



Integrated Cloud Applications & Platform Services

Oracle Database 12c R2: SQL Tuning for Developers

Student Guide - Volume II

D99667GC20

Edition 2.0 | June 2018 | D104178

Learn more from Oracle University at education.oracle.com



ORACLE®

Author

Apoorva Srinivas

**Technical Contributors
and Reviewers**

Nigel Bayliss
Lance Ashdown

Editors

Aju Kumar
Smita Kommini

Graphic Designer

Seema Bopaiah

Publishers

Sujatha Nagendra
Jayanthi Keshavamurthy
Raghunath M

Copyright © 2018, 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 Course Introduction

- Lesson Agenda 1-2
- Course Objectives 1-3
- Audience and Prerequisites 1-4
- Course Outline Map 1-5
- Activities 1-6
- About You 1-7
- Lesson Agenda 1-8
- Sample Schemas Used in the Course 1-9
- Human Resources (HR) Schema 1-10
- Sales History (SH) Schema 1-11
- Order Entry(OE) Schema 1-13
- Lesson Agenda 1-14
- Class Account Information 1-15
- Lesson Agenda 1-16
- SQL Environments Available in the Course 1-17
- Lesson Agenda 1-18
- Oracle Cloud: Introduction 1-19
- Oracle Cloud Services 1-20
- Cloud Deployment Models 1-21
- Lesson Agenda 1-22
- Workshops, Demo Scripts, Code Examples, and Solution Scripts 1-23
- Appendices in the Course 1-24
- Additional Resources 1-25

2 Introduction to SQL Tuning

- Objectives 2-2
- Lesson Agenda 2-3
- What Is SQL Tuning? 2-4
- SQL Tuning Session 2-5
- Recognize: What Is Bad SQL? 2-6
- Clarify: Understand the Current Issue 2-7
- Verify: Collect Data 2-8
- Verify: Is the Bad SQL a Real Problem? (Top-Down Analysis) 2-10
- Lesson Agenda 2-12

SQL Tuning Strategies: Overview	2-13
Checking the Basics	2-14
Advanced SQL Tuning Analysis	2-15
Parse Time Reduction Strategy	2-16
Plan Comparison Strategy	2-17
Quick Solution Strategy	2-18
Finding a Good Plan	2-19
Implementing the New Good Plan	2-20
Query Analysis Strategy: Overview	2-21
Query Analysis Strategy: Collecting Data	2-22
Query Analysis Strategy: Examining SQL Statements	2-23
Query Analysis Strategy: Analyzing Execution Plans	2-24
Query Analysis Strategy: Finding Execution Plans	2-25
Query Analysis Strategy: Reviewing Common Observations and Causes	2-26
Query Analysis Strategy: Determining Solutions	2-27
Lesson Agenda	2-28
Development Environments: Overview	2-29
What Is Oracle SQL Developer?	2-30
Coding PL/SQL in Oracle SQL*Plus	2-31
Quiz	2-32
Summary	2-34
Practice 2: Overview	2-35

3 Using Application Tracing Tools

Objectives	3-2
Lesson Agenda	3-3
Using Application Tracing: Overview	3-4
Steps Needed Before Tracing	3-5
Lesson Agenda	3-7
Application Tracing Tools: Overview	3-8
Lesson Agenda	3-9
Using the SQL Tracing Facility	3-10
Tracing Your Own Session: Example	3-11
Tracing a Specific User: Example	3-12
Consideration: Tracing Challenge	3-13
Lesson Agenda	3-14
Creating a Database Operation	3-15
Monitoring Using Cloud Control	3-16
Lesson Agenda	3-17
End-to-End Application Tracing	3-18
Service Tracing: Example	3-19

Module Tracing: Example	3-20
Action Tracing: Example	3-21
Client Tracing: Example	3-22
Session Tracing: Example	3-23
Tracing Your Own Session: Example	3-24
Lesson Agenda	3-25
trcseSS Utility	3-26
Invoking the trcseSS Utility	3-27
Using the trcseSS Utility: Example	3-28
Lesson Agenda	3-29
TKPROF Utility: Overview	3-30
Invoking the TKPROF Utility	3-31
TKPROF Sorting Options	3-33
TKPROF Report Structure	3-35
Interpreting a TKPROF Report: Example 1	3-38
Interpreting a TKPROF Report: Example 2	3-39
Interpreting a TKPROF Report: Example 3	3-40
What to Verify: Example	3-41
Quiz	3-46
Summary	3-47
Practice 3: Overview	3-48

4 Understanding Basic Tuning Techniques

Objectives	4-2
Lesson Agenda	4-3
Developing Efficient SQL: Overview	4-4
Scripts Used in This Lesson	4-5
Example 1: Table Design	4-6
Example 2: Index Usage	4-7
Example 3: Transformed Index	4-8
Example 4: Data Type Mismatch	4-9
Example 5: Tuning the ORDER BY Clause	4-10
Example 6: Retrieving a MAX value	4-12
Example 7: Retrieving a MAX value	4-13
Example 8: Correlated Subquery	4-15
Example 9: UNION and UNION ALL	4-16
Example 10: Avoiding the Use of HAVING	4-17
Example 11: Tuning the BETWEEN Operator	4-19
Example 12: Tuning the Join Order	4-20
Example 13: Testing for Existence of Rows	4-22
Example 14: LIKE '%STRING'	4-23

- Example 15: Using Caution When Managing Views 4-24
- Example 16: Writing a Combined SQL Statement 4-26
- Example 17: Writing a Multitable INSERT Statement 4-27
- Example 18: Using a Temporary Table 4-28
- Example 19: Using the WITH Clause 4-29
- Example 20: Partition Pruning 4-31
- Example 21: Using a Bind Variable 4-32
- Example 22: NULL Usage 4-33
- Example 23: Tuning a Star Query by Using the Join Operation 4-34
- Example 24: Creating a New Index 4-35
- Example 25: Join Column and Index 4-36
- Example 26: Ordering Keys for Composite Index 4-37
- Example 27: Bitmap Join Index 4-38
- Example 28: Tuning a Complex Logic 4-39
- Example 29: Using a Materialized View 4-40
- Example 30: Star Transformation 4-41
- Summary 4-42
- Practice 4: Overview 4-43

5 Optimizer Fundamentals

- Objectives 5-2
- Lesson Agenda 5-3
- SQL Statement Representation 5-4
- Lesson Agenda 5-6
- SQL Statement Processing: Overview 5-7
- SQL Statement Parsing 5-8
- SQL Optimization 5-10
- SQL Row Source Generation 5-11
- SQL Row Source Generation: Example 5-12
- SQL Execution 5-13
- Quiz 5-14
- Lesson Agenda 5-15
- Why Do You Need an Optimizer? 5-16
- Components of the Optimizer 5-18
- Lesson Agenda 5-19
- Query Transformer 5-20
- Transformer: Cost-Based OR Expansion Example 5-21
- Transformer: Subquery Unnesting Example 5-22
- Transformer: View Merging Example 5-23
- Transformer: Predicate Pushing Example 5-24
- Transformer: Transitivity Example 5-25

Hints for Query Transformation	5-26
Quiz	5-29
Lesson Agenda	5-32
Query Estimator: Selectivity and Cardinality	5-33
Importance of Selectivity and Cardinality	5-34
Selectivity and Cardinality: Example	5-35
Query Estimator: Cost	5-37
Query Estimator: Cost Components	5-38
Lesson Agenda	5-39
Plan Generator	5-40
Quiz	5-41
Lesson Agenda	5-42
Adaptive Query Optimization	5-43
Lesson Agenda	5-44
Controlling the Behavior of the Optimizer	5-45
Optimizer Features and Oracle Database Releases	5-50
Summary	5-51
Practice 5: Overview	5-52

