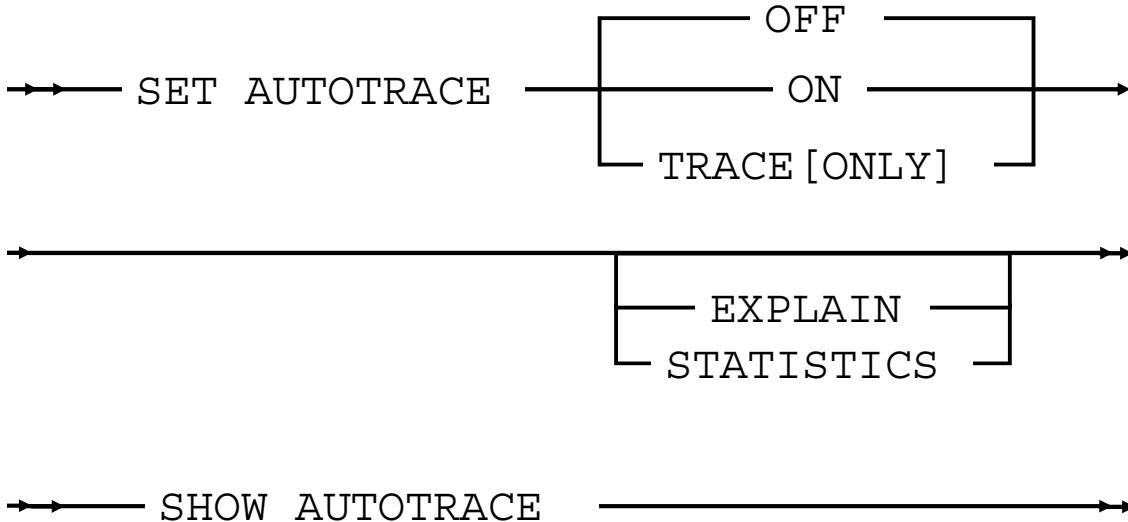
```
SQL>SELECT PLAN_TABLE_OUTPUT FROM TABLE
(DBMS_XPLAN.DISPLAY_AWR('454rug2yva18w'));
```

```
PLAN_TABLE_OUTPUT
-----
SQL_ID 454rug2yva18w
-----
select /* example */ * from hr.employees natural join hr.departments

Plan hash value: 2052257371

-----
| Id  | Operation           | Name      | Rows | Bytes | Cost (%CPU) | Time      |
-----
|   0 | SELECT STATEMENT    |           |     |       |    7 (100)  |           |
|   1 | HASH JOIN            |           | 11  | 968   |    7 (15)  | 00:00:01  |
|   2 | TABLE ACCESS FULL   | DEPARTMENTS | 11  | 220   |    3 (0)   | 00:00:01  |
|   3 | TABLE ACCESS FULL   | EMPLOYEES  | 107 | 7276  |    3 (0)   | 00:00:01  |
-----
```

## SQL\*Plus AUTOTRACE



ORACLE

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

In SQL\*Plus, you can automatically obtain the execution plan and some additional statistics about the running of a SQL command by using the `AUTOTRACE` setting. Unlike the `EXPLAIN PLAN` command, the statement is actually run. However, you can choose to suppress the display of the statement results by specifying `AUTOTRACE TRACEONLY EXPLAIN`.

`AUTOTRACE` is a convenient diagnostic tool for SQL statement tuning. Because it is purely declarative, it is easier to use than `EXPLAIN PLAN`.

### Command Options

- `OFF`: Disables autotracing SQL statements
- `ON`: Enables autotracing SQL statements
- `TRACEONLY`: Enables autotracing SQL statements and suppresses statement output
- `EXPLAIN`: Displays execution plans but does not display statistics
- `STATISTICS`: Displays statistics but does not display execution plans

**Note:** If both the `EXPLAIN` and `STATISTICS` command options are omitted, execution plans and statistics are displayed by default.

# Using SQL\*Plus AUTOTRACE

- To start tracing statements using AUTOTRACE:

```
set autotrace on
```

- To hide statement output:

```
set autotrace traceonly
```

- To display only execution plans:

```
set autotrace traceonly explain
```

- Control the layout with column settings



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

## Prerequisites

To access STATISTICS data, you must have access to several dynamic performance tables. You can grant access by using the PLUSTRACE role created in the plustrce.sql script. You must run the script as the SYS user, and then anyone with the DBA role may grant the PLUSTRACE role to users who want to use the STATISTICS option of AUTOTRACE.

## Examples

The slide shows examples of the AUTOTRACE command.

### Controlling the AUTOTRACE Execution Plan Layout

The execution plan consists of four columns displayed in the following order:

- ID\_PLUS\_EXP: Line number of each step
- PARENT\_ID\_PLUS\_EXP: Parent step line number
- PLAN\_PLUS\_EXP: Report step
- OBJECT\_NODE\_PLUS\_EXP: Database links or parallel query servers used

You can change the format of these columns, or suppress them, by using the SQL\*Plus COLUMN command. For additional information, see *Oracle SQL\*Plus User's Guide and Reference*.

## SQL\*Plus AUTOTRACE: Statistics

```
set autotrace traceonly statistics  
  
SELECT *  
FROM products;
```

```
Statistics  
-----  
      1 recursive calls  
      0 db block gets  
      9 consistent gets  
      3 physical reads  
      0 redo size  
 15028 bytes sent via SQL*Net to client  
   556 bytes received via SQL*Net from client  
      6 SQL*Net roundtrips to/from client  
      0 sorts (memory)  
      0 sorts (disk)  
    72 rows processed
```



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

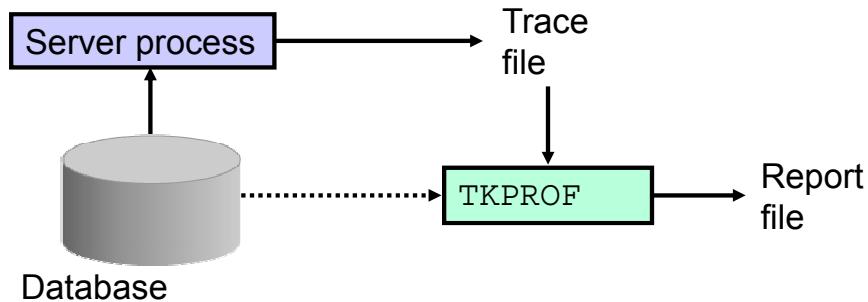
AUTOTRACE displays several statistics, not all of them relevant to the discussion at this stage. The most important statistics are the following:

- db block gets: Number of logical I/Os for current gets
- consistent gets: Reads of buffer cache blocks
- physical reads: Number of blocks read from disk
- redo size: Amount of redo generated (for DML statements)
- sorts (memory): Number of sorts performed in memory
- sorts (disk): Number of sorts performed using temporary disk storage

**Note:** DB block gets are reads of the current blocks in the buffer cache. Consistent gets are reads of buffer cache blocks that have undo data. Physical reads are reads of blocks from disk. DB block gets, consistent gets, and physical reads are the three statistics that are usually monitored. These should be low compared to the number of rows retrieved. Sorts should be performed in memory rather than on disk.

## SQL Trace Facility

- Is usually enabled at the session level
- Gathers session statistics for SQL statements grouped by session
- Produces output that can be formatted by TKPROF



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

If you are using Standard Edition or do not have the Diagnostics Pack, the SQL Trace facility and TKPROF enable you to collect the statistics for SQL executions plans to compare performance. A good way to compare two execution plans is to execute the statements and compare the statistics to see which one performs better. SQL Trace writes its session statistics output to a file, and you use TKPROF to format it. You can use these tools along with EXPLAIN PLAN to get the best results.

**SQL Trace facility:**

- Can be enabled for a session or for an instance
- Reports on volume and time statistics for the parse, execute, and fetch phases
- Produces output that can be formatted by TKPROF

When the SQL Trace facility is enabled for a session, the Oracle Database server generates a trace file containing session statistics for traced SQL statements for that session. When the SQL Trace facility is enabled for an instance, the Oracle Database server creates trace files for all sessions.

**Note:** SQL Trace involves some overhead, so you usually do not want to enable SQL Trace at the instance level.

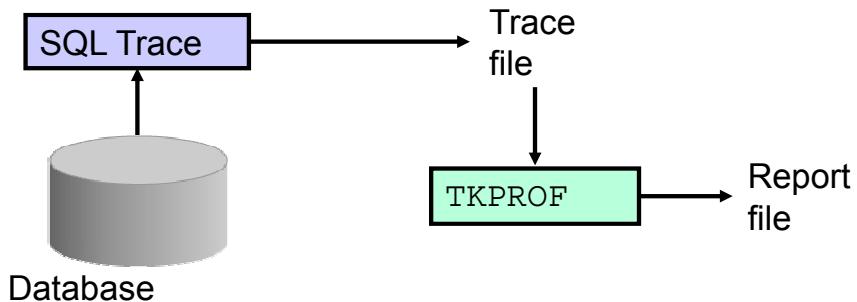
The SQL Trace facility provides performance information about individual SQL statements. SQL Trace provides the following, including row source information:

- Parse, execute, and fetch counts
- CPU and elapsed times
- Physical reads and logical reads
- Number of rows processed
- Misses on the library cache
- Username under which each parse occurred
- Each commit and rollback
- Row operations showing the actual execution plan of each SQL statement
- Number of rows, number of consistent reads, number of physical reads, number of physical writes, and time elapsed for each operation on a row

**Note:** A summary for each trace file can be obtained using the TKPROF utility.

# How to Use the SQL Trace Facility

1. Set the initialization parameters.
2. Enable tracing.
3. Run the application.
4. Disable tracing.
5. Close the session.
6. Format the trace file.
7. Interpret the output.



**ORACLE**

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

To use SQL Trace, perform the following steps:

1. Set the appropriate initialization parameters.
2. Enable SQL Trace.
3. Run the application (and disable tracing when done).
4. Disable SQL Trace.
5. Close the session.(Closing the session also disables tracing at the session level.)
6. Format the trace file produced by SQL Trace with TKPROF.
7. Interpret the output and tune the SQL statements when needed.

Running SQL Trace increases system overhead. Use SQL Trace only when required, and use it at the session level rather than at the instance level.

**Note:** This example assumes the use of dedicated servers. In a shared server environment, XA, or application-level connection pooling, a single session may be serviced by multiple servers. All of the servers involved need to be traced, and the trace files are combined by using the `trcsess` utility, and then submitted to TKPROF for formatting. For more information about `trcsess`, see the lesson titled “Monitoring Applications.”

# Initialization Parameters

```
STATISTICS_LEVEL = {BASIC|TYPICAL|ALL}
TIMED_STATISTICS = {false|true}
MAX_DUMP_FILE_SIZE = {n|unlimited}
DIAGNOSTIC_DEST={directory_path|$ORACLE_BASE/diag}
```



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

Several initialization parameters relate to SQL Trace.

## STATISTICS\_LEVEL

The STATISTICS\_LEVEL initialization parameter controls all major statistics collections or advisories in the database. This parameter sets the statistics collection level for the database. Depending on the setting of STATISTICS\_LEVEL, certain advisories or statistics are collected:

- **BASIC:** No advisories or statistics are collected. Monitoring and many automatic features are disabled. Oracle does not recommend this setting because it disables important Oracle features.
- **TYPICAL:** This is the default value and ensures collection for all major statistics while providing best overall database performance. This setting should be adequate for most environments. TYPICAL causes TIMED\_STATISTICS to be enabled.
- **ALL:** All of the advisories or statistics that are collected with the TYPICAL setting are included, along with timed operating system statistics and row source execution statistics.

This view lists the status of the statistics or advisories controlled by STATISTICS\_LEVEL.

**TIMED\_STATISTICS**

The SQL Trace facility provides a variety of information about SQL execution within a process, optionally including timing information. If you want timing information, this parameter must be set to TRUE. The value of the STATISTICS\_LEVEL parameter automatically determines the value of this parameter. The value of the TIMED\_STATISTICS parameter cannot be set to FALSE if the value of STATISTICS\_LEVEL is set to TYPICAL or ALL.

You can set this parameter separately from the STATISTICS\_LEVEL by setting the TIMED\_STATISTICS parameter in the parameter file with:

```
ALTER SYSTEM SET TIMED_STATISTICS = TRUE
```

The parameter also can be set dynamically for a particular session with the following command:

```
ALTER SESSION SET TIMED_STATISTICS = TRUE
```

The timing statistics have a resolution measured in microseconds.

**MAX\_DUMP\_FILE\_SIZE**

When the SQL Trace facility is enabled at the instance level, every call to the server produces a text line in a file in the operating system's file format. The maximum size of each file (in operating system blocks) is limited by this initialization parameter. This is a dynamic parameter as well as a session parameter.