6 Generating and Displaying Execution Plans

Objectives	6-2
Lesson Agenda	6-3
What Is an Execution Plan?	6-4
Reading an Execution Plan	6-5
Reviewing the Execution Plan	6-6
Where to Find Execution Plans	6-7
Viewing Execution Plans	6-9
Lesson Agenda	6-10
The EXPLAIN PLAN Command: Overview	6-11
The EXPLAIN PLAN Command: Syntax	6-12
The EXPLAIN PLAN Command: Example	6-13
Lesson Agenda	6-14
PLAN_TABLE	6-15
Displaying from PLAN_TABLE	6-16
Displaying from PLAN_TABLE: ALL	6-17
Displaying from PLAN_TABLE: ADVANCED	6-18
Explain Plan Using Oracle SQL Developer	6-19
Quiz	6-20
Lesson Agenda	6-22
AUTOTRACE	6-23
The AUTOTRACE Syntax	6-24

AUTOTRACE: Examples	6-25
AUTOTRACE: Statistics	6-26
AUTOTRACE by Using SQL Developer	6-28
Quiz	6-29
Lesson Agenda	6-30
DBMS_SQL_MONITOR Package	6-31
Monitoring Using Cloud Control	6-32
Lesson Agenda	6-33
Links Between Important Dynamic Performance Views	6-34
V\$SQL_PLAN View: Overview	6-35
V\$SQL_PLAN Columns	6-36
The V\$SQL_PLAN_STATISTICS View	6-37
Querying V\$SQL_PLAN	6-39
Lesson Agenda	6-41
Automatic Workload Repository	6-42
Important AWR Views	6-44
Comparing Execution Plans by Using AWR	6-45
Generating SQL Reports from AWR Data	6-47
Lesson Agenda	6-48
SQL Monitoring: Overview	6-49
SQL Monitoring Report: Example	6-51
Quiz	6-54
Summary	6-55
Practice 6: Overview	6-56

7 Interpreting Execution Plans and Enhancements

Objectives	7-2
Lesson Agenda	7-3
Interpreting a Serial Execution Plan	7-4
Execution Plan Interpretation: Example 1	7-6
Execution Plan Interpretation: Example 2	7-9
Execution Plan Interpretation: Example 3	7-11
Execution Plan Interpretation: Example 4	7-12
Lesson Agenda	7-13
Reading More Complex Execution Plans	7-14
Lesson Agenda	7-15
Looking Beyond Execution Plans	7-16
Quiz	7-17
Lesson Agenda	7-18
Adaptive Query Optimization: Overview	7-19
Adaptive Plans: Join Method	7-20

Adaptive Join Method: Example 7-21
Adaptive Join Method: Working 7-22
Displaying the Default Plan 7-23
Displaying the Final Plan 7-24
Displaying the Full Adaptive Plan 7-25
Adaptive Plans: Indicator in V\$SQL 7-26
Lesson Agenda 7-27
Adaptive Plans: Parallel Distribution Method 7-28
Parallel Distribution Method: Example 7-29
HYBRID-HASH Method: Example 7-30
Quiz 7-31
Summary 7-33
Practice 7: Overview 7-34

8 Optimizer: Table and Index Access Paths

Objectives 8-2
Lesson Agenda 8-3
Row Source Operations 8-4
Lesson Agenda 8-5
Main Structures and Access Paths 8-6
Lesson Agenda 8-7
Full Table Scan 8-8
Using Full Table Scan 8-9
ROWID Scan 8-10
Sample Table Scans 8-11
Quiz 8-13
Lesson Agenda 8-14
Indexes: Overview 8-15
Normal B*-tree Indexes 8-17
Index Scans 8-18
Index Unique Scan 8-19
Index Range Scan 8-20
Index Range Scan: Descending 8-21
Descending Index Range Scan 8-22
Index Range Scan: Function-Based 8-23
Index Full Scan 8-24
Index Fast Full Scan 8-25
Index Skip Scan 8-26
Index Skip Scan: Example 8-27
Index Join Scan 8-28
B*-tree Indexes and Nulls 8-29

Using Indexes: Considering Nullable Columns 8-30
Index-Organized Tables 8-31
Index-Organized Table Scans 8-32
Bitmap Indexes 8-33
Bitmap Index Access: Examples 8-34
Combining Bitmap Indexes: Examples 8-35
Combining Bitmap Index Access Paths 8-36
Bitmap Operations 8-37
Bitmap Join Index 8-38
Composite Indexes 8-39
Invisible Index: Overview 8-40
Invisible Indexes: Examples 8-41
Guidelines for Managing Indexes 8-42
Quiz 8-44
Lesson Agenda 8-46
Common Observations 8-47
Why Is a Full Table Scan Used? 8-48
Why Is Full Table Scan Not Used? 8-49
Why Is an Index Scan Not Used? 8-51
Summary 8-55
Practice 8: Overview 8-56

9 Optimizer: Join Operators

Objectives 9-2
Lesson Agenda 9-3
How the Query Optimizer Executes Join Statements 9-4
Join Methods 9-5
Nested Loops Join 9-6
Nested Loops Join: Prefetching 9-7
Nested Loops Join: 12c Implementation 9-8
Sort-Merge Join 9-9
Hash Join 9-10
Cartesian Join 9-11
Lesson Agenda 9-12
Join Types 9-13
Equijoins and Nonequijoins 9-14
Outer Joins 9-15
Semijoins 9-16
Antijoins 9-17
Quiz 9-18

Summary 9-22
Practice 9: Overview 9-23

10 Other Optimizer Operators

Objectives 10-2
Lesson Agenda 10-3
Result Cache Operator 10-4
Using RESULT_CACHE 10-6
Using Result Cache Table Annotations 10-7
Lesson Agenda 10-8
Clusters 10-9
When Are Clusters Useful? 10-10
Cluster Access Path: Examples 10-12
Lesson Agenda 10-13
Sorting Operators 10-14
Lesson Agenda 10-15
Buffer Sort Operator 10-16
Lesson Agenda 10-17
Inlist Iterator 10-18
Lesson Agenda 10-19
View Operator 10-20
Lesson Agenda 10-21
Count Stop Key Operator 10-22
Lesson Agenda 10-23
Min/Max and First Row Operators 10-24
Lesson Agenda 10-25
Other N-Array Operations 10-26
FILTER Operations 10-27
OR Expansion Operation 10-28
UNION [ALL], INTERSECT, MINUS 10-29
Quiz 10-30
Summary 10-34
Practice 10: Overview 10-35

11 Introduction to Optimizer Statistics Concepts

Objectives 11-2
Lesson Agenda 11-3
Optimizer Statistics 11-4
Table Statistics (USER_TAB_STATISTICS) 11-5
Index Statistics(USER_IND_STATISTICS) 11-6

Index Clustering Factor	11-8
Column Statistics (USER_TAB_COL_STATISTICS)	11-10
Lesson Agenda	11-11
Column Statistics: Histograms	11-12
Frequency Histograms	11-13
Viewing Frequency Histograms	11-14
Top Frequency Histogram	11-15
Viewing Top Frequency Histograms	11-16
Height-Balanced Histograms	11-17
Viewing Height-Balanced Histograms	11-18
Hybrid Histograms	11-19
Viewing Hybrid Histograms	11-20
Best Practices: Histogram	11-21
Best Practices: Histograms	11-22
Lesson Agenda	11-23
Column Statistics: Extended Statistics	11-24
Column Group Statistics	11-26
Expression Statistics	11-28
Lesson Agenda	11-30
Session-Specific Statistics for Global Temporary Tables	11-31
Session-Specific Statistics for Global Temporary Tables: Example	11-33
Lesson Agenda	11-34
System Statistics	11-35
System Statistics: Example	11-36
Lesson Agenda	11-37
Gathering Statistics: Overview	11-38
Manual Statistics Gathering	11-39
When to Gather Statistics Manually	11-40
Managing Statistics: Overview (Export / Import / Lock / Restore / Publish)	11-41
Quiz	11-42
Summary	11-43
Practice 11: Overview	11-44

12 Using Bind Variables

Objectives	12-2
Lesson Agenda	12-3
Cursor Sharing and Different Literal Values	12-4
Lesson Agenda	12-5
Cursor Sharing and Bind Variables	12-6
Bind Variables in SQL*Plus	12-7
Bind Variables in Enterprise Manager	12-8

Bind Variables in Oracle SQL Developer 12-9
Lesson Agenda 12-10
Bind Variable Peeking 12-11
Lesson Agenda 12-13
Cursor Sharing Enhancements 12-14
CURSOR_SHARING Parameter 12-16
Forcing Cursor Sharing: Example 12-17
Lesson Agenda 12-18
Adaptive Cursor Sharing: Overview 12-19
Adaptive Cursor Sharing: Architecture 12-20
Adaptive Cursor Sharing: Views 12-22
Adaptive Cursor Sharing: Example 12-23
Interacting with Adaptive Cursor Sharing 12-24
Common Observations 12-25
Quiz 12-26
Summary 12-29
Practice 12: Overview 12-30

13 SQL Plan Management

Objectives 13-2
Lesson Agenda 13-3
Maintaining SQL Performance 13-4
Lesson Agenda 13-5
SQL Plan Management: Overview 13-6
Components of SQL Plan Management 13-7
SQL Plan Baseline: Architecture 13-8
Lesson Agenda 13-11
Basic Tasks in SQL Plan Management 13-12
Configuring SQL Plan Management 13-13
Loading SQL Plan Baselines 13-14
SQL Plan Selection 13-15
Evolving SQL Plan Baselines 13-17
Lesson Agenda 13-18
Adaptive SQL Plan Management 13-19
Managing the SPM Evolve Advisor Task 13-20
Important SQL Plan Baseline Attributes 13-22
Lesson Agenda 13-24
Possible SQL Plan Manageability Scenarios 13-25
SQL Performance Analyzer and SQL Plan Baseline Scenario 13-26
Loading a SQL Plan Baseline Automatically 13-27
Purging SQL Management Base Policy 13-28

Lesson Agenda	13-29
Enterprise Manager and SQL Plan Baselines	13-30
Lesson Agenda	13-31
Loading Hinted Plans into SPM: Example	13-32
Quiz	13-34
Summary	13-35
Practice 13: Overview	13-36

14 Workshops

Objectives	14-2
Overview	14-3
Workshop 1	14-4
Workshop 2	14-5
Workshop 3	14-6
Workshop 4	14-7
Workshop 5	14-8
Workshop 6 and 7 (Optional)	14-9
Workshop 8	14-10
Workshop 9	14-11
Summary	14-12

A Using SQL Developer

Objectives	A-2
What Is Oracle SQL Developer?	A-3
SQL Developer: Specifications	A-4
SQL Developer 17.2 Interface	A-5
Creating a Database Connection	A-7
Browsing Database Objects	A-10
Displaying the Table Structure	A-11
Browsing Files	A-12
Creating a Schema Object	A-13
Creating a New Table: Example	A-14
Using the SQL Worksheet	A-15
Executing SQL Statements	A-18
Saving SQL Scripts	A-19
Executing Saved Script Files: Method 1	A-20
Executing Saved Script Files: Method 2	A-21
Formatting the SQL Code	A-22
Using Snippets	A-23
Using Snippets: Example	A-24
Using the Recycle Bin	A-25

Debugging Procedures and Functions	A-26
Database Reporting	A-27
Creating a User-Defined Report	A-28
Search Engines and External Tools	A-29
Setting Preferences	A-30
Resetting the SQL Developer Layout	A-32
Data Modeler in SQL Developer	A-33
Summary	A-34

B SQL Tuning Advisor

Objectives	B-2
Tuning SQL Statements Automatically	B-3
Application Tuning Challenges	B-4
SQL Tuning Advisor: Overview	B-5
Stale or Missing Object Statistics	B-6
SQL Statement Profiling	B-7
Plan Tuning Flow and SQL Profile Creation	B-8
SQL Tuning Loop	B-9
Access Path Analysis	B-10
SQL Structure Analysis	B-11
SQL Tuning Advisor: Usage Model	B-12
Cloud Control and SQL Tuning Advisor	B-13
Running SQL Tuning Advisor: Example	B-14
Schedule SQL Tuning Advisor Page	B-15
Implementing Recommendations	B-16
Compare Explain Plan Page	B-17
Quiz	B-18
Summary	B-20

C Using SQL Access Advisor

Objectives	C-2
SQL Access Advisor: Overview	C-3
SQL Access Advisor: Usage Model	C-4
Recommendations	C-5
SQL Access Advisor Session: Initial Options	C-6
SQL Access Advisor: Workload Source	C-8
SQL Access Advisor: Recommendation Options	C-9
SQL Access Advisor: Schedule and Review	C-10
SQL Access Advisor: Results	C-11
SQL Access Advisor: Results and Implementation	C-12

Quiz C-13

Summary C-15

D Exploring the Oracle Database Architecture

Objectives D-2

Oracle Database Server Architecture: Overview D-3

Connecting to the Database Instance D-4

Oracle Database Memory Structures: Overview D-6

Database Buffer Cache D-7

Redo Log Buffer D-8

Shared Pool D-9

Processing a DML Statement: Example D-10

COMMIT Processing: Example D-11

Large Pool D-12

Java Pool and Streams Pool D-13

Program Global Area D-14

Background Process D-15

Automatic Shared Memory Management D-17

Automated SQL Execution Memory Management D-18

Automatic Memory Management D-19

Database Storage Architecture D-20

Logical and Physical Database Structures D-22

Segments, Extents, and Blocks D-24

SYSTEM and SYSAUX Tablespaces D-25

Quiz D-26

Summary D-29

E Real-Time Database Operation Monitoring

Objectives E-2

Real-Time Database Operation Monitoring: Overview E-3

Use Cases E-4

Current Tools E-5

Defining a DB Operation E-6

Scope of a Composite DB Operation E-7

Database Operation Concepts E-8

Identifying a Database Operation E-9

Enabling Monitoring of Database Operations E-10

Identifying, Starting, and Completing a Database Operation E-11

Monitoring the Progress of a Database Operation E-12

Monitoring Load Database Operations E-13

Monitoring Load Database Operation Details	E-14
Reporting Database Operations by Using Views	E-15
Reporting Database Operations by Using Functions	E-17
Database Operation Tuning	E-18
Quiz	E-19
Summary	E-21

F Gathering and Managing Optimizer Statistics

Objectives	F-2
Lesson Agenda	F-3
Gathering Statistics: Overview	F-4
Automatic Optimizer Statistics Gathering	F-5
Optimizer Statistic Preferences: Overview	F-7
Manual Statistics Gathering	F-9
When to Gather Statistics Manually	F-10
Manual Statistics Collection: Factors	F-11
Gathering Object Statistics: Example	F-12
Object Statistics: Best Practices	F-13
Lesson Agenda	F-14
Dynamic Statistics: Overview	F-15
Dynamic Statistics at Work	F-16
OPTIMIZER_DYNAMIC_SAMPLING	F-18
Lesson Agenda	F-19
Automatic Re-optimization	F-20
Re-optimization: Statistics Feedback	F-21
Statistics Feedback: Monitoring Query Executions	F-22
Statistics Feedback: Reparsing Statements	F-23
Lesson Agenda	F-24
SQL Plan Directives	F-25
SQL Plan Directives: Example	F-29
Lesson Agenda	F-30
Online Statistics Gathering for Bulk Loads	F-31
Lesson Agenda	F-33
Concurrent Statistics Gathering	F-34
Concurrent Statistics Gathering: Creating Jobs at Different Levels	F-35
Lesson Agenda	F-36
Gathering System Statistics: Automatic Collection – Example	F-37
Gathering System Statistics: Manual Collection – Example	F-38
Lesson Agenda	F-39
Managing Statistics: Overview (Export / Import / Lock / Restore / Publish)	F-40
Exporting and Importing Statistics	F-41

Locking and Unlocking Statistics	F-42
Restoring Statistics	F-43
Deferred Statistics Publishing: Overview	F-45
Deferred Statistics Publishing: Example	F-47
Running Statistics Gathering Functions in Reporting Mode	F-48
Reporting on Past Statistics Gathering Operations	F-49
Managing SQL Plan Directives	F-50
Quiz	F-51
Summary	F-54

Other Optimizer Operators

The ORACLE logo, consisting of the word "ORACLE" in white capital letters on a red rectangular background.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe the result cache operator
- Describe SQL operators for:
 - Clusters
 - In-List
 - Sorts
 - Filters
 - Set Operations



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

This lesson helps you to understand the execution plans that use the common operators of other access methods.