**Warning:** The default value is UNLIMITED, so the trace files can grow to fill the file system.

**DIAGNOSTIC\_DEST** is the root directory for the Automatic Diagnostic Repository. The default for this directory is derived from the ORACLE\_BASE environment variable; on UNIX, it is \$ORACLE\_BASE/diag. The file generated while the Trace Facility is enabled will be placed in the trace sub-directory of this repository at  
`./rdbms/<db_name>/<instance_name>/trace.`

**Obtaining Information About Parameter Settings**

You can display current parameter values by querying the V\$PARAMETER view:

```
SQL> SELECT name, value
  2  FROM v$parameter
  3 WHERE name LIKE '%dest%';
```

Or you can use the following:

```
SQL> SHOW PARAMETER dest
```

# Enabling SQL Trace

- For your current session:

```
SQL> EXEC dbms_monitor.session_trace_enable;
```

```
SQL> EXECUTE dbms_session.set_sql_trace(true);
```

- For any session:

```
SQL> EXECUTE dbms_monitor.session_trace_enable
      2  (session_id, serial_id, waits, binds);
```

- For instance-wide tracing:

```
SQL> EXEC dbms_monitor.database_trace_enable();
```



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

You can use the commands shown to enable SQL Trace for your session. The procedure calls are useful when you want to enable or disable SQL Trace from within a PL/SQL unit. The DBA can also enable SQL Trace for another user's session with a supplied package.

```
SQL> EXECUTE dbms_monitor.session_trace_enable
      2  (session_id, serial_id);
```

In this procedure call, `session_id` and `serial_id` are the values in the `SID` and `SERIAL#` columns of `V$SESSION`, a data dictionary view commonly used by database administrators.

To enable SQL Trace for an entire instance use the `DATABASE_TRACE_ENABLE` procedure in the `DBMS_MONITOR` package. Tracing of both wait events and bind values may be enabled in the `DBMS_MONITOR` methods as.

```
SQL> EXECUTE dbms_monitor.session_trace_enable
      2  (session_id, serial_id,
      3  waits => TRUE, binds => TRUE);
```

**Warning:** Instance-wide tracing produces a large number of trace files and impacts performance.

**Note:** `EXECUTE` must be granted on the `DBMS_MONITOR` package before it can be used.

## Disabling SQL Trace

- For your current session:

```
SQL> EXEC dbms_monitor.session_trace_disable;
```

```
SQL> EXECUTE dbms_session.set_sql_trace(false);
```

- For any session:

```
SQL> EXECUTE dbms_monitor.session_trace_disable  
      2  (session_id, serial_id);
```

- For instance-wide tracing:

```
SQL> EXEC dbms_monitor.database_trace_disable();
```



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

When you have finished tuning, disable SQL Trace by using any of the preceding methods, substituting the word FALSE for TRUE or disable for enable. If you enable SQL Trace for a single session, exiting that session also disables SQL Trace.

## Formatting Your Trace Files

```
OS> tkprof tracefile outfile [options]
```

TKPROF command examples:

```
OS> tkprof
OS> tkprof ora_902.trc run1.txt
OS> tkprof ora_902.trc run2.txt sys=no
      sort=execpu print=3
```



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

Use the TKPROF command to format your trace files into readable output. This is the TKPROF syntax:

```
OS> tkprof tracefile outfile [options]
tracefile          Name of the trace output file (input for TKPROF)
outfile           Name of the file to store the formatted results
```

When the TKPROF command is entered without any arguments, it generates a usage message together with a description of all TKPROF options. See the next slide for a full listing. This is the output that you get when you enter the TKPROF command without any arguments:

Usage: tkprof tracefile outfile [explain= ] [table= ]

[print= ] [insert= ] [sys= ] [sort= ]

By default, the .trc file is named after the SPID. You can find the SPID in V\$PROCESS. An easier way of finding the file is the following:

```
SQL> ALTER SESSION SET TRACEFILE_IDENTIFIER = 'MY_FILE';
```

Then the trace file in TKPROF will include the 'MY\_FILE' string.

## TKPROF Command Options

```
SORT = option
PRINT = n
EXPLAIN = username/password
INSERT = filename
SYS = NO
AGGREGATE = NO
RECORD = filename
TABLE = schema.tablename
```



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

The options listed in **bold** are the most commonly used options:

<b>INSERT</b>	Creates a SQL script to load the TKPROF results into a database table
<b>SORT</b>	Order in which to sort the statements in the report (see the next page for a list of values)
<b>PRINT</b>	Produces a report on this (sorted) number of statements only (This option is especially useful in combination with the SORT option.)
<b>EXPLAIN</b>	Logs on and executes EXPLAIN PLAN in the specified schema
<b>SYS</b>	Disables the listing of recursive SQL statements, issued by the user SYS
<b>AGGREGATE</b>	Disables or enables the (default) behavior of TKPROF, aggregating identical SQL statements into one record
<b>WAITS</b>	Specifies whether to record summaries for any wait events found in the trace files
<b>TABLE</b>	Specifies the table to temporarily store execution plans before writing them to the output file (This parameter is ignored if EXPLAIN is not specified. It is useful when several individuals use TKPROF to tune the same schema concurrently, to avoid destructive interference.)
<b>RECORD</b>	Creates a SQL script with all the nonrecursive SQL statements found in the trace file (This script can be used to replay the tuning session later.)

**Sort Options**

prscnt	Number of times parse was called
prscpu	CPU time parsing
prsela	Elapsed time parsing
prsdsk	Number of disk reads during parse
prsqry	Number of buffers for consistent read during parse
prscu	Number of buffers for current read during parse
prsmis	Number of misses in library cache during parse
execnt	Number of executes that were called
execpu	CPU time spent executing
exeela	Elapsed time executing
exedsk	Number of disk reads during execute
exeqry	Number of buffers for consistent read during execute
execu	Number of buffers for current read during execute
exerow	Number of rows processed during execute
exemis	Number of library cache misses during execute
fchcnt	Number of times fetch was called
fchcpu	CPU time spent fetching
fchela	Elapsed time fetching
fchdsk	Number of disk reads during fetch
fchqry	Number of buffers for consistent read during fetch
fchcu	Number of buffers for current read during fetch
fchrow	Number of rows fetched
userid	ID of user who parsed the cursor

## Output of the TKPROF Command

- Text of the SQL statement
- Trace statistics (for statement and recursive calls) separated into three SQL processing steps:

<b>PARSE</b>	Translates the SQL statement into an execution plan
<b>EXECUTE</b>	Executes the statement (This step modifies the data for <code>INSERT</code> , <code>UPDATE</code> , and <code>DELETE</code> statements.)
<b>FETCH</b>	Retrieves the rows returned by a query (Fetches are performed only for <code>SELECT</code> statements.)



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

The TKPROF output lists the statistics for a SQL statement by the SQL processing step. The step for each row that contains statistics is identified by the value of the call column.

- |         |   |
|---------|---|
| PARSE   | This step translates the SQL statement into an execution plan and includes checks for proper security authorization and checks for the existence of tables, columns, and other referenced objects.  |
| EXECUTE | This step is the actual execution of the statement by the Oracle Server. For <code>INSERT</code> , <code>UPDATE</code> , and <code>DELETE</code> statements, this step modifies the data (including sorts when needed). For <code>SELECT</code> statements, this step identifies the selected rows. |
| FETCH   | This step retrieves rows returned by a query and sorts them when needed. Fetches are performed only for <code>SELECT</code> statements.   |

**Note:** The PARSE value includes both *hard* and *soft* parses. A hard parse refers to the development of the execution plan (including optimization); it is subsequently stored in the library cache. A soft parse means that a SQL statement is sent for parsing to the database, but the database finds it in the library cache and only needs to verify things such as access rights. Hard parses can be expensive, particularly due to the optimization. A soft parse is mostly expensive in terms of library cache activity.

## Output of the TKPROF Command

There are seven categories of trace statistics:

<b>Count</b>	Number of times the procedure was executed
<b>CPU</b>	Number of seconds to process
<b>Elapsed</b>	Total number of seconds to execute
<b>Disk</b>	Number of physical blocks read
<b>Query</b>	Number of logical buffers read for consistent read
<b>Current</b>	Number of logical buffers read in current mode
<b>Rows</b>	Number of rows processed by the fetch or execute



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

The output is explained on the following page.

Sample output is as follows:

```
SQL ID : 6assxhyzbq5jf
select max(cust_credit_limit)
from customers where cust_city ='Paris'
call      count      cpu   elapsed      disk      query      current          rows
-----  -----  -----  -----  -----  -----  -----  -----
Parse        1    0.00     0.02          0          0          0          0          0
Execute      1    0.00     0.00          0          0          0          0          0
Fetch        2    0.01     0.26     1455     1457          0          1          1
-----  -----  -----  -----  -----  -----  -----  -----
total       4    0.02     0.28     1455     1457          0          1          1
```

Next to the CALL column, TKPROF displays the following statistics for each statement:

Count	Number of times a statement was parsed, executed, or fetched (Check this column for values greater than 1 before interpreting the statistics in the other columns. Unless the AGGREGATE = NO option is used, TKPROF aggregates identical statement executions into one summary table.)
CPU	Total CPU time in seconds for all parse, execute, or fetch calls
Elapsed	Total elapsed time in seconds for all parse, execute, or fetch calls
Disk	Total number of data blocks physically read from the data files on disk for all parse, execute, or fetch calls
Query	Total number of buffers retrieved in consistent mode for all parse, execute, or fetch calls (Buffers are usually retrieved in consistent mode for queries.)
Current	Total number of buffers retrieved in current mode (Buffers typically are retrieved in current mode for DML statements. However, segment header blocks are always retrieved in current mode.)
Rows	Total number of rows processed by the SQL statement (This total does not include rows processed by subqueries of the SQL statement. For SELECT statements, the number of rows returned appears for the fetch step. For UPDATE, DELETE, and INSERT statements, the number of rows processed appears for the execute step.)

### Note

- DISK is equivalent to physical reads from v\$sysstat or AUTOTRACE.
- QUERY is equivalent to consistent gets from v\$sysstat or AUTOTRACE.
- CURRENT is equivalent to db block gets from v\$sysstat or AUTOTRACE.

## Output of the TKPROF Command

The TKPROF output also includes the following:

- Recursive SQL statements
- Library cache misses
- Parsing user ID
- Execution plan
- Optimizer mode or hint
- Row source operation

```
...
Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 85  (SH)

Rows  Row Source Operation
-----
 1  SORT AGGREGATE (cr=1457 pr=1455 pw=1455 time=0 us)
 77   TABLE ACCESS FULL CUSTOMERS (cr=1457 pr=1455 pw=1455 time=3338
...
...
```



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

To execute a SQL statement issued by a user, the Oracle server must occasionally issue additional statements. Such statements are called *recursive SQL statements*. For example, if you insert a row in a table that does not have enough space to hold that row, the Oracle server makes recursive calls to allocate the space dynamically. Recursive calls are also generated when data dictionary information is not available in the data dictionary cache and must be retrieved from disk.

If recursive calls occur while the SQL Trace facility is enabled, TKPROF marks them clearly as recursive SQL statements in the output file. You can suppress the listing of recursive calls in the output file by setting the `SYS=NO` command-line parameter. Note that the statistics for recursive SQL statements are always included in the listing for the SQL statement that caused the recursive call.

### Library Cache Misses

TKPROF also lists the number of library cache misses resulting from parse and execute steps for each SQL statement. These statistics appear on separate lines following the tabular statistics.

## Recursive Calls

### Parsing User ID

This is the ID of the last user to parse the statement.

### Row Source Operation

The row source operation shows the data sources for execution of the SQL statement. This section provides the number of rows processed for each operation executed on the rows and additional row source information, such as physical reads and writes; cr = consistent reads, pw = physical writes, pr = physical reads, time = time (in microseconds), cost = estimated cost, size = estimates bytes of row source, card=cardinality(number of rows).

### Execution Plan

If you specify the EXPLAIN parameter on the TKPROF command line, TKPROF uses the EXPLAIN PLAN command to generate the execution plan of each SQL statement traced. TKPROF also displays the number of rows processed by each step of the execution plan.

**Note:** Be aware that the execution plan is generated *at the time that the TKPROF command is run* and not at the time the trace file was produced. This could make a difference if, for example, an index has been created or dropped since tracing the statements.

### Optimizer Mode or Hint

This indicates the optimizer hint that is used during the execution of the statement. If there is no hint, then it shows the optimizer mode that is used.

## TKPROF Output with No Index: Example

```
...
select max(cust_credit_limit)
from customers where cust_city ='Paris'

call      count      cpu    elapsed      disk      query      current      rows
-----
Parse        1      0.00      0.00          0          0          0          0
Execute      1      0.00      0.00          0          0          0          0
Fetch        2      0.03      0.03     1455      1459          0          1
-----
total       4      0.04      0.04     1455      1459          0          1

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 85  (SH)

Rows      Row Source Operation
-----
   1  SORT AGGREGATE (cr=1459 pr=1455 pw=1455 time=0 us)
  77  TABLE ACCESS FULL CUSTOMERS (cr=1459 pr=1455 pw=1455 time=170
                                             us cost=406 size=1260 card=90)
...

```



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

The example in the slide shows that the aggregation of results across several executions (rows) is being fetched from the CUSTOMERS table. It requires .12 second of CPU fetch time. The statement is executed through a full table scan of the CUSTOMERS table, as you can see in the row source operation of the output.

The statement must be optimized.

**Note:** If CPU or elapsed values are 0, then timed\_statistics is not set.

## TKPROF Output with Index: Example

```
...
select max(cust_credit_limit)
from customers where cust_city = 'Paris'

call      count      cpu      elapsed      disk      query      current      rows
-----
Parse        1    0.00      0.00          0          0          0          0
Execute      1    0.00      0.00          0          0          0          0
Fetch        2    0.00      0.00         77          77          0          1
-----
total       4    0.01      0.01         77          77          0          1

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 85 (SH)

Rows      Row Source Operation
-----
1  SORT AGGREGATE (cr=77 pr=77 pw=77 time=0 us)
77   TABLE ACCESS BY INDEX ROWID CUSTOMERS (cr=77 pr=77 pw=77
               time=555 us cost=85 size=1260 card=90)
77   INDEX RANGE SCAN CUST_CUST_CITY_IDX (cr=2 pr=2 pw=2
               time=1 us cost=1 size=0 card=90) (object id 75264)
```



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

The results shown in the slide indicate that CPU time was reduced to .01 second when an index was created on the CUST\_CITY column. These results may have been achieved because the statement uses the index to retrieve the data. Additionally, because this example is reexecuting the same statement, most of the data blocks are already in memory. You can achieve significant improvements in performance by indexing sensibly. Identify areas for potential improvement using the SQL Trace facility.

**Note:** Indexes should not be built unless required. Indexes do slow down the processing of INSERT, UPDATE, and DELETE commands because references to rows must be added, changed, or removed. Unused indexes should be removed. However, instead of processing all of the application SQL through EXPLAIN PLAN, you can use index monitoring to identify and remove any indexes that are not used or use the SQL Access Advisor to identify unused indexes.

## Generate an Optimizer Trace

Set an event—10053 optimizer trace.

```
SQL> ALTER SESSION SET EVENTS
2> '10053 trace name context forever, level 1';
```

Execute the statement of interest.

```
SQL> select *
2> from hr.employees natural join hr.departments
3> where department_id = 10;
```

Find and view the trace file.



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

The optimizer can be commanded to produce a trace of the costing decisions it makes with the command. This is occasionally used to provide Oracle Support with additional information about the optimizer actions.

```
ALTER SESSION SET EVENTS
'10053 trace name context forever, level 1';
```

The location of the optimizer trace in the same location as the other trace files, and the name of the trace file, can be modified with the command:

```
ALTER SESSION SET TRACEFILE_IDENTIFIER='opt';
```

The trace file does not require formatting, but it is quite large; be sure to increase the size allowed for the trace in the session with:

```
ALTER SESSION SET MAX_DUMPFILE_SIZE=UNLIMITED;
```

Stop the tracing in your session by exiting the session or the command:

```
ALTER SESSION SET EVENTS
'10053 trace name context off';
```

The trace file contains a large amount of information that is not easy to read. For more information, see My Oracle Support Document 338137.1 *CASE STUDY: Analyzing 10053 Trace Files*.

## Quiz

The optimizer creates an execution plan that performs the action requested by the SQL statement. The execution plan shows you the access path and associated cost estimates. Identify the tools that can display the execution plan.

- a. SQL Tuning Advisor
- b. The EXPLAIN PLAN command
- c. SQL Profiles
- d. The DBMS\_XPLAN.DISPLAY procedure
- e. SQL Plus AUTOTRACE



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

### Answer: a, d, and e

The SQL Tuning advisor, the DBMS\_XPLAN.DISPLAY procedure, and SQL Plus AUTOTRACE will show an execution plan. SQL Profiles are objects in the database that are used by the optimizer to produce a better execution plan. The EXPLAIN PLAN command generates an execution plan and places it in a plan table, but you have to use SQL or DBMS\_XPLAN.DISPLAY to display the plan.

## Summary

In this lesson, you should have learned how to:

- Describe SQL statement processing
- Describe the role of the optimizer
- View the SQL statement statistics
- Identify the SQL statements that perform poorly
- Generate and view an explain plan
- Generate a TKPROF report
- Generate an optimizer trace



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

## **Practice Overview 10: Using Execution Plan Utilities**

This practice covers the following topics:

- Using AUTOTRACE
- Using EXPLAIN PLAN
- Using AWR
- Retrieving the execution plan using DBMS\_XPLAN
- Using TKPROF
- Generating an optimizer trace



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

# **Influencing the Optimizer**

11

**ORACLE®**

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

# Objectives

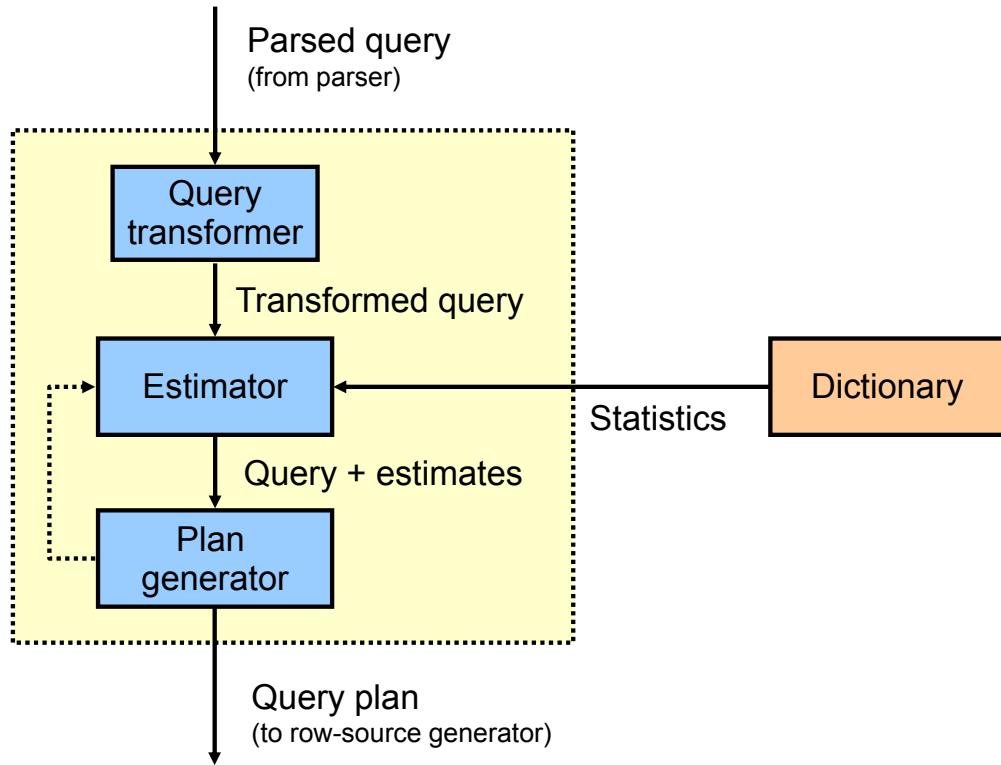
After completing this lesson, you should be able to do the following:

- Describe the optimizer's behavior
- Explain how statistics can affect the optimizer
- Describe how data structures affect the optimizer
- Adjust parameters to influence the optimizer



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

## Functions of the Query Optimizer



**ORACLE**

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

The optimizer (also known as “query optimizer”) performs its tasks during the parse phase of SQL processing. Most of these tasks are performed only during a hard parse because the optimizer output, the execution plan, is stored with the cursor in the shared pool. Optimizer operations include:

- Transforming queries
- Estimating
- Generating plans

The illustration in the slide depicts a query entering the query transformer. The transformed query is then sent to the estimator. Statistics are retrieved from the dictionary, and the query and estimates are sent to the plan generator. The plan generator either returns the plan to the estimator for comparison with other plans or sends the query plan to the row-source generator.

The optimizer tests various access paths, join orders, and join methods to find the best plan. The optimizer can be influenced by SQL statement structure, statistics, data structures (indexes, table types), and parameters. These influences determine the order in which the various combinations are tried. Because the run-time optimizer runs for a limited time, it might not test all combinations.

## Transforming Queries

The input to the query transformer is a parsed query, which is represented by a set of query blocks. A query block can be defined as a complete query, nested subquery, or nonmerged view. The query blocks are nested or interrelated to each other. The form of the query determines how the query blocks are interrelated to each other. The main objective of the query transformer is to determine if it is advantageous to change the form of the query so that it enables generation of a better query plan.

## Estimating

The estimator generates three types of measures:

- Selectivity
- Cardinality
- Cost

These measures are related to each other, and one is derived from another. The end goal of the estimator is to estimate the overall cost of a given plan. If statistics are available, the estimator uses them to compute the measures. The optimizer will calculate approximate costs of each candidate execution plan based on statistics available to it. In the absence of statistics, the optimizer may perform dynamic sampling, or fall back to hard-coded defaults. The quality of the statistics are directly related to the accuracy of the cost estimated for the plan.

## Generating Plans

The main function of the plan generator is to test different possible plans for a given query and pick the one that has the lowest cost. Many different plans are possible because of the various combinations of different access paths, join methods, and join orders that can be used to access and process data in different ways and produce the same result.

The plan for a query is established by first generating subplans for each of the nested subqueries and nonmerged views. Each nested subquery or nonmerged view is represented by a separate query block. The query blocks are optimized separately in a bottom-to-top order. That is, the innermost query block is optimized first, and a subplan is generated for it. The outermost query block, which represents the entire query, is optimized last.

The plan generator explores various plans for a query block by testing different access paths, join methods, and join orders. The number of possible plans for a query block is proportional to the number of join items in the `FROM` clause. This number rises factorially with the number of join items. The plan generator uses an internal cutoff to reduce the number of plans it tries when finding the one with the lowest cost. The cutoff is based on the cost of the current best plan. If the current best cost is large, the plan generator tries harder (in other words, explores more alternate plans) to find a better plan with a lower cost. If the current best cost is small, the plan generator ends the search swiftly, because further cost improvement will not be significant. The cutoff works well if the plan generator starts with an initial join order that produces a plan with a cost that is close to optimal.

**Note:** The primary difference between the query optimizer and the Automatic Tuning Optimizer is the time allowed to check alternate plans. The query optimizer is limited in the number of plans it is allowed to check.

# Selectivity

- Selectivity represents a fraction of rows from a row source.
  - Selectivity affects the estimates of I/O cost.
  - Selectivity affects the sort cost.
- Selectivity lies in a value range from 0.0 through 1.0.
- When statistics are available, the estimator uses them to estimate selectivity.
- When statistics are not available, the estimator uses default values or dynamic sampling.
- With histograms on columns that contain skewed data, the results are good selectivity estimates.



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

Selectivity times the number of rows in a row source represents a fraction of rows returned from a row source. The row source can be a base table, a view, or the result of a join or a GROUP BY operator. The selectivity depends on the predicate, such as `last_name = 'Smith'`, or a combination of predicates, such as `last_name = 'Smith' AND salary > 2000`. The selectivity of a predicate indicates how many rows from a row source will pass the predicate test.

$$\text{Selectivity} = \frac{\text{\# rows satisfying a condition}}{\text{Total \# of rows}}$$

When statistics are available, the estimator uses them to estimate selectivity. For example, for an equality predicate (`last_name = 'Smith'`), selectivity is set to the reciprocal of the number of  $n$  distinct values of `last_name`, because the query selects rows that all contain one out of  $n$  distinct values. If a histogram is available on the `LAST_NAME` column, the estimator uses it instead of the number of distinct values. If no statistics are available, the optimizer uses either dynamic sampling or an internal default value, depending on the value of the `OPTIMIZER_DYNAMIC_SAMPLING` initialization parameter. Assumption of a default value can result in a less-than-optimal plan being used.

## Cardinality and Cost

- Cardinality represents the number of rows in a row source.
- Cost represents the units of work or resource that are used.



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

**Cardinality** represents the number of rows in a row source. Here, the row source can be a base table, a view, or the result of a join or GROUP BY operator. If a select from a table is performed, the table is the row source and the cardinality is the number of rows in that table.

**Note:** Not all of the possible row sources are considered here.

**Cost** represents the number of units of work (or resource) that are used. The query optimizer uses disk I/O, CPU usage, and memory usage as units of work. So the cost used by the query optimizer represents an estimate of the number of disk I/Os and the amount of CPU and memory used in performing an operation. The operation can be scanning a table, accessing rows from a table by using an index, joining two tables together, or sorting a row source. The cost of a query plan is the number of work units that are expected to be incurred when the query is executed and its result is produced.

The access path determines the number of units of work that are required to get data from a base table. The access path can be a table scan, a fast full index scan, or an index scan.

The cost of various resources is normalized to a unit corresponding to the cost of a single block read. Then the costs are compared for various execution plans. The work units are a normalized estimate not a physical measurement, but the work units are proportional to time.

# Changing Optimizer Behavior

The optimizer is influenced by:

- SQL statement construction
- Data structure
- Statistics
- SQL Plan Management options
- Session parameters
- System parameters
- Hints



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

Ideally the optimizer would take any correct SQL statement and produce the same optimal execution plan. The optimizer starts with the construction of the SQL statement as a guide to choose access paths. Even though a SQL statement could achieve the same result whether it is written as an intersection, an outer join, or a correlated subquery, each form of this statement will cause the access paths to be evaluated in a different order, and possibly result in a different explain plan.

The next major guide to the optimizer is data structure. Optimizer behavior is always a set of decisions based on relative cost. Some access paths are more efficient than others. The available access paths depend heavily on the data structure. Adding or removing indexes, and changing from heap organized tables to index-organized tables are examples of changes in data structure.