Lesson Agenda

- Result Cache Operator
- Clusters
 - When Are Clusters Useful?
 - Cluster Access Path: Examples
- Sorting Operators
- Buffer Sort Operator
- Inlist Iterator
- View Operator
- Count Stop Key Operator
- Min/Max and First Row Operators
- Other N-Array Operations

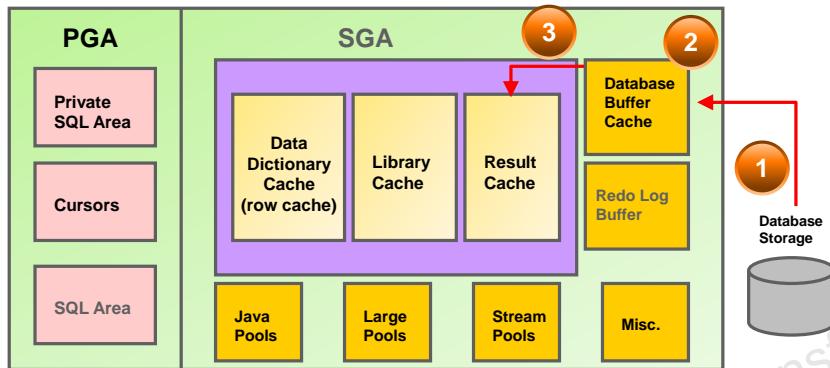


ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Result Cache Operator

A result cache is an area of memory either in SGA or client application memory that stores the results of a database query or query block for reuse.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The SQL query result cache enables explicit caching of query result sets and query fragments in database memory. A dedicated memory buffer stored in the shared pool can be used for storing and retrieving the cached results. The query results stored in this cache become invalid when data in the database objects that are accessed by the query is modified. Although the SQL query cache can be used for any query, good candidate statements are the ones that need to access a very high number of rows to return only a fraction of them. This is mostly the case for data-warehousing applications.

When a query is executed for the very first time, the user's process searches for the data in the database buffer cache. If data exists (because someone else had retrieved this data before), it uses it; otherwise, it performs an I/O operation to retrieve data from the data file on disk into the buffer cache, and from this data, the final result set is built.

The Result Cache can be managed either on the client side or the server side. A client-side Result Cache implementation would require the application to use the Oracle Call Interface (OCI) calls. Comparatively, server-side implementation is much simpler. In the course, the focus is on the server-side implementation of the Result Cache.

If you want to use the query result cache and the `RESULT_CACHE_MODE` initialization parameter is set to `MANUAL`, you must explicitly specify the `RESULT_CACHE` hint in your query. For a server-side implementation, the same query could be executed with `/*+ RESULT_CACHE */` hint or the `result_cache_mode` parameter could be set to `AUTO`.

This hint introduces the `ResultCache` operator in the execution plan for the query. When you execute the query, the `ResultCache` operator looks up the result cache memory to check whether the result for the query already exists in the cache. If it exists, the result is retrieved directly out of the cache. If it does not yet exist in the cache, the query is executed, the result is returned as output, and

it is also stored in the result cache memory. If the `RESULT_CACHE_MODE` initialization parameter is set to `FORCE`, and you do not want to store the result of a query in the result cache, you must use the `NO_RESULT_CACHE` hint in your query.

The `DBMS_RESULT_CACHE` package provides statistics, information, and operators that enable you to manage memory allocation for the server result cache. Use the `DBMS_RESULT_CACHE` package to perform operations such as retrieving statistics on cache memory usage and flushing the cache.

Using RESULT_CACHE

```
SELECT /*+ RESULT_CACHE */ department_id, AVG(salary)
FROM employees
GROUP BY department_id;
```

OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT		
RESULT CACHE	c7mhsc1jxgwq2bkrvb8y33fs6v	
HASH		GROUP BY
TABLE ACCESS	EMPLOYEES	FULL

```
SELECT id, type, creation_timestamp, block_count,
column_count, pin_count, row_count FROM
V$RESULT_CACHE_OBJECTS
WHERE cache_id = 'c7mhsc1jxgwq2bkrvb8y33fs6v';
```

Script Output X Explain Plan X Query Result X						
SQL All Rows Fetched: 1 in 0.006 seconds						
ID	TYPE	CREATION_TIMESTAMP	BLOCK_COUNT	COLUMN_COUNT	PIN_COUNT	ROW_COUNT
1	1069	Result 08-FEB-18	1	2	0	12



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

The parameters `client_result_cache_lag` and `client_result_cache_size` are used to configure the Result Cache at the client side. The other parameters are used for configuring the Result Cache at the server side.

The size of the Result Cache on the server is determined by two parameters: `result_cache_max_result` and `result_cache_max_size`.

The example in the slide shows a query of employees that uses the `RESULT_CACHE` hint to retrieve rows from the server result cache. Here, the results are retrieved directly from the cache, as indicated in step 1 of the execution plan. The value in the Name column is the cache ID of the result. The `CACHE_ID` for a query does not match the `SQL_ID` used to identify the query in the library cache and contained in `V$SQL`. Unlike the `SQL_ID`, which is generated for every SQL query that is executed against an Oracle database, the `CACHE_ID` is for an area or bucket in the Result Cache section of the shared pool that stores the end result of the query.

There are several views to monitor information related to the Result Cache. The objects that are related to the Result Cache can be obtained from the `V$RESULT_CACHE_OBJECTS` view. The following query helps verify the result set contained in the Result Cache for the `CACHE_ID` named `c7mhsc1jxgwq2bkrvb8y33fs6v`.