Statistics play a very important role in the optimizer choices. If the statistics do not reflect the current state of the data objects, the optimizer cannot make accurate choices. Though sometimes updating the statistics can cause some statements to perform worse (regress).

Up to this point, the influences require very little intervention by the DBA. Statements are in the code, data structure is relatively fixed, and statistics are based on the size of the objects and can be updated automatically as needed. The rest of these influences are ways to change the way the optimizer behaves and require different levels of DBA intervention. They are arranged in order of preference.

SQL Plan Management options are discussed in the lesson titled “SQL Performance Management.”

The system parameters, usually set in the initialization parameter file, control several of the factors used in the optimizer cost calculations. The danger of using system parameters to tune SQL is that while setting a particular parameter can improve the performance of a set of SQL statements, it could also hurt the overall performance of the application or instance.

System parameters apply to the entire instance including the recursive SQL the instance does on your behalf. Often system parameters will be set to different values in a session while testing to compare the performance before the parameter is set at the system level.

Hints applied to the SQL statement are suggestions to the optimizer, and affect the choices the optimizer makes. As a DBA you are seldom allowed to change the SQL code, therefore changing the statement construction or adding hints are usually not available to you. Hints create a maintenance problem, that is, as hardware and software change, the hints may no longer be valid and changes in the data could cause the hint to produce a less efficient statement than the optimizer. Hints should only be used as development tests. Hints are NOT recommended for production deployment.

# Optimizer Statistics

The optimizer depends on various statistics to determine an optimal execution plan for a SQL statement.

- Most statistics are gathered automatically and are controlled by statistic preferences.
  - Object statistics
  - Dictionary statistics
- Other statistics are only gathered manually.
  - Statistics on fixed objects
  - Operating system statistics



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

The optimizer depends on the object statistics related to the objects referenced by the SQL statement and their subobjects and indexes. The dictionary statistics are important for optimizing recursive SQL. Both of these statistics sets are monitored and refreshed by the Optimizer Statistics Gathering automatic maintenance task. Typically, the dictionary statistics do not change rapidly. Both of these statistic sets may also be refreshed manually.

The statistics on fixed objects and the operating system statistics must be collected manually. The statistics on fixed objects, do not change much unless the workload changes, so these can be collected during a typical workload period. A change in workload can occur when deploying a new application or when adding many new users. Object, dictionary, and fixed object statistics may be collected manually either from a job scheduled from the Enterprise Manager Gather Optimizer Statistics page or with procedures in the DBMS\_STATS package.

The operating system statistics are collected with either manual start and end times, or for a duration. These statistics may also be gathered with the WORKLOAD or NOWORKLOAD options. The operating system statistics should be gathered periodically by using the WORKLOAD option while a typical workload is running. Use the GATHER\_SYSTEM\_STATS procedure in the DBMS\_STATS package. The NOWORKLOAD statistics are collected by default when the instance is started. For more details, see *Oracle Database PL/SQL Packages and Types Reference*.

# Extended Statistics

The optimizer needs additional statistics in some cases.

- Multicolumn statistics for related columns (column groups)
  - Collect statistics over a group of columns
  - For complex predicates
  - For equality predicates only
- Expression statistics for function-based indexes
  - Collect statistics over an expression
  - For expressions used in predicates
  - Creates a virtual column



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

The optimizer cannot calculate the selectivity for related columns without statistics over all the columns involved. You can create a column group to provide these statistics to the optimizer. However, a column group works only for equality predicates. Column group statistics enable you to collect, store, and use the following statistics to capture functional dependency between two or more columns: number of distinct values, number of nulls, frequency histograms, and density. When a function is used to transform a column value for use in a predicate, the statistics for the transformed column can be very different. You can create expression statistics for the expression. The expression statistics may be created with or without a function-based index. Statistics for function based-indexes are collected by default.

Both of the extended statistics methods create a virtual column for the table and then statistics can be gathered on that column. The `DBMS_STATS.CREATE_EXTENDED_STATS` function is used to create the virtual column, and then the `GATHER_TABLE_STATS` procedure has syntax to collect statistics on the virtual column. For more information about the procedures used to create and maintain extended statistics, see the `DBMS_STATS` section in the *Oracle Database PL/SQL Packages and Types Reference* and the *Oracle Database Performance Tuning Guide*.

# Optimizer Parameters

Optimizer parameters can be set at:

- The system level
- The session level

You can display the optimizer parameter settings for:

- The system with V\$SYS\_OPTIMIZER\_ENV
- Sessions with V\$SES\_OPTIMIZER\_ENV
- A specific plan by using V\$SQL\_OPTIMIZER\_ENV joined with V\$SQL or V\$SQLAREA



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

The optimizer is affected by parameters set in the environment. Some parameters directly affect the optimizer behavior and others indirectly. All of these parameters may be set at the system (instance) level and some parameters may be set at the session level.

The current system parameter settings may be displayed by using V\$SYS\_OPTIMIZER\_ENV, and the session level parameters with V\$SES\_OPTIMIZER\_ENV.

The optimizer parameters that are used to build a particular SQL Plan can be displayed using V\$SQL\_OPTIMIZER\_ENV joined with V\$SQL on (HASH\_VALUE, CHILD\_ADDRESS) or V\$SQLAREA on (HASH\_VALUE, ADDRESS).

# Controlling the Behavior of the Optimizer with Parameters

Optimizer behavior can be controlled by using the following initialization parameters:

- CURSOR\_SHARING
- DB\_FILE\_MULTIBLOCK\_READ\_COUNT (autotuned)
- OPTIMIZER\_INDEX\_CACHING
- OPTIMIZER\_INDEX\_COST\_ADJ
- PGA\_AGGREGATE\_TARGET
- OPTIMIZER\_MODE
- OPTIMIZER\_FEATURES\_ENABLE
- OPTIMIZER\_ADAPTIVE\_FEATURES
- OPTIMIZER\_ADAPTIVE\_REPORTING\_ONLY



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

- **CURSOR\_SHARING**: Converts literal values in SQL statements to bind variables. Converting the values improves cursor sharing and can affect the execution plans of SQL statements. The optimizer generates the execution plan based on the presence of the bind variables and not on the literal values. For more details on this topic, see the lesson titled “Tuning the Shared Pool.”
- **DB\_FILE\_MULTIBLOCK\_READ\_COUNT**: Specifies the number of blocks that are read in a single I/O during a full table scan or an index fast-full scan. The optimizer uses the explicitly specified value of DB\_FILE\_MULTIBLOCK\_READ\_COUNT to estimate the cost of full table scans and index fast-full scans. Larger values result in a lower estimated cost for full table scans and can result in the optimizer choosing a full table scan over an index scan. This parameter is automatically tuned and set at instance startup if not explicitly specified. The automatic value is used for determining the read request size for full table scans and index fast-full scans. When the automatic value is set, the optimizer uses a value of 8. Even if the default value is large, the optimizer will not favor plans with full table scans if you do not explicitly set this parameter.

- **OPTIMIZER\_INDEX\_CACHING:** Controls the costing of an index probe in conjunction with a nested loop. The range of values *0* through *100* for **OPTIMIZER\_INDEX\_CACHING** indicates the percentage of index blocks in the buffer cache; this modifies the optimizer's assumptions about index caching for nested loops and IN-list iterators. A value of *100* implies that 100% of the index blocks are likely to be found in the buffer cache, and the optimizer adjusts the cost of an index probe or nested loop accordingly. Use caution when using this parameter because execution plans can change in favor of using indexes.
- **OPTIMIZER\_INDEX\_COST\_ADJ:** Can be used to adjust the cost of index probes. The range of values is *1* through *10000*. The default value is *100*, which means that indexes are evaluated as an access path based on the normal costing model. A value of *10* means that the cost of an index access path is one-tenth the normal cost of an index access path. Setting this parameter influences the optimizer when considering an index as part of the execution plan.
- **PGA\_AGGREGATE\_TARGET:** Automatically controls the amount of memory allocated for sorts and hash joins when the **WORKAREA\_SIZE\_POLICY** parameter is set to **AUTO**. Allocations of larger amounts of memory for sorts or hash joins reduce the optimizer cost of these operations. This parameter, if set to **true**, enables the query optimizer to cost a star transformation for star queries. The star transformation combines the bitmap indexes on the various fact table columns.
- **OPTIMIZER\_MODE:** Sets the approach that the optimizer uses for determining the best plan. See the following slides for more details.
- **OPTIMIZER\_FEATURES\_ENABLE:** Acts as an umbrella parameter for enabling a series of optimizer features based on an Oracle release number. The **OPTIMIZER\_FEATURES\_ENABLE** parameter controls a set of optimizer parameters, most of which are underscore parameters. These parameters can be set independently, but doing so is not recommended. These underscore parameters generally enable or disable certain optimizer features. Oracle Support recommends that customers do not set these underscore parameters independently but only through the **OPTIMIZER\_FEATURES\_ENABLE** parameter.
- **OPTIMIZER\_ADAPTIVE\_FEATURES** and **OPTIMIZER\_ADAPTIVE\_REPORTING\_ONLY**: Control the adaptive optimizer feature in Oracle Database 12c and later

## Other Initialization Parameters

The **V\$SES\_OPTIMIZER\_ENV** view lists all the session parameters used by the optimizer. Most of these are hidden parameters. Many are related to parallel query, or materialized views. There are parameters related SQL Plan Management that will be discussed in the lesson titled “SQL Performance Management.” You can view all of the session parameters with the following command:

```
SELECT * FROM V$SES_OPTIMIZER_ENV;
```

# Enabling Query Optimizer Features

- The optimizer behavior can be set to prior releases of the database.
- The `OPTIMIZER_FEATURES_ENABLE` initialization parameter can be set to values of different database releases (such as 11.1.0.7, 11.2.0.1, or 12.1.0.1).
- Example:

```
OPTIMIZER_FEATURES_ENABLE='11.2.0.1';
```



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

The `OPTIMIZER_FEATURES_ENABLE` parameter acts as an umbrella parameter for the query optimizer. This parameter can be used to enable a series of optimizer-related features, depending on the release. It accepts one of a list of valid string values corresponding to the release numbers, such as 10.2.0.5, 11.1.0.7, and 12.1.0.1.

The `OPTIMIZER_FEATURES_ENABLE` parameter enables you to upgrade the Oracle Server, yet preserve the old behavior of the query optimizer after the upgrade. For example, when you upgrade the Oracle Server from release 11.2.0.1 to release 12.1.0.1, the default value of the `OPTIMIZER_FEATURES_ENABLE` parameter changes from 11.2.0.1 to 12.1.0.1.

This upgrade results in the query optimizer enabling optimization features based on 12.1.0.1. For plan stability or backward compatibility reasons, you might not want the query plans to change because of new optimizer features in a new release. In such a case, you can set the `OPTIMIZER_FEATURES_ENABLE` parameter to an earlier version.

For example, the following setting enables the use of the optimizer features in Oracle 11g Release 2:

```
OPTIMIZER_FEATURES_ENABLE='11.2.0.1';
```

## Adaptive Execution Plans

- A query plan changes during execution because runtime conditions indicate that optimizer estimates are inaccurate.
- All adaptive execution plans rely on statistics that are collected during query execution.
- The two adaptive plan techniques are:
  - Dynamic plans
  - Re-optimization
- The database uses adaptive execution plans when `OPTIMIZER_FEATURES_ENABLE` is set to 12.1.0.1 or later, and the `OPTIMIZER_ADAPTIVE_REPORTING_ONLY` initialization parameter is set to the default FALSE.



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

The Adaptive Execution Plans feature enables the optimizer to automatically adapt a poorly performing execution plan at run time and prevent a poor plan from being chosen on subsequent executions. With an adaptive plan, the optimizer instruments its chosen plan so that at run time, it can detect if the estimates are not optimal, and the plan can change to automatically adapt to the actual conditions.

The optimizer can adapt plans based on statistics that are collected during statement execution. All adaptive mechanisms can execute a plan that differs from the plan which was originally determined during hard parse. This improves the ability of the query-processing engine (compilation and execution) to generate better execution plans.

The two Adaptive Plan techniques are:

- **Dynamic plans:** A dynamic plan chooses among subplans during statement execution. For dynamic plans, the optimizer must decide which subplans to include in a dynamic plan, which statistics to collect to choose a subplan, and thresholds for this choice.
- **Re-optimization:** In contrast, re-optimization changes a plan for executions after the current execution. For re-optimization, the optimizer must decide which statistics to collect at which points in a plan and when re-optimization is feasible.

**Note:** `OPTIMIZER_ADAPTIVE_REPORTING_ONLY` controls reporting-only mode for adaptive optimizations. When set to TRUE, adaptive optimizations gather the information required for an adaptive optimization, but no action is taken to change the plan.

## Dynamic Plans

- The final decision is based on statistics that are collected during execution.
- Alternate subplans are precomputed and stored in the cursor.
- Statistic collectors are inserted at key points in the plan.
- If statistics prove to be out of range, subplans can be swapped.
- It requires buffering near the swap point to avoid returning rows to users.
- Only join methods and the distribution method can change.



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

A dynamic plan is an execution plan that has multiple built-in plan options. During the first execution, before a specific subplan becomes active, the optimizer makes a final decision about which option to use. The optimizer bases its choice on observations made during the execution up to this point. A dynamic plan can enable a different final plan for a statement from the default plan, thereby potentially improving query performance.

A subplan is a portion of a plan that the optimizer can switch to as an alternative at run time.

Portions of the plan preceding the statistics collector can have alternate subplans, each of which is valid for a subset of possible values returned by the collector. During statement execution, the statistics collector buffers a portion of rows, and then the subplan is chosen.

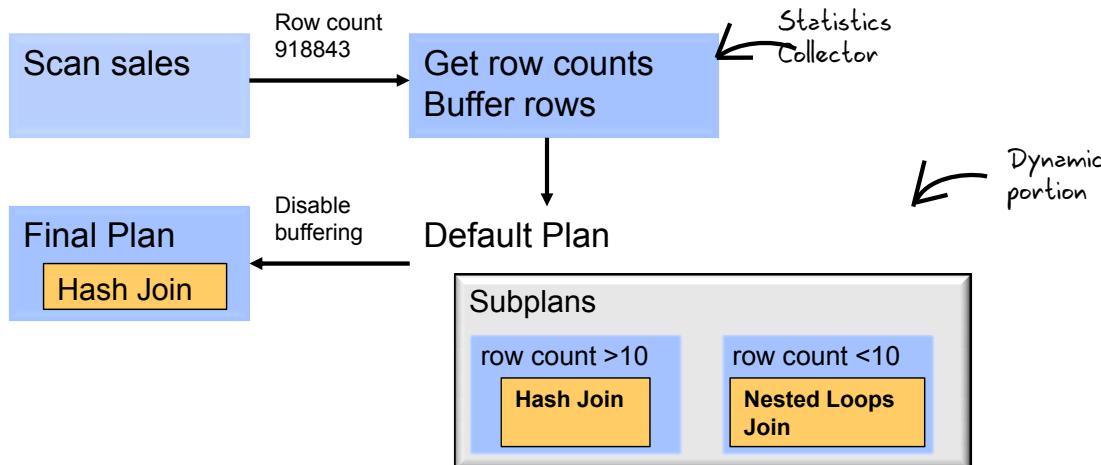
Often the set of values for a subplan is a range. If a statistic falls into the valid values of a subplan that was not the default plan, then the optimizer chooses the alternative subplan. After a subplan is chosen, buffering is disabled. The statistics collector stops collecting rows, and passes them through instead. On subsequent executions of the child cursor, the optimizer disables buffering, and chooses the same final plan.

With dynamic plans, the execution plan changes the optimizer's estimated plan choices, and correct decisions can be made during the first execution.

**Note:** The new column in `v$SQL IS_RESOLVED_DYNAMIC_PLAN` indicates whether the final plan is the default plan or not. Information found via dynamic plans is persisted as SQL plan directives.

## Dynamic Plan: Adaptive Process

```
SQL> SELECT s.cust_id, c.cust_last_name
  2  FROM sh.sales s, sh.customers c
  3 WHERE s.cust_id = c.cust_id;
```



ORACLE

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

The code example shows a join of the sales and customers tables. A dynamic plan for this statement could show two possible plans: one with a nested loop and the other with a hash join.

The graphic in the slide shows the adaptive process. For the query given in the slide, the dynamic portion of the default plan contains two subplans, each of which uses a different join method. The collector retrieves row count statistics, and also buffers up to 10 rows.

If the row count is below 10, the optimizer chooses the nested loop; otherwise, the optimizer chooses the hash join. Because the row count in the sales table is over 10, the optimizer chooses a hash join, stops buffering, and makes the final plan.

## Dynamic Plans: Example

A.1. The default plan show NLJ		A.2. The final plan shows HJ																																					
<pre>SQL&gt; explain plan for select /*Q10*/ product_name from order_items o, product_information p where o.unit_price = 15   and quantity &gt; 1   and p.product_id = o.product_id;  Explained.</pre>		<pre>SQL&gt; select /*Q10*/ product_name from order_items o, product_information p where o.unit_price = 15   and quantity &gt; 1   and p.product_id = o.product_id;  PRODUCT_NAME ----- Screws &lt;5.25.S&gt; 13 rows selected.  SQL&gt; select * from table(dbms_xplan.display_cursor(format=&gt;'basic +note'));  PLAN_TABLE_OUTPUT</pre>																																					
<pre>Plan hash value: 384646879</pre>		<pre>EXPLAINED SQL STATEMENT: select /*Q10*/ product_name from order_items o, product_information p where o.unit_price = 15 and quantity &gt; 1 and p.product_id = o.product_id  Plan hash value: 2615131494</pre>																																					
<table border="1"> <thead> <tr> <th>Id</th> <th>Operation</th> <th>Name</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>SELECT STATEMENT</td> <td></td> </tr> <tr> <td>1</td> <td>NESTED LOOPS</td> <td></td> </tr> <tr> <td>2</td> <td>NESTED LOOPS</td> <td></td> </tr> <tr> <td>3</td> <td>TABLE ACCESS FULL</td> <td>ORDER_ITEMS</td> </tr> <tr> <td>4</td> <td>INDEX UNIQUE SCAN</td> <td>PRODUCT_INFORMATION_PK</td> </tr> <tr> <td>5</td> <td>TABLE ACCESS BY INDEX ROWID</td> <td>PRODUCT_INFORMATION</td> </tr> </tbody> </table>		Id	Operation	Name	0	SELECT STATEMENT		1	NESTED LOOPS		2	NESTED LOOPS		3	TABLE ACCESS FULL	ORDER_ITEMS	4	INDEX UNIQUE SCAN	PRODUCT_INFORMATION_PK	5	TABLE ACCESS BY INDEX ROWID	PRODUCT_INFORMATION	<table border="1"> <thead> <tr> <th>Id</th> <th>Operation</th> <th>Name</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>SELECT STATEMENT</td> <td></td> </tr> <tr> <td>1</td> <td>HASH JOIN</td> <td></td> </tr> <tr> <td>2</td> <td>TABLE ACCESS FULL</td> <td>ORDER_ITEMS</td> </tr> <tr> <td>3</td> <td>TABLE ACCESS FULL</td> <td>PRODUCT_INFORMATION</td> </tr> </tbody> </table>		Id	Operation	Name	0	SELECT STATEMENT		1	HASH JOIN		2	TABLE ACCESS FULL	ORDER_ITEMS	3	TABLE ACCESS FULL	PRODUCT_INFORMATION
Id	Operation	Name																																					
0	SELECT STATEMENT																																						
1	NESTED LOOPS																																						
2	NESTED LOOPS																																						
3	TABLE ACCESS FULL	ORDER_ITEMS																																					
4	INDEX UNIQUE SCAN	PRODUCT_INFORMATION_PK																																					
5	TABLE ACCESS BY INDEX ROWID	PRODUCT_INFORMATION																																					
Id	Operation	Name																																					
0	SELECT STATEMENT																																						
1	HASH JOIN																																						
2	TABLE ACCESS FULL	ORDER_ITEMS																																					
3	TABLE ACCESS FULL	PRODUCT_INFORMATION																																					

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

A statement that has a dynamic plan could show two possible plans: one plan with a nested loop before execution and another with a hash join after execution. This information is displayed in the plan output table.

The Note section of the execution plan indicates whether the plan is dynamic: This is a dynamic plan.

## Reoptimization: Cardinality Feedback

- Cardinality feedback was introduced in Oracle Database 11g, Release 2.
- Cardinality feedback is useful for queries where the data volume being processed is stable over time.
- During query execution, optimizer estimates are compared to execution statistic. If they vary significantly, a new plan will be chosen for subsequent executions.

**Note:** The `CURSOR_SHARING` parameter is either `EXACT` or `FORCE`.



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

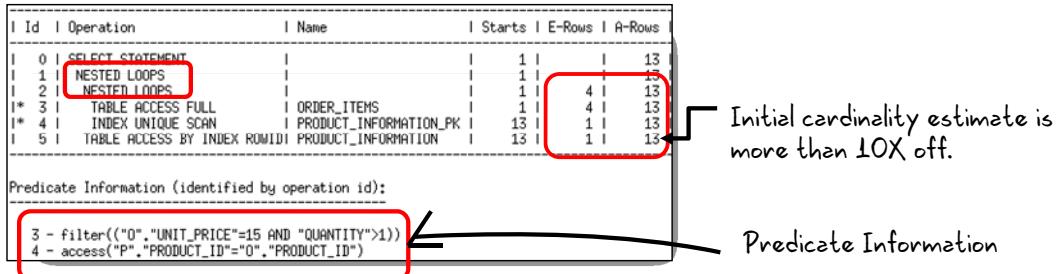
Cardinality feedback was introduced in Oracle Database 11g, Release 2. This feature does a very limited form of re-optimization, after the first execution is completed. It automatically improves plans for queries for which the optimizer estimates improper cardinalities in the plan. The optimizer misestimates cardinalities for a variety of reasons, such as missing or inaccurate statistics, or complex predicates. Cardinality feedback is not used for volatile tables. It is useful for queries where the data volume being processed is stable over time.

The optimizer may enable cardinality monitoring feedback for the cursor in the following cases: tables with no statistics, multiple conjunctive or disjunctive filter predicates on a table, and predicates containing complex operators for which the optimizer cannot accurately compute selectivity estimates.

The feature works as follows: During query execution, optimizer estimates are compared to execution statistics. If they vary significantly, a new plan is chosen for subsequent executions. Statistics are gathered for the actual data volume and data type seen during execution. If the original optimizer estimates vary significantly, the statement is hard parsed during the next execution by using the execution statistics. Statements are monitored only once. If they do not show significant differences initially, they do not change in the future. Only individual table cardinalities are examined. Information is stored only in the cursor and is lost if the cursor ages out.

# Cardinality Feedback: Monitoring Query Executions

```
SQL> SELECT /*+ gather_plan_statistics */ product_name
  2  FROM    order_items o, product_information p
  3  WHERE   o.unit_price = 15
  4  AND     quantity > 1
  5  AND     p.product_id = o.product_id;
```



ORACLE

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

Execution of the query given in the code box is monitored. The query has a combination of an equality filter predicate and an inequality filter predicate, which causes the misestimation in the cardinality of the `order_items` table after these filters are applied. The optimizer chooses the given plan where you can see from the execution statistics (estimation: E-Rows and actual: A-Rows columns in the plan) that the cardinality of `order_items` is underestimated.

# Cardinality Feedback: Reparsing Statements

```
SQL> SELECT /*+ gather_plan_statistics */ product_name
  2   FROM order_items o, product_information p
  3 WHERE o.unit_price = 15
  4 AND quantity > 1
  5 AND p.product_id = o.product_id;
```

I	Id	Operation	I	Name	I	Starts	E-Rows	I	A-Rows
I	0	SELECT STATEMENT	I		I	1		I	13
I*	1	HASH JOIN	I		I	1	13	I	13
I*	2	TABLE ACCESS FULL	I	ORDER_ITEMS	I	1	13	I	13
I	3	TABLE ACCESS FULL	I	PRODUCT_INFORMATION	I	1	288	I	288

Predicate Information (identified by operation id):

```
1 - access("P"."PRODUCT_ID"="O"."PRODUCT_ID")
2 - filter(("O"."UNIT_PRICE"=15 AND "QUANTITY">>1))
```

Note

```
- cardinality feedback used for this statement
```

Cardinality estimates are now correct and trigger the join method to change.

Cardinality feedback appears in the notes section of the plan.

ORACLE®

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

Execution statistics are used to reparse the statement during the second execution of the query in the code box. During the second execution, the actual cardinality is fed back to the optimizer.

This time the optimizer gets the cardinality estimate right, and the optimizer displays a different plan.

## Automatic Re-optimization

- The optimizer automatically changes a plan during subsequent executions of a SQL statement.
- Join statistics are also included.
- Statements are continually monitored to see if statistics fluctuate over different executions.
- It works with adaptive cursor sharing for statements with bind variables.
- A new column is added in `v$SQL_IS_REOPTIMIZABLE`.
- Information found at execution time is persisted as SQL plan directives.



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

In re-optimization, the optimizer adapts a plan only on subsequent statement executions. Deferred optimization can fix any suboptimal plan chosen because of incorrect optimizer estimates, such as a suboptimal distribution method or an incorrect choice of degree of parallelism. After the first execution, the optimizer has a more complete set of statistics, this information can improve plan selection in the future. A query may be re-optimized several times, each time creating a more precise set of optimizer adjustments.

Re-optimization fixes plan problems that dynamic plans cannot solve. Dynamic plans can change relatively local parts of a plan, dynamic plans are not feasible for some global plan changes. For example, a query with an inefficient join order might cause suboptimal performance, but it is not feasible to adapt the join order during execution. In these cases, the optimizer automatically considers re-optimization. For re-optimization, the optimizer must decide which statistics to collect and when.

As of Oracle Database 12c, join statistics are collected. Statements are monitored over different executions for fluctuations in statistics. This works with adaptive cursor sharing for statements with bind variables. This feature improves the ability of the query processing engine (compilation and execution) to generate better execution plans.

The optimizer uses the statistics in the following ways:

- The statistics collectors and their associated optimizer estimates determine if automatic re-optimization is an option. If statistics differ from estimates by more than a threshold, the optimizer looks for a replacement plan.
- After the optimizer identifies a query as a re-optimization candidate, the database submits all collected statistics to the optimizer.