Using Result Cache Table Annotations

- You can also use table annotations to control result caching.
- You can use table annotations to avoid the necessity of adding result cache hints to queries at the application level.

```
CREATE TABLE sales (...) RESULT_CACHE (MODE DEFAULT); SELECT prod_id,  
SUM(amount_sold)  
FROM sales  
GROUP BY prod_id ORDER BY prod_id;
```

```
ALTER TABLE sales RESULT_CACHE (MODE FORCE);  
SELECT prod_id, SUM(amount_sold)  
FROM sales  
GROUP BY prod_id HAVING prod_id=136;  
  
SELECT /*+ NO_RESULT_CACHE */ *  
FROM sales ORDER BY time_id DESC;
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Table annotations affect the entire query, not query segments. Because a table annotation has a lower precedence than a SQL result cache hint, you can override table and session settings by using hints at the query level.

The `DEFAULT` table annotation prevents the database from caching results at the table level.

- `DEFAULT`: If at least one table in a query is set to `DEFAULT`, result caching is *not* enabled at the table level for this query.
- `FORCE`: If all the tables of a query are marked as `FORCE`, the query result is considered for caching. The `FORCE` table annotation forces the database to cache results at the table level.

The example in the slide shows a `CREATE TABLE` statement that uses the `DEFAULT` table annotation to create a table called `sales` and a `SELECT` query on this table. In this example, the `sales` table is created with a table annotation that disables result caching. The example also shows a query of the `sales` table, whose results are not considered for caching because of the table annotation.

The second example includes two queries of the `sales` table. The first query, which is frequently used and returns a few rows, is eligible for caching because of the table annotation. The second query, which is a one-time query that returns many rows, uses a hint to prevent result caching.

Lesson Agenda

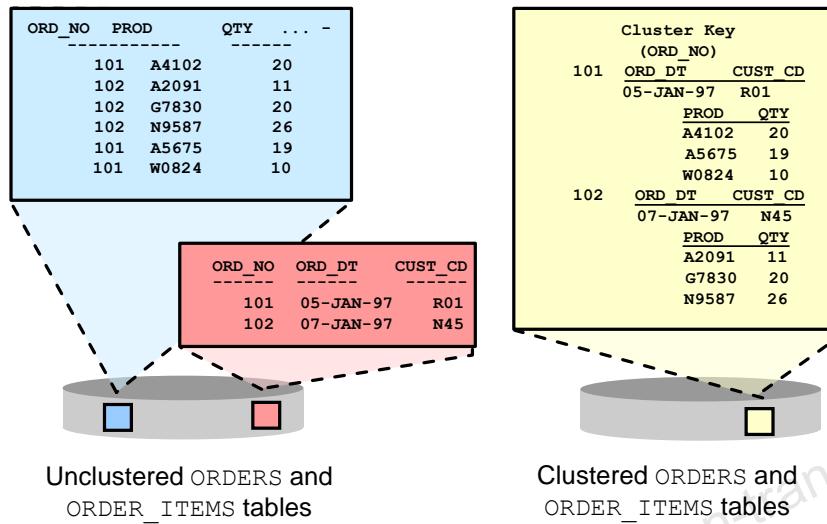
- Result Cache Operator
- Clusters
 - When Are Clusters Useful?
 - Cluster Access Path: Examples
- Sorting Operators
- Buffer Sort Operator
- Inlist Iterator
- View Operator
- Count Stop Key Operator
- Min/Max and First Row Operators
- Other N-Array Operations



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Clusters



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Clusters are an optional method for storing table data. A cluster is a group of tables that share the same data blocks because they share common columns and are often used together. For example, the ORDERS and ORDER_ITEMS tables share the ORDER_ID column. When you cluster the ORDERS and ORDER_ITEMS tables, the system physically stores all rows for each order from both the ORDERS and ORDER_ITEMS tables in the same data blocks.

Cluster index: A cluster index is an index defined specifically for a cluster. Such an index contains an entry for each cluster key value. To locate a row in a cluster, the cluster index is used to find the cluster key value, which points to the data block associated with that cluster key value. Therefore, the system accesses a given row with a minimum of two I/Os.

Hash clusters: Hashing is an optional way of storing table data to improve the performance of data retrieval. To use hashing, you create a hash cluster and load tables into the cluster. The system physically stores the rows of a table in a hash cluster and retrieves them according to the results of a hash function. The key of a hash cluster (just as the key of an index cluster) can be a single column or composite key. To find or store a row in a hash cluster, the system applies the hash function to the row's cluster key value; the resulting hash value corresponds to a data block in the cluster, which the system then reads or writes on behalf of the issued statement.

Note: Hash clusters are a better choice than an indexed table or index cluster when a table is queried frequently with equality queries.

When Are Clusters Useful?

- Index cluster:
 - Tables are always joined on the same keys.
 - The size of the table is not known.
 - It can be used in any type of search.
- Hash cluster:
 - Tables are always joined on the same keys.
 - Storage for all cluster keys is allocated initially.
 - It can be used in either equality (=) or nonequality (<>) searches.
- Single-table hash cluster:
 - This is fastest way to access a large table with an equality search.
- Sorted hash cluster:
 - This is used only for equality search.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

- Index clusters allow row data from one or more tables that share a cluster key value to be stored in same block. You can locate these rows by using a cluster index, which has one entry per cluster key value and *not* for each row. Therefore, the index is smaller and less costly to access for finding multiple rows. The rows with the same key are in a small group of blocks. This means that in an index cluster, the clustering factor is very good and provides clustering for data from multiple tables that share the same join key. The smaller index and smaller group of blocks reduce the cost of access by reducing block visits to the buffer cache. Index clusters are useful when the size of the tables is not known in advance (for example, creating a new table rather than converting an existing one whose size is stable), because a cluster bucket is created only after a cluster key value is used. They are also useful for all filter operations or searches. Note that full table scans do not perform well on a table in a multiple-table cluster because it has more blocks than the table would have if it were created as a heap table.
- Hash clusters allow row data from one or more tables that share a cluster key value to be stored in the same block. You can locate these rows by using a system- or user-provided hashing function or by using the cluster key value. The cluster key value method assumes that its value is evenly distributed so that row access is faster than with index clusters. Table rows with the same cluster key values hash into the same cluster buckets and can be stored in the same block or small group of blocks.

This means that in a hash cluster, the clustering factor is very good and a row may be accessed by its key with only one block visit and without needing an index. Hash clusters allocate all the storage for all the hash buckets when the cluster is created, so they may waste space. They also do not perform well other than on equality searches or non-equality searches. Like index clusters, if they contain multiple tables, full scans are more expensive for the same reason.

- Single-table hash clusters are similar to hash clusters, but are optimized in the block structures for access to a single table, thereby providing the fastest possible access to a row other than by using a row ID filter. Because they have only one table, full scans—if they happen—cost as much as they would in a heap table.
- Sorted hash clusters are designed to reduce the costs of accessing ordered data by using a hashing algorithm on the hash key. Accessing the first row that matches the hash key may be less costly than using an index-organized table (IOT) for a large table because it saves the cost of a B*-tree probe. All the rows that match on a particular hash key (for example, account number) are stored in the cluster in the order of the sort key or keys (for example, phone calls), thereby eliminating the need for a sort to process the ORDER BY clause. These clusters are very good for batch reporting, billing, and so on.

Cluster Access Path: Examples

```
select * from employees2 where department_id =30;
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY
SELECT STATEMENT TABLE ACCESS Access Predicates DEPARTMENT_ID=30	EMPLOYEES2	HASH	10 10

```
select * from employees3 where department_id =30;
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY
SELECT STATEMENT TABLE ACCESS INDEX Access Predicates DEPARTMENT_ID=30	EMPLOYEES3 IDX_EMP_DEPT_CLUSTER	CLUSTER UNIQUE SCAN	6 1



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

The example in the slide shows two different cluster access paths.

In the access path example at the top, a hash scan is used to locate rows in a hash cluster, based on a hash value. In a hash cluster, all rows with the same hash value are stored in the same data block. To perform a hash scan, the system first obtains the hash value by applying a hash function to a cluster key value (department_id = 30) specified by the statement. The system then scans the data blocks containing rows with that hash value.

The second access path example assumes that a cluster index was used to cluster both the EMPLOYEES and DEPARTMENTS tables. In this case, a cluster scan is used to retrieve, from a table stored in an indexed cluster, all rows that have the same cluster key value. In an indexed cluster, all rows with the same cluster key value are stored in the same data block. To perform a cluster scan, the system first obtains the ROWID of one of the selected rows (row describing department_id = 30) by scanning the cluster index. The system then locates the rows in EMPLOYEES3 based on this ROWID.

Note: You see examples of how to create clusters in the practices for this lesson.

Lesson Agenda

- Result Cache Operator
- Clusters
 - When Are Clusters Useful?
 - Cluster Access Path: Examples
- Sorting Operators
- Buffer Sort Operator
- Inlist Iterator
- View Operator
- Count Stop Key Operator
- Min/Max and First Row Operators
- Other N-Array Operations

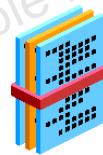


ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Sorting Operators

- **SORT operator:**
 - AGGREGATE: Retrieves a single row using a group function
 - UNIQUE: Removes duplicates
 - JOIN: Precedes a merge join
 - GROUP BY, ORDER BY: Groups and orders the rows
- **HASH operator:**
 - GROUP BY: Groups the rows
 - UNIQUE: Equivalent to SORT UNIQUE
- If you want ordered results, *always* use ORDER BY.



ORACLE

Copyright© 2018, 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 AGGREGATE does not involve a sort. It retrieves a single row that is the result of applying a group function to a group of selected rows. Operations such as COUNT and MIN are shown as SORT AGGREGATE.
- SORT UNIQUE sorts output rows to remove duplicates. It occurs if a user specifies a DISTINCT clause or if an operation requires unique values for the next step.
- SORT JOIN happens during a sort-merge join, if the rows need to be sorted by the join key.
- SORT GROUP BY is used when aggregates are computed for different groups in the data. The sort is required to separate the rows into different groups.
- SORT ORDER BY is required when the statement specifies an ORDER BY that cannot be satisfied by one of the indexes.
- HASH GROUP BY hashes a set of rows into groups for a query with a GROUP BY clause.
- HASH UNIQUE hashes a set of rows to remove duplicates. It occurs if a user specifies a DISTINCT clause or if an operation requires unique values for the next step. This is similar to SORT UNIQUE.

Note: Several SQL operators cause implicit sorts (or hashes since Oracle Database 10g, Release 2), such as DISTINCT, GROUP BY, UNION, MINUS, and INTERSECT. However, do not rely on these SQL operators to return ordered rows. If you want to have rows ordered, use the ORDER BY clause.

Lesson Agenda

- Result Cache Operator
- Clusters
 - When Are Clusters Useful?
 - Cluster Access Path: Examples
- Sorting Operators
- **Buffer Sort Operator**
- Inlist Iterator
- View Operator
- Count Stop Key Operator
- Min/Max and First Row Operators
- Other N-Array Operations



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Buffer Sort Operator