**Note:** The `IS_REOPTIMIZABLE` column can have three values. `N` indicates no reoptimization information is available. `Y` shows that the next execution matching this child cursor will trigger a re-optimization. And `R` indicates the child cursor does contain re-optimization information, but will not trigger re-optimization because the cursor was compiled in reporting mode.

## Quiz

A query plan changes during execution because runtime conditions indicate that optimizer estimates are inaccurate.

- a. True
- b. False



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

**Answer: a**

# Influencing the Optimizer Approach

The optimizer approach is to evaluate based on one of the following:

- Best throughput (default)
- Fastest response

The OPTIMIZER\_MODE parameter can be set to change the approach:

- ALL\_ROWS (default)
- FIRST\_ROWS [\_n]

The OPTIMIZER\_MODE can be set at various levels

- System – ALTER SYSTEM SET...
- Session – ALTER SESSION SET...
- Statement – hint - /\*+ FIRST\_ROWS\_10 \*/



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

The optimizer can be influenced to choose plans that either produce the fastest first *n* rows or produce the lowest cost for the entire returned set of rows. With the default ALL\_ROWS method, the optimizer looks for the lowest-cost plan without regard for the response time. With the fast-response method, the optimizer explores different plans and computes the cost to produce the first *n* rows for each plan. It picks the plan that produces the first *n* rows at the lowest cost. Remember that with fast-response optimization, a plan that produces the first *n* rows at the lowest cost might not be the optimal plan to produce the entire result. If the requirement is to obtain the entire result of a query, you should not use fast-response optimization. Instead, use the ALL\_ROWS parameter value or hint.

The parameters and hints have the same key words: ALL\_ROWS and FIRST\_ROWS\_n. The value of *n* can be 1, 10, 100, or 1000. The FIRST\_ROWS value is still available for backward compatibility only. If the OPTIMIZER\_MODE parameter is not set explicitly, it defaults to ALL\_ROWS.

At the session level, the optimizer approach set at the instance level can be overruled by using the ALTER SESSION command. A hint at the statement level overrides the optimizer approach.

# Optimizing SQL Statements

- Best throughput
  - Time required to complete the request
  - Suitable for:
    - Batch processing
    - Report applications
- Fast response
  - Time for retrieving the first rows
  - Suitable for:
    - Interactive applications
    - Web-based or GUI applications



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

Choose a goal for the optimizer based on the needs of your application:

- For applications performed in batch, optimize for best throughput. Throughput is usually more important in batch applications because the user initiating the application is concerned with only the time necessary for the application to complete. Response time is less important because the user does not examine the results of individual statements while the application is running.
- For interactive applications, optimize for best response time. Response time is usually important in interactive applications because the interactive user is waiting to see the first row or first few rows accessed by the statement. You can set the number of rows that should be retrieved first by using the optimizer mode of `FIRST_ROWS_n`, where *n* specifies the number of rows and can be customized for the different pages in the application.

**Note:** `FIRST_ROWS_n` mode is based solely on costs, and it is sensitive to the value of *n*. With small values of *n*, the optimizer tends to generate plans that consist of nested loop joins with index lookups. With large values of *n*, the optimizer tends to generate plans that consist of hash joins and full table scans.

# Quiz

The optimizer always calculates the cost of a full table scan from the table statistics, then tries other access paths to find a lower cost plan. One over the number of distinct values in a column (1/distinct values) is called:

- a. Cardinality
- b. Cost
- c. Predicate filter
- d. Selectivity
- e. Extended statistics



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

**Answer: d**

# Access Paths

- Full table scan
- Row ID scan
- Index scan
- Sample table scan
- Cluster scan



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

Access paths are the ways in which data is retrieved from the database. Any row can be located and retrieved with one of the methods mentioned in the slide. Access paths are ways in which data is retrieved from the database.

The access method must correspond to the data structures that are used in the database. Tables and indexes are the most common, but clusters, and index-organized tables (IOTs) are used by many. Even indexes have differing structures, B\*Tree, bitmap, and domain. (Domain index structure and access methods are user defined and beyond the scope of this course.)

In general, index access paths should be used for statements that retrieve a small subset of table rows, while full scans are more efficient when accessing a large portion of the table. Online transaction processing (OLTP) applications, which consist of short-running SQL statements with high selectivity, often are characterized by the use of index access paths. Decision support systems (DSS), on the other hand, tend to use full scans of the relevant objects. IOT and cluster structures are used in particular scenarios.

The sample table scan returns a random sample of the rows or blocks in the table. This method is used for sampling data in DSS, and data mining applications.

**Note:** This class includes a subset of the most commonly used access path operations.

# Choosing an Access Path

The optimizer chooses an access path based on:

- Available access paths for the statement
- Estimated cost of executing the statement, using each access path or combination of paths
- Presence of hints in SQL statement
- Presence of SQL profile
- Presence of SQL baseline plans



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

To choose an access path, the optimizer first determines which access paths are available by examining the conditions in the statement's WHERE clause and its FROM clause. The optimizer then generates a set of possible execution plans using available access paths and estimates the cost of each plan, using the statistics for the index, columns, and tables accessible to the statement. Finally, the optimizer chooses the execution plan with the lowest estimated cost.

When choosing an access path, the optimizer is influenced by the following:

- **Optimizer hints:** The optimizer's choice among available access paths can be overridden with hints. (Using hints requires changing the code.)
- **Statistics:** For example, if a table has not been analyzed since it was created, and the table statistics show that it is small, then the optimizer uses a full table scan. The LAST\_ANALYZED and BLOCKS columns in the ALL\_TABLES table reflect the statistics used by the optimizer.
- **SQL profiles:** These are additional statistics produced by the SQL Tuning Analyzer and they will influence the optimizer to produce a better plan.
- **SQL baseline plans:** If the plan the optimizer produces does not match a plan in the plan baseline, the plan in the baseline with the lowest cost will be used. For more information, see the lesson titled "SQL Performance Management."

**Note:** NLS\_SORT and NLS\_COMP session parameters will influence the access path.

## Full Table Scans

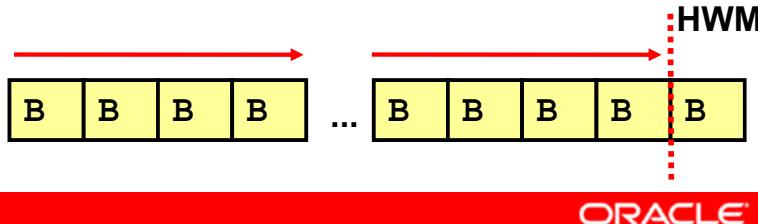
- Lack of index
- Large amount of data
- Small table
- Multiblock I/O calls
- All blocks below high-water mark

```
select * from emp where ename = 'KING';
-----
```

ID	Operation	Name	Rows	Bytes	Cost
0	SELECT STATEMENT		1	22	2
*	TABLE ACCESS FULL	EMP	1	22	2

Predicate Information (identified by operation id):

```
1 - filter("EMP"."ENAME"='KING')
```



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

This type of scan reads all rows from a table and filters out those that do not meet the selection criteria. A full table scan is one of the first access methods considered by the optimizer. A full table scan is usually one of the more expensive methods; however, a full table scan can be the most efficient access method, especially when a large portion of the table is being accessed, or when parallel servers can be used.

A full table scan will be used when a table does not have an index on the column in the predicate, the statistics indicate that a large portion of the table must be accessed, or the table is a small table. During a full table scan, all formatted blocks in the table that are under the high-water mark are scanned even if the blocks are empty. The high-water mark indicates the amount of space that has been formatted to receive data. Each row is examined to determine whether it satisfies the statement's WHERE clause, and unneeded rows are discarded.

The blocks read in a full table scan usually go into the PGA. If the buffer cache is relatively empty, compared to the size of the table, the blocks go into the database buffer cache, otherwise the PGA is the destination of choice. This default behavior can be altered with hints or table properties.

When the Oracle Database server performs a full table scan, the blocks are read sequentially. Because the blocks are adjacent, I/O calls larger than a single block can be used to speed up the process. Using multiblock reads usually means a full table scan can be performed more efficiently.

Full table scans have a lower cost than index range scans when accessing a large fraction of the blocks in a table, because full table scans can use larger I/O calls, and making fewer large I/O calls has a lower cost than making many smaller calls.

**Note:** A multiblock I/O call is more expensive than a single block I/O call, but a multiblock I/O call for 8 blocks, for example, is less expensive than 8 single-block calls.

`DB_MULTIBLOCK_READ_COUNT` is determined automatically on startup by testing the I/O subsystem. A value that is most efficient for the OS and I/O is calculated.

**Example:** A full table scan of a table that can be read with one I/O call, and is less expensive than an index lookup to retrieve only one row. The index must do one I/O to read the index header, at least one more to read the leaf block, and a third to read the table block.

The optimizer uses a full table scan in each of the following cases:

- **Lack of index:** If the query is unable to use any existing indexes, it uses a full table scan. For example, if there is a function used on the indexed column in the query, the optimizer is unable to use the index and instead uses a full table scan.
- **Large amount of data:** If the optimizer thinks that the query will access most of the blocks in the table, then it uses a full table scan, even though indexes might be available.
- **Small table:** If a table contains blocks fewer than the value of `DB_FILE_MULTIBLOCK_READ_COUNT` under the high-water mark, then a full table scan might be less costly because this can be read in a single I/O call.

#### Read I/O Wait events:

- **db file sequential read:** A count of standard reads, one block at a time into the buffer cache. This is seldom seen in a full table scan (FTS).
- **db files scattered read:** A count of multiblock reads into the buffer cache . Up to `DB_FILE_MULTIBLOCK_READ_COUNT` consecutive blocks are read at a time and scattered into buffers in the buffer cache.
- **direct path read:** A count of waits on direct reads into PGA or a temporary segment.
- **direct path read temp:** A count of waits on direct reads from a temporary tablespace. This is most like seen with sorts or temporary tables that do not fit in memory.

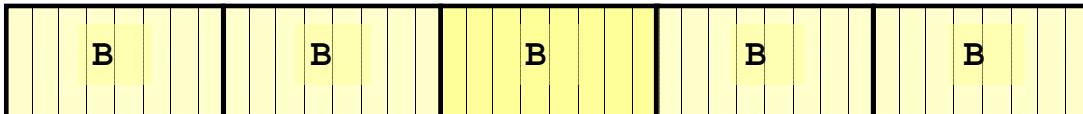
## Row ID Scans

The row ID specifies the data file and data block containing the row as well as the location of the row in that block.

```
explain plan for
select * from emp where rowid='AAAJ5QAAFAAABsvAAC';
-----
```

Id	Operation	Name	Rows	Cost
0	SELECT STATEMENT		1	1
1	TABLE ACCESS BY USER ROWID	EMP	1	1

Block 6959 – Row 2



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

The row ID of a row specifies the exact location of the row in the database. The row ID provides the data file, data block, and the location of the row in that block. Locating a row by specifying its row ID is the fastest way to retrieve a single row.

To access a table by row ID, the Oracle Database server first obtains the row IDs of the selected rows, either from the statement's WHERE clause or through an index scan of one or more of the table's indexes. The Oracle Database server then locates each selected row in the table based on its row ID. This is generally the next step after retrieving the row ID from an index.

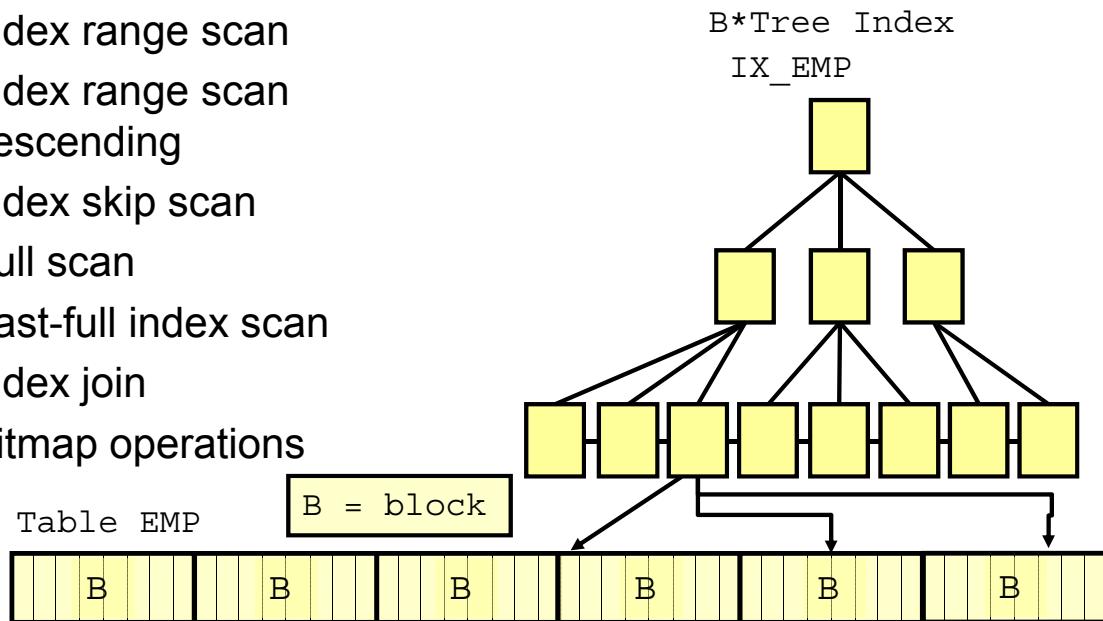
If the index contains all the columns needed for the statement, then the table access by row ID will not occur. The table access will be required for any columns in the statement that are not present in the index.

**Note:** Sometimes, because of row migration, a row will not be in the location specified by the index row ID. When there is not enough space in a block for an update to a row to fit in the block, the row is moved to another block and a pointer is placed in the original block. The index row ID, however, will still have only the address of the original block. For more information about row migration, see the lesson titled “Reducing the Cost of SQL Operations.”

# Index Operations

Types of index scans:

- Index unique scan
- Index range scan
- Index range scan descending
- Index skip scan
- Full scan
- Fast-full index scan
- Index join
- Bitmap operations



**ORACLE**

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

For all the index scan options, column values required by the SQL statement are used to search the index. The row ID is returned, and is used to access the table, if needed. If the SQL statement can be satisfied with just the index column values, the table access is not used. If the statement accesses other columns in addition to the indexed columns, then Oracle Database can find the rows in the table by using either a table access by row ID or a cluster scan. The index join operation avoids a table access by joining multiple indexes on the row ID to satisfy the select list. The bitmap operations include BITMAP INDEX SINGLE VALUE, BITMAP AND and OR operations, and BITMAP CONVERSION TO ROWID.

# B\*Tree Index Operations

## Index unique scan

Operation	Object	Rows
INDEX UNIQUE SCAN	PK_EMP	1

## Index range scan (ascending or descending)

INDEX RANGE SCAN	EMP_DEPT_NO	10
------------------	-------------	----

## Index skip scan

INDEX SKIP SCAN	EMP_DEPT_SAL	2
-----------------	--------------	---

## Full scan

INDEX FULL SCAN	EMP_LAST_SAL	54
-----------------	--------------	----

## Fast-full index scan

INDEX FAST FULL SCAN	EMP_LAST_SAL	54
----------------------	--------------	----

## Index join

HASH JOIN		
INDEX RANGE SCAN	EMP_MGR	10
INDEX FAST FULL SCAN	EMP_DEPT_NO	10

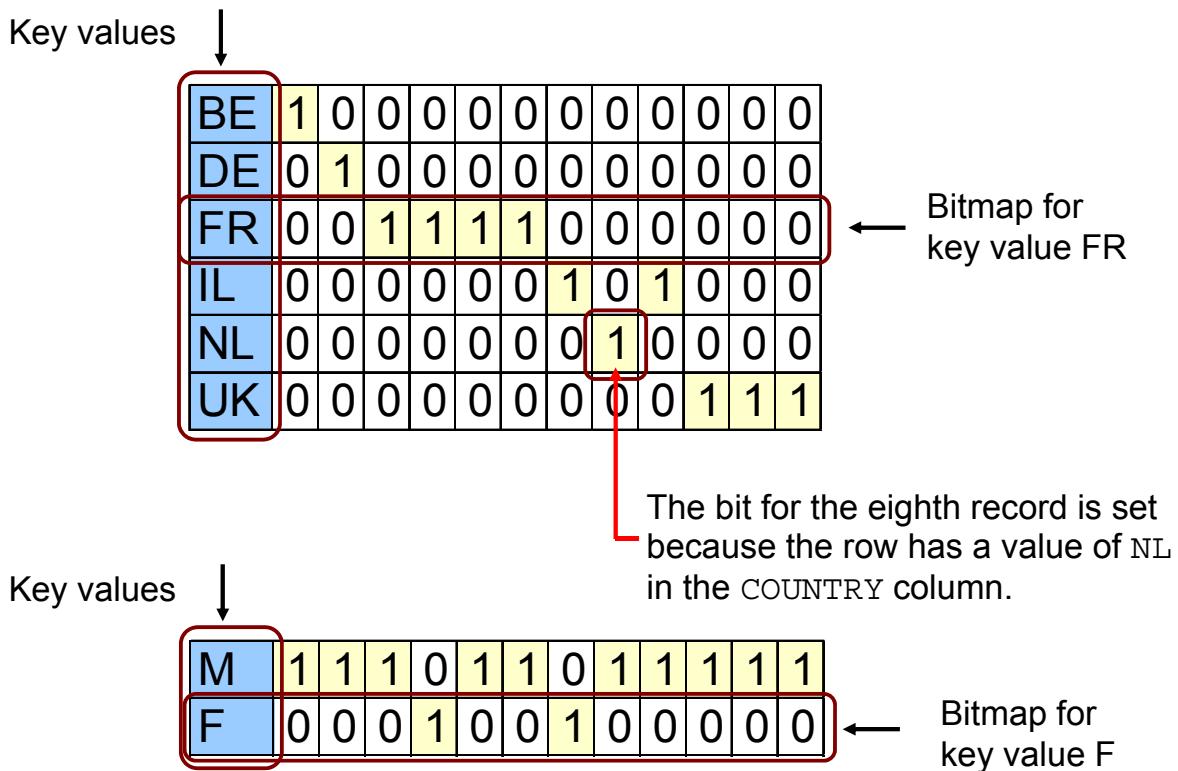


Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Each B\*Tree index operation and the corresponding execution plan operation name is shown in the slide. Each index operation also shows the index name and the number of rows that are retrieved or estimated.

- **Unique scan** is available only for equality predicates against a column with either a unique or primary key constraint.
- **Index range scan** is normally an ascending scan, but if the `INDEX_DESC` hint is used in the statement or there is an `ORDER BY DESC` clause, the scan can be done in descending order.
- **Index skip scan** lets a composite index be split logically into smaller subindexes. In skip scanning, the initial column of the composite index is not specified in the query. In other words, it is skipped.
- **Index full scan** reads all the blocks of an index in index order with single block reads. **Note:** The index entries are in order inside the blocks, but the order of the blocks in the segment are not usually sequential in the segment.
- **Index fast full scan** reads all the blocks of an index using multiblock I/O and discards unneeded blocks. It can be used when the index contains all the columns needed by the plan.
- **Index join** is a hash join on the row ID of the index entries. All columns needed must be in the indexes.

## Bitmap Indexes



ORACLE

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

A B\*Tree index stores an entry containing the key value and row ID for each row. A bitmap index starts with the root and branch blocks of the B\*Tree, but instead of an entry for each row, a bitmap is created for each key value. For each row ID a bit is set if the row contains a key value. A mapping function is used to transform bit position to an actual row ID. In a B\*Tree, an index entry points to a single row. With bitmap indexes, a single index entry uses a bitmap to point to many rows simultaneously.

Bitmap indexes are appropriate for highly repetitive data (data with few distinct values relative to the total number of rows in the table) that is mostly read-only. Bitmap indexes should never be considered in an OLTP database because of concurrency-related issues.

A bitmap index has a very different structure; no row IDs are stored. Each different key value has its own bitmap. Each bitmap header stores start and end row IDs. Based on these values, the server uses an internal algorithm to map bitmaps onto row IDs.

Each position in a bitmap maps to a potential row in the table. The contents of that position in the bitmap for a particular value indicate whether that row has that value in the bitmap columns. The value stored is 1 if the row values match the bitmap condition; otherwise, it is 0.

# Bitmap Index Access

```

create bitmap index CUST_COUNTRY on CUST(country_iso)

SELECT CUST_LAST_NAME
  FROM CUST
 WHERE country_iso = 'FR';

-----|-----|-----|-----|-----|-----|
| Id | Operation           | Name      | Rows  | Cost   |
-----|-----|-----|-----|-----|-----|
| 0  | SELECT STATEMENT    |           | 2921 | 368   |
| 1  |  TABLE ACCESS BY INDEX ROWID | CUST     | 2921 | 368   |
| 2  |    BITMAP CONVERSION TO ROWIDS |
| * 3 |    BITMAP INDEX SINGLE VALUE | CUST_COUNTRY |
-----|-----|-----|-----|-----|-----|
Predicate Information (identified by operation id):
-----|-----|
 3 - access(COUNTRY_ISO<'FR')

```



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

This query scans the bitmap for the country FR for positions containing a 1. Positions with a 1 have their corresponding rows returned for the query. In some cases (such as a query counting the number of rows with COUNTRY\_ISO = 'FR'), the query might simply use the bitmap itself and count the number of 1s (not needing the actual rows).

When the optimizer uses a bitmap, it must always include a BITMAP CONVERSION TO ROWIDS step to be able to access the table row.

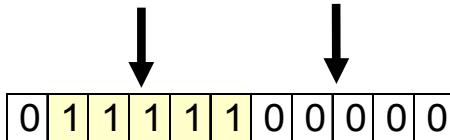
## Combining Bitmaps

```
SELECT *
FROM CUST
WHERE country_iso in('FR','DE');
```

FR	0	0	1	1	1	0	0	0	0	0	0
----	---	---	---	---	---	---	---	---	---	---	---

OR

DE	0	1	0	0	0	0	0	0	0	0	0
----	---	---	---	---	---	---	---	---	---	---	---



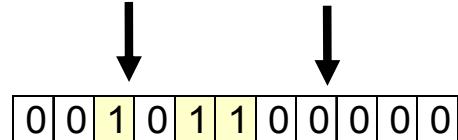
```
1 | TABLE ACCESS BY INDEX ROWID
2 | BITMAP CONVERSION TO ROWIDS
3 | BITMAP INDEX SINGLE VALUE
-----
4 - access(("COUNTRY"='DE' OR
"COUNTRY"='FR'))
```

```
SELECT *
FROM CUST
WHERE country = 'FR'
AND gender = 'M';
```

FR	0	0	1	1	1	0	0	0	0	0	0
----	---	---	---	---	---	---	---	---	---	---	---

AND

M	1	1	1	0	1	1	0	1	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---



```
1 | TABLE ACCESS BY INDEX ROWID
2 | BITMAP CONVERSION TO ROWIDS
3 | BITMAP AND
4 | BITMAP INDEX SINGLE VALUE
5 | BITMAP INDEX SINGLE VALUE
-----
4 - access("GENDER"='M')
5 - access("COUNTRY"='FR')
```

**ORACLE**

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

Because of fast bit-and, bit-minus, and bit-or operations, bitmap indexes are efficient:

- When using `IN (value_list)`
- When predicates are combined with `AND/OR`

Bitmap indexes can be used efficiently when a query combines several possible values for a column or when two separately indexed columns are used.

In some cases, a `WHERE` clause might reference several separately indexed columns.

If both the `COUNTRY` and `GENDER` columns have bitmap indexes, a bit-and operation on the two bitmaps would quickly locate the rows that you are looking for. The more complex the compound `WHERE` clauses become, the more you benefit from bitmap indexing.

# Bitmap Operations

## BITMAP CONVERSION

- Can be TO or FROM ROWIDS

## BITMAP INDEX

- Single value lookup
- Range scan
- Full scan
- Not fast full scan

## BITMAP MERGE

## BITMAP AND / OR

## BITMAP MINUS

## BITMAP KEY ITERATION



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

With BITMAP CONVERSION FROM ROWID, a B\*Tree index is converted on the fly by the optimizer into a bitmap (when the cost is lower than other methods) to be able to use these efficient bitmaps comparison operations. After the bitmap comparison has been done, the resultant bitmap is converted back into row IDs (BITMAP CONVERSION TO ROWIDS) to perform the data lookup.

With the BITMAP MINUS operation, the server takes the second bitmap and negates it— changing ones to zeros, and zeros to ones. The bitmap minus operation can then be performed as a BITMAP AND using this negated bitmap.

The BITMAP MINUS, AND, and OR operations are binary operations taking two bitmap row sources as input. The bitmap KEY ITERATION and MERGE operations are n-ary operations. MERGE combines a set of bitmaps. KEY ITERATION takes each row from a table row source and finds the corresponding bitmap from a bitmap index. This operation is followed by BITMAP MERGE operation that merges the set of bitmaps into one bitmap.

Because the bitmaps are stored in a B\*Tree structure, the basic access methods SINGLE VALUE, RANGE SCAN, and FULL SCAN work with bitmap indexes. SKIP SCAN and FAST FULL SCAN do not work with bitmap indexes.

# Join Operations

- Join is a binary operation that joins only two row sources.
- For each join, the optimizer determines (in sequence) the:
  1. Order in which to join the tables
  2. Best join method to apply for each join
  3. Access path for each row source
- The join order is affected by the following built-in rules:
  - A single-row predicate forces its row source to be placed first in the join order.
  - For outer joins, the table with the outer-joined table must come after the other table in the join order for processing the join.



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

A join is a binary operation that joins only two row sources. A multi-way join consists of joining two tables together and then joining the resultant row source to another table or row source until all the joins are complete.

For each join statement, the optimizer determines the following (in sequence):

1. The order in which to join the tables
2. The best join operation to apply for each join
3. The access path for each row source

The number of tables involved determines the number of join orders to be evaluated, the number of join orders grows factorially with the number of tables: (Number of tables)! = Number of join orders. Some join orders may not be considered, due to join order rules built into the optimizer:

- A single-row predicate forces its row source to be placed first in the join order.
- For outer joins, the table with the outer-joined table must come after the other table in the join order for processing the join.