```
SELECT last_name, e.department_id, d.department_id, department_name
FROM employees e, departments d
WHERE last_name like 'A%';
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY
SELECT STATEMENT			124
MERGE JOIN		CARTESIAN	124
TABLE ACCESS	EMPLOYEES	BY INDEX ROWID BATCHED	5
INDEX	EMP_NAME_IK	RANGE SCAN	5
Access Predicates			
LAST_NAME LIKE 'A%			
Filter Predicates			
LAST_NAME LIKE 'A%			
BUFFER		SORT	27
TABLE ACCESS	DEPARTMENTS	FULL	27



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The BUFFER SORT operator uses a temporary table or a sort area in memory to store intermediate data. However, the data is not necessarily sorted.

The BUFFER SORT operator is needed if there is an operation that needs all the input data before it can start. (See “Cartesian Join.”)

So BUFFER SORT uses the buffering mechanism of a traditional sort, but it does not do the sort itself. The system simply buffers the data, in the user global area (UGA) or program global area (PGA), to avoid multiple table scans against real data blocks.

The whole sort mechanism is reused, including the swap to disk when not enough sort area memory is available, but without sorting the data.

The difference between a temporary table and a buffer sort is as follows:

- A temporary table uses system global area (SGA).
- A buffer sort uses UGA.

Lesson Agenda

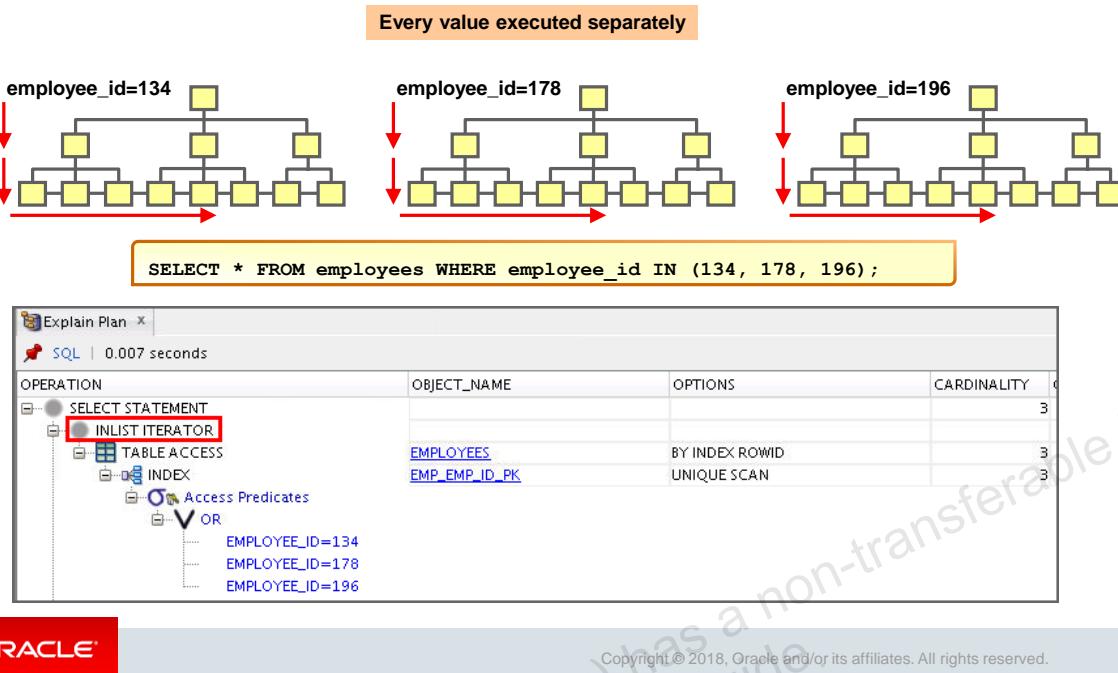
- Result Cache Operator
- Clusters
 - When Are Clusters Useful?
 - Cluster Access Path: Examples
- Sorting Operators
- Buffer Sort Operator
- **Inlist Iterator**
- View Operator
- Count Stop Key Operator
- Min/Max and First Row Operators
- Other N-Array Operations



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Inlist Iterator



It is used when a query contains an `IN` clause with values or multiple equality predicates on the same column linked with `OR`s.

The `INLIST ITERATOR` operator iterates over the enumerated value list, and every value is executed separately.

The execution plan is identical to the result of a statement with an equality clause instead of `IN`, except for one additional step. The extra step occurs when `INLIST ITERATOR` feeds the equality clause with unique values from the list.

You can view this operator as a `FOR LOOP` statement in PL/SQL. In the example in the slide, you iterate the index probe over three values: 134, 178, and 196.

Also, it is a function that uses an index, which is scanned for each value in the list. An alternative handling is `UNION ALL` of each value or a `FILTER` of the values against all the rows; this is significantly more efficient.

The optimizer uses an `INLIST ITERATOR` when an `IN` clause is specified with values, and the optimizer finds a selective index for that column. If there are multiple `OR` clauses using the same index, the optimizer selects this operation rather than `CONCATENATION` or `UNION ALL`, because it is more efficient.

Lesson Agenda

- Result Cache Operator
- Clusters
 - When Are Clusters Useful?
 - Cluster Access Path: Examples
- Sorting Operators
- Buffer Sort Operator
- Inlist Iterator
- **View Operator**
- Count Stop Key Operator
- Min/Max and First Row Operators
- Other N-Array Operations



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

View Operator

```
create view V as select /*+ NO_MERGE */ department_id, salary from employees ;
select * from V;
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY
SELECT STATEMENT			107
VIEW	V		107
TABLE ACCESS	EMPLOYEES	FULL	107

```
select v.* ,d.department_name from (select department_id, sum(salary) SUM_SAL
from employees group by department_id) v, departments d where
v.department_id=d.department_id;
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY
SELECT STATEMENT			11
MERGEJOIN			11
TABLE ACCESS	DEPARTMENTS	BY INDEX ROWID	27
INDEX	DEPT_ID_PK	FULL SCAN	27
SORT		JOIN	11
Access Predicates	V.DEPARTMENT_ID=D.DEPARTMENT_ID		
Filter Predicates	V.DEPARTMENT_ID=D.DEPARTMENT_ID		
VIEW			11
HASH		GROUP BY	11
TABLE ACCESS	EMPLOYEES	FULL	107



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Each query produces a variable set of data in the form of a table. A view simply gives a name to this set of data.

When views are referenced in a query, the system can handle them in two ways. If a number of conditions are met, they can be merged into the main query. This means that the view text is rewritten as a join with the other tables in the query. Views can also be left as stand-alone views and selected from directly as in the case of a table. Predicates can also be pushed into or pulled out of the views as long as certain conditions are met.

When a view is not merged, you can see the `VIEW` operator. The view operation is executed separately. All rows from the view are returned, and the next operation can be performed.

Sometimes a view cannot be merged and must be executed independently in a separate query block. In this case, you can also see the `VIEW` operator in the explain plan. The `VIEW` keyword indicates that the view is executed as a separate query block. For example, views containing `GROUP BY` functions cannot be merged.

The second example in the slide shows a non-mergeable inline view. An inline view is basically a query within the `FROM` clause of your statement.