The costs of various join methods are compared for each join order. Some methods cannot be used with some joins due to the limitations of the method. For example, a hash join cannot be used for a non-equijoin.

## Join Methods

- A join operation combines the output from two row sources and returns one resulting row source.
- Join methods include the following:
  - Nested loop join
  - Sort-merge join
  - Hash join
- Join methods have variations for:
  - Outer join
  - Full outer join



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

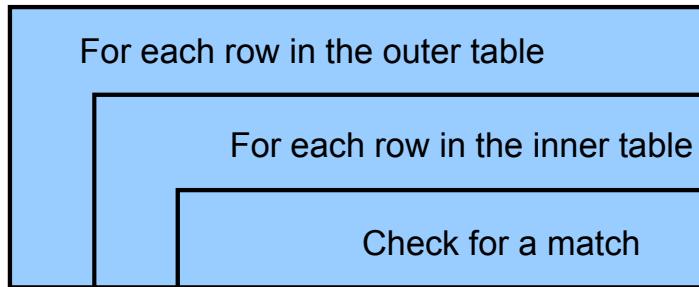
A join operation combines the output from two row sources (such as tables or views) and returns one resulting row source (data set). The Oracle Database server supports different join methods such as the following:

- **Nested loop join:** Useful when small subsets of data are being joined and if the join condition is an efficient way of accessing the second table
- **Hash join:** Used for joining large data sets. The optimizer uses the smaller of two tables or data sources to build a hash table on the join key in memory. It then scans the larger table, probing the hash table to find the joined rows. This method is best used when the smaller table fits in available memory. The cost is then limited to a single read pass over the data for the two tables.
- **Sort-merge join:** Can be used to join rows from two independent sources. Hash joins generally perform better than sort-merge joins. On the other hand, sort-merge joins can perform better than hash joins if both of the following conditions are met:
  - The row sources are already sorted.
  - A sort operation does not have to be done.

These methods are covered in the following slides.

## Nested Loop Joins

- One of the two tables is defined as the *outer* table (or the *driving* table).
- The other table is called the *inner* table.
- For each row in the outer table, all matching rows in the inner table are retrieved.



**ORACLE**

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

### NESTED LOOPS

```
TABLE ACCESS (...) OF our_outer_table
TABLE ACCESS (...) OF our_inner_table
```

Because they are often used with indexed table accesses, nested loops are common with an execution plan that is similar to the following:

```
NESTED LOOPS
  TABLE ACCESS (BY row ID) OF our_outer_table
    INDEX (... SCAN) OF outer_table_index (.....)
  TABLE ACCESS (BY row ID) OF our_inner_table
    INDEX (RANGE SCAN) OF inner_table_index (NON-UNIQUE)
```

The outer (driving) table is usually accessed with a full table scan. If a table in the join is determined to return a single row, the join order rule will force it to be the driving table. If the join predicate is a non-equijoin, the inner table may be accessed with a full table scan, once for each row in the driving table, resulting in repeated full table scans.

The nested loop join can always be used to join two tables. Therefore, the optimizer will use the nested loop join when sort-merge, hash, or cluster joins cannot be used. These join techniques are discussed later in this lesson.

For each row in the outer (driving) table, all rows in the inner table that satisfy the join predicate are retrieved. Any nonjoin predicates on the inner table are considered after this initial retrieval, unless a composite index (combining both the join and the nonjoin predicate) is used.

Oracle Database uses an internal optimization called *data block prefetching*. Data block prefetching is used by table lookup. When an index access path is chosen and the query cannot be satisfied by the index alone, the data rows indicated by the row ID must be fetched. The index entries are retrieved and sorted by row ID, then used to fetch table rows. This behavior is automatic and cannot be influenced by hints. In joins using data block prefetching, you may see an explain plan such as the following:

```
NESTED LOOPS (Cost=...)
  NESTED LOOPS (Cost=...)
    TABLE ACCESS (FULL) OF ...
      INDEX (RANGE SCAN) OF ...
        TABLE ACCESS (BY INDEX row ID) OF ...
```

# Hash Joins

A hash join is executed as follows:

- Both tables are split into as many partitions as required, using a full table scan.
- For each partition pair, a hash table is built in memory on the smallest partition.
- The other partition is used to probe the hash table.



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

Hash joins are performed only for equijoins. To execute a hash join, the Oracle Database server:

1. Performs a full table scan of one of the tables that must be joined (Scanning the smaller table first produces a lower cost.)
2. Builds a hash table out of the rows coming from first table (row source), on the join key
3. Performs a full table scan of the other table
4. Probes the hash table with each row coming from the second table (row source)

## Basic Execution Plan

```
HASH JOIN
  TABLE ACCESS (...) OF table_A
  TABLE ACCESS (...) OF table_B
```

## Performance Considerations

As a general rule, hash joins outperform sort-merge joins. In some cases, a sort-merge join can outperform a nested loop join; it is even more likely that a hash join will do so.

## Sort-Merge Joins

A sort-merge join is executed as follows:

1. The rows from each row source are sorted on the join predicate columns.
2. The two sorted row sources are then merged and returned as the resulting row source.



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

In the sort operations, the two row sources are sorted on the values of the columns used in the join predicate. If a row source has already been sorted in a previous operation, the sort-merge operation skips the sort on that row source. Sorting could make this join technique expensive, especially if sorting cannot be performed in memory.

The merge operation combines the two sorted row sources to retrieve every pair of rows that contain matching values for the columns used in the join predicate.

### Basic Execution Plan

```
MERGE (JOIN)
      SORT (JOIN)
        TABLE ACCESS (...) OF table_A
      SORT (JOIN)
        TABLE ACCESS (...) OF table_B
```

The table accesses (table\_A and table\_B) can be based on index scans if there are nonjoin predicates that can use an index. However, this join operation most often appears with full table scans on both tables.

The sequence (which table is sorted first) does not matter; there is no concept of an outer or driving table. Remember that the sorting can result in the creation of temporary segments on disk (giving more I/O than just the table scans themselves).

Sort-merge joins tend to perform better than an indexed nested loop join if the number of rows satisfying the join condition represents a major part of the total number of rows in the two tables.

**Note:** The breakeven point for a sort-merge join and an indexed nested loop join depends on the sorting performance of the database. Sort performance can be influenced by defining temporary tablespaces, which are of type `TEMPORARY`. For more details, see the lesson titled “Tuning PGA and Temporary Space.”

# Join Performance

Each join method works best in certain situations.

- Nested join
  - Small tables
  - Indexes on join columns
- Hash joins
  - Large tables
  - A large and small table
  - A large portion of the table is selected for the join
- Merge-sort
  - Non-equijoin
  - Row sources are already sorted



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

The optimizer will choose the method with the lowest estimated cost. Each method performs best in certain situations. There are some cases where a certain method will be required. A nested loops join is able to join any two row sources. The merge-sort join can join non-equijoin conditions. In general, the hash join is the fastest if at least one of the row sources can be held completely in memory, but the hash join can perform only equijoins.

Nested loops works best on small tables with indexes on the join conditions. If one of the row sources is a one row source, as with an equality lookup on a primary key value, then the join turns into a simple lookup. The optimizer always tries to put the smallest row source first, making it the *driving table*.

For hash joins, if the tables are so large that neither will fit in available memory, the row sources are partitioned, and the join proceeds partition by partition. This can use a lot of sort area memory, and I/O to the temporary tablespace. This method can still be the most cost effective, especially when parallel query servers are used.

Merge-sort can perform non-equijoins. This method may be chosen by the optimizer when the data can be accessed in sorted order; otherwise, the required sorts often make this method more expensive.

# How the Query Optimizer Chooses Execution Plans for Joins

The query optimizer determines:

- Row sources
- Type of join
- Join method
- Cost of execution plans
- Other costs such as:
  - I/O
  - CPU time
- Hints specified



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

The query optimizer considers the following when choosing an execution plan:

- The optimizer first determines whether joining two or more tables definitely results in a row source containing at most one row. The optimizer recognizes such situations based on `UNIQUE` and `PRIMARY KEY` constraints on the tables. If such a situation exists, the optimizer places these tables first in the join order. The optimizer then optimizes the join of the remaining set of tables.
- For join statements with outer join conditions, the table with the outer join operator must come after the other table in the condition in the join order. The optimizer does not consider join orders that violate this rule. Similarly, when a subquery has been converted into an antijoin or a semijoin, the tables from the subquery must come after those tables in the outer query block to which they were connected or correlated. However, hash antijoins and semijoins are able to override this ordering condition in certain circumstances.

**Note:** Hash antijoins are used to quickly calculate `NOT EXISTS` subquery joins.

- The optimizer generates a set of execution plans according to possible join orders, join methods, and available access paths. The optimizer then estimates the cost of each plan and chooses the one with the lowest cost. The optimizer estimates costs in the following ways:
  - The cost of a nested loop operation is based on the cost of reading each selected row of the outer table and each of its matching rows of the inner table into memory. The optimizer estimates these costs using the statistics in the data dictionary.
  - The cost of a sort-merge join is based largely on the cost of reading all the sources into memory and sorting them.
  - The cost of a hash join is based largely on the cost of building a hash table on one of the input sides to the join and using the rows from the other of the join to probe it.
- The optimizer also considers other factors when determining the cost of each operation. Here is an example: A smaller sort area size is likely to increase the cost for a sort-merge join because sorting takes more CPU time and I/O in a smaller sort area. Multiblock I/O is likely to decrease the cost for a sort-merge join in relation to a nested loop join. If a large number of sequential blocks can be read from disk in a single I/O, then an index on the inner table for the nested loop join is less likely to improve performance over a full table scan. The size of the multiblock is autotuned and determined at instance startup.
- With the query optimizer, the optimizer's choice of join orders can be overridden with the ORDERED hint. If the ORDERED hint specifies a join order that violates the rule for an outer join, then the optimizer ignores the hint and chooses the order. Also, you can override the optimizer's choice of join method with hints.

# Sort Operations

- **SORT JOIN**
- **SORT ORDER BY**
- **SORT AGGREGATE**
- **SORT UNIQUE**
- **SORT GROUP BY**



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

Sort operations result when users specify an operation that requires a sort. Commonly encountered operations include the following:

- **SORT JOIN** happens during a sort-merge join if the rows need to be sorted by the join key.
- **SORT ORDER BY** is required when the statement specifies an **ORDER BY** that cannot be satisfied by one of the indexes.
- **SORT AGGREGATE** does not actually involve a sort. It is used when aggregates are being computed across the whole set of rows.

**Note:** The following are rarely used by the optimizer. These have been replaced by a hash except in the case of set operations.

- **SORT UNIQUE** occurs if an operation requires unique values for the next step. Duplicate values are removed. This is required for **MINUS** and **INTERSECT** operations.
- **SORT GROUP BY** is used when aggregates are being computed for different groups in the data. For this operation, a hash is performed to aggregate the groups. The order of the groups is not guaranteed unless an **ORDER BY** clause is used.

# Tuning Sort Performance

- Because sorting large sets can be expensive, you should tune sort parameters.
- Note that most set operators cause implicit sorts.
- Minimize sorting by one of the following:
  - Use UNION ALL instead of UNION.
  - Enable index access to avoid sorting.



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

Several SQL language components, such as UNION, MINUS, and INTERSECT, caused implicit sorts. As a DBA, you can be aware of these conditions, but you can seldom change the code.

**Note:** The SQL standard does not guarantee the output in any order unless an ORDER BY clause is specified. In Oracle Database, GROUP BY and aggregate operations do not produce an ordering.

Some sorts can be avoided by creating indexes. Concatenated indexes, with the appropriate DESC or ASC attributes, can especially help to avoid sorts. Note, however, that you lose the advantage of multiblock I/O because, to avoid a sort, the index must be scanned sequentially. An index fast-full scan (using multiblock I/O) does not guarantee that the rows are returned in the correct order.

For information about tuning the sort area parameters see the lesson titled “Tuning PGA Memory and Temporary Space.”

## Quiz

You can think of a row source used by the optimizer as a table in memory. A row source 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 can be passed one row at a time to the next operation, but others must be complete (all rows retrieved) before going to the next operation. The cost of these intermediate operations are affected by:

- a. Size of the SQL work area (sort area)
- b. Number of columns in the row source
- c. Selectivity of the indexed columns
- d. Number of indexes on the table



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

### Answer: a

Some operations require the previous operation to complete before continuing, for example, a join in the sort-merge join. The sort has to be complete before the join can be performed. When this is the case, there must be memory space available to do the sort. This memory space is called SQL work areas. If this area is too small to hold the entire sort, the sort is broken into pieces and sent to disk in a temporary tablespace. It is much more costly to break the sort into pieces and write pieces to disk than doing the sort entirely in memory.

## Summary

In this lesson, you should have learned how to:

- Describe the optimizer's behavior
- Explain how statistics can affect the optimizer
- Describe how data structures affect the optimizer
- Adjust parameters to influence the optimizer



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

## **Practice 11 Overview: Influencing the Optimizer**

This practice covers managing statistics:

- Capturing extended statistics
- Observing the change in explain plan when the statistics change



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