Basically, this operator collects all rows from a query block before they can be processed by higher operations in the plan.

Lesson Agenda

- Result Cache Operator
- Clusters
 - When Are Clusters Useful?
 - Cluster Access Path: Examples
- Sorting Operators
- Buffer Sort Operator
- Inlist Iterator
- View Operator
- Count Stop Key Operator
- Min/Max and First Row Operators
- Other N-Array Operations



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Count Stop Key Operator

```
SELECT count(*)
FROM (SELECT /*+ NO_MERGE */ *
      FROM employees WHERE employee_id = '101' and rownum < 10);
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY
SELECT STATEMENT			1
SORT		AGGREGATE	1
VIEW		STOPKEY	1
COUNT			
Filter Predicates			
ROWNUM<10			
INDEX			
Access Predicates			
EMPLOYEE_ID=101	EMP_EMP_ID_PK	UNIQUE SCAN	1



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

COUNT STOPKEY limits the number of rows returned. The limitation is expressed by the ROWNUM expression in the WHERE clause. It terminates the current operation when the count is reached.

Note: The cost of this operator depends on the number of occurrences of the values you try to retrieve. If the value appears very frequently in the table, the count is reached quickly. If the value is very infrequent, and there are no indexes, the system has to read most of the table's blocks before reaching the count.

Lesson Agenda

- Result Cache Operator
- Clusters
 - When Are Clusters Useful?
 - Cluster Access Path: Examples
- Sorting Operators
- Buffer Sort Operator
- Inlist Iterator
- View Operator
- Count Stop Key Operator
- Min/Max and First Row Operators
- Other N-Array Operations



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Min/Max and First Row Operators

```
SELECT MIN(quantity_on_hand)
FROM INVENTORIES
WHERE quantity_on_hand < 500;
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY
SELECT STATEMENT			1
SORT		AGGREGATE	1
FIRST ROW			1
INDEX	INV_QTY_INDEX	RANGE SCAN (MIN/MAX)	1
Access Predicates	QUANTITY_ON_HAND<500		



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

FIRST ROW retrieves only the first row selected by a query. It stops accessing the data after the first value is returned. This optimization was introduced in Oracle 8*i*, and it works with the index range scan and the index full scan.

In the example in the slide, it is assumed that there is an index on the `quantity_on_hand` column.

Lesson Agenda

- Result Cache Operator
- Clusters
 - When Are Clusters Useful?
 - Cluster Access Path: Examples
- Sorting Operators
- Buffer Sort Operator
- Inlist Iterator
- View Operator
- Count Stop Key Operator
- Min/Max and First Row Operators
- Other N-Array Operations



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Other N-Array Operations

- **FILTER**
- **CONCATENATION**
- **UNION ALL/UNION**
- **INTERSECT**
- **MINUS**

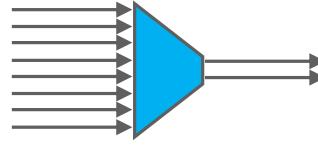


Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

You now learn about other N-Array operations.

FILTER Operations

- Accepts a set of rows
- Eliminates some of them
- Returns the rest



`SELECT department_id, sum(salary) SUM_SAL
FROM employees GROUP BY department_id
HAVING sum(salary) > 9000;`

1

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY
SELECT STATEMENT FILTER Filter Predicates SUM(SALARY)>9000 HASH TABLE ACCESS	EMPLOYEES	GROUP BY FULL	1 107

`SELECT department_id, department_name
FROM departments d WHERE NOT EXISTS (select 1
from employees e
where e.department_id=d.department_id);`

2

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY
SELECT STATEMENT MERGE JOIN TABLE ACCESS INDEX SORT Access Predicates E.DEPARTMENT_ID=D.DEPARTMENT_ID Filter Predicates E.DEPARTMENT_ID=D.DEPARTMENT_ID TABLE ACCESS	DEPARTMENTS DEPT_ID_PK EMPLOYEES	ANTI BY INDEX ROWID FULL SCAN UNIQUE FULL	17 17 27 27 107 107

ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

A FILTER operation is any operation that discards rows returned by another step, but is not involved in retrieving the rows itself. All sorts of operations can be filters, including subqueries and single table predicates.

1. In the first example, FILTER applies to the groups that are created by the GROUP BY operation.
2. In the second example, FILTER is used almost in the same way as NESTED LOOPS. DEPARTMENTS is accessed once, and for each row from DEPARTMENTS, EMPLOYEES is accessed by its index on DEPARTMENT_ID. This operation is done as many times as the number of rows in DEPARTMENTS.

The FILTER operation is applied, for each row, after the DEPARTMENTS rows are fetched. FILTER discards rows for the inner query (`select 1 from employees e where e.department_id=d. department_id`) and returns at least one row that is TRUE.

OR Expansion Operation

```
SELECT * FROM departments WHERE department_id=10
or location_id=1700;
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY
SELECT STATEMENT			21
VIEW	SYS.VW_ORE_CFB3EF8E		21
UNION-ALL			
TABLE ACCESS	DEPARTMENTS	BY INDEX ROWID	1
INDEX	DEPT_ID_PK	UNIQUE SCAN	1
Access Predicates			
DEPARTMENT_ID=10			
TABLE ACCESS	DEPARTMENTS	BY INDEX ROWID BATCHED	20
Filter Predicates			
LNNVL(DEPARTMENT_ID=10)			
INDEX	DEPT_LOCATION_IDX	RANGE SCAN	21
Access Predicates			
LOCATION_ID=1700			



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

CONCATENATION concatenates the rows returned by two or more row sets. This works like UNION ALL and does not remove duplicate rows.

In previous releases, the optimizer used the **CONCATENATION** operator to perform the OR expansion. Starting in Oracle Database 12c Release 2 (12.2), the optimizer uses the **UNION-ALL** operator instead. This enhancement provides several benefits, including enabling interaction among various transformations, and avoiding the sharing of query structures.

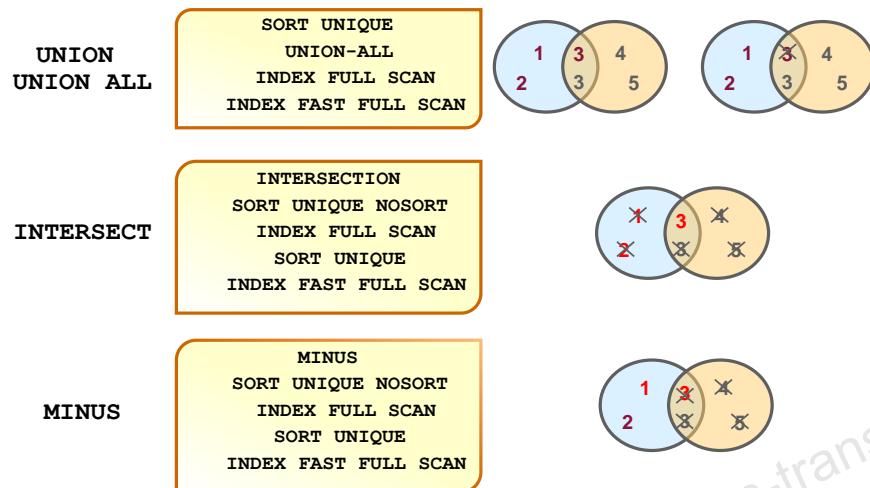
However, **OR** does not return duplicate rows, so for each component after the first, it appends a negation of the previous components (**LNNVL**).

The **LNNVL** function is generated by the **OR** clause to process this negation.

The **LNNVL ()** function returns TRUE if the predicate is NULL or FALSE.

So **filter (LNNVL (DEPARTMENT_ID=2))** returns all rows for which **DEPARTMENT_ID!=10** or **DEPARTMENT_ID** is NULL. (Because **DEPARTMENT_ID** is a primary key, it cannot be NULL).

UNION [ALL], INTERSECT, MINUS



ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

SQL handles duplicate rows with an `ALL` or `DISTINCT` modifier in different places in the language. `ALL` preserves duplicates and `DISTINCT` removes them. Here is a quick description of the possible SQL set operations:

- **INTERSECTION:** Operation accepting two sets of rows and returning the intersection of the sets, eliminating duplicates. Subrow sources are executed or optimized individually. This is very similar to sort-merge-join processing: Full rows are sorted and matched.
- **MINUS:** Operation accepting two sets of rows and returning rows appearing in the first set, but not in the second, eliminating duplicates. Subrow sources are executed or optimized individually. This is similar to `INTERSECT` processing. However, instead of match-and-return, it is match-and-exclude.
- **UNION:** Operation accepting two sets of rows and returning the union of the sets, eliminating duplicates. Subrow sources are executed or optimized individually. Rows retrieved are concatenated and sorted to eliminate duplicate rows.
- **UNION ALL:** Operation accepting two sets of rows and returning the union of the sets, and not eliminating duplicates. The expensive sort operation is not necessary. Use `UNION ALL` if you know you do not have to deal with duplicates.

Quiz



Hash clusters are a better choice than indexed tables or index clusters when a table is queried frequently with equality queries.

- a. True
- b. False



ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Answer: a

Quiz



The _____ operator uses a temporary table to store intermediate data.

- a. Inlist
- b. Min/Max
- c. Buffer Sort operator
- d. N-Array



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Answer: c

Quiz



The following query uses the _____ operator:

```
SELECT * FROM emp WHERE empno IN (7876, 7900, 7902);
```

- a. Buffer Sort operator
- b. Inlist
- c. Min/Max
- d. N-Array



ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Answer: b

Quiz



A FILTER operation retrieves the rows returned by another statement.

- a. True
- b. False



ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Answer: b

A filter operation filters the rows retrieved from another statement.

Summary

In this lesson, you should have learned to:

- Describe SQL operators for:
 - Clusters
 - In-List
 - Sorts
 - Filters
 - Set Operations
- Describe result cache operators



ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Practice 10: Overview

This practice covers the following topics:

- Using the Result Cache
- Using Different Access Paths for Better Optimization (case 13 to case 15) (Optional)



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2019, Oracle and/or its affiliates.

GANG LIU (gangl@baylorhealth.edu) has a non-transferable license
to use this Student Guide.

Introduction to Optimizer Statistics Concepts

The ORACLE logo, consisting of the word "ORACLE" in white capital letters on a red rectangular background.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe optimizer statistics
 - Table statistics
 - Index statistics
 - Column statistics (histogram)
 - Column statistics (extended statistics)
 - Session-specific statistics for global temporary tables
 - System statistics
- Gather and manage optimizer statistics



ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

This lesson explains why statistics are important for the query optimizer and how to gather and use optimizer statistics.

Lesson Agenda

- Optimizer Statistics
 - Table Statistics
 - Index Statistics
 - Index Clustering Factor
 - Column Statistics
- Column Statistics: Histograms
- Column Statistics: Extended Statistics
- Session-Specific Statistics for Global Temporary Tables
- System Statistics
- Gathering and Managing Optimizer Statistics

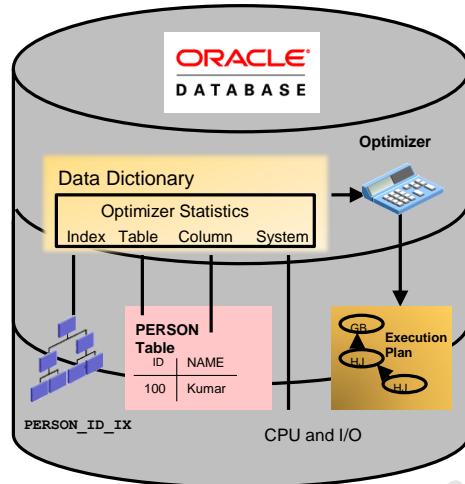


ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Optimizer Statistics

Describe the database and the objects in the database



ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Optimizer statistics provide details about a database and the objects in the database. These statistics are used by the query optimizer to select the best execution plan for each SQL statement. The information used by the query optimizer to estimate is as follows:

- Selectivity of predicates
- Cost of each execution plan
- Access method, join order, and join method
- CPU and I/O costs

Because the objects in a database change constantly, statistics must be regularly updated so that they accurately describe these database objects. Statistics are maintained automatically by Oracle Database, or you can maintain the optimizer statistics manually by using the `DBMS_STATS` package.

- Table statistics
- Index statistics
- Column statistics
- System statistics

Note: The statistics mentioned in this slide are optimizer statistics, which are created for query optimization and are stored in the data dictionary. These statistics should not be confused with performance statistics that are visible through `V$` views.

Table Statistics (USER_TAB_STATISTICS)

- Used to determine:
 - Table access cost
 - Join cardinality
 - Join order
- Example:

The screenshot shows the Oracle SQL Developer interface. In the Worksheet tab, a SQL query is written:

```
SELECT NUM_ROWS, AVG_ROW_LEN, BLOCKS, LAST_ANALYZED
FROM USER_TAB_STATISTICS
WHERE TABLE_NAME = 'CUSTOMERS';
```

In the Query Result tab, the output of the query is displayed in a table:

	NUM_ROWS	AVG_ROW_LEN	BLOCKS	LAST_ANALYZED
1	55500	189	1551	31-JAN-18

ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In Oracle Database, table statistics include information about rows and blocks. The optimizer uses these statistics to determine the cost of table scans and table joins. DBMS_STATS can gather statistics for both permanent and temporary tables. Some of the table statistics gathered are:

NUM_ROWS

This statistic is the basis for cardinality computations. Row count is especially important if the table is the driving table of a nested loops join, because it defines how many times the inner table is probed.

BLOCKS

This statistic indicates the number of used data blocks. Block count in combination with DB_FILE_MULTIBLOCK_READ_COUNT gives the base table access cost.

AVG_ROW_LEN

This statistic is the average length of a row in the table in bytes.

STALE_STATS

This statistic tells you if statistics are valid on the corresponding table.

Note: There are three other statistics (EMPTY_BLOCKS, AVE_ROW_LEN, and CHAIN_CNT) that are not used by the optimizer and are not gathered by the DBMS_STATS procedures. If they are required, the ANALYZE command must be used.

Index Statistics (USER_IND_STATISTICS)

- The index statistics include information about the number of index levels, the number of index blocks, and the relationship between the index and the data blocks.
- They are used to decide between:
 - Full table scan and index scan

The screenshot shows the Oracle SQL Developer interface. In the top pane, there is a SQL query:

```
SELECT INDEX_NAME, BLEVEL, LEAF_BLOCKS AS "LEAFBLK", DISTINCT_KEYS AS "DIST_KEY",
       AVG(LEAF_BLOCKS.PER_KEY) AS "LEAFBLK_PER_KEY",
       AVG(DATA_BLOCKS.PER_KEY) AS "DATABLK_PER_KEY"
  FROM USER_IND_STATISTICS
 WHERE INDEX_NAME = 'CUSTOMERS_PK';
```

In the bottom pane, the results are displayed in a table:

INDEX_NAME	LEVEL	LEAFBLK	DIST_KEY	LEAFBLK_PER_KEY	DATABLK_PER_KEY
CUSTOMERS_PK	1	115	5500	1	1

ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In general, to select an index access, the optimizer requires a predicate on the prefix of the index columns. However, in case there is no predicate and all the columns referenced in the query are present in an index, the optimizer considers using a full index scan versus a full table scan. Index statistics stored in the `USER_IND_STATISTICS` view track the following:

BLEVEL

This statistic is used to calculate the cost of leaf block lookups. It indicates the depth of the index from its root block to its leaf blocks. A depth of “0” indicates that the root block and leaf block are the same.

LEAF_BLOCKS

This statistic is used to calculate the cost of a full index scan.

CLUSTERING_FACTOR

This statistic measures the order of the rows in the table based on the values of the index. If the value is near the number of blocks, the table is very well ordered. In this case, the index entries in a single leaf block tend to point to the rows in the same data blocks. If the value is near the number of rows, the table is very randomly ordered. In this case, it is unlikely that the index entries in the same leaf block point to rows in the same data blocks.

DISTINCT_KEYS

This statistic is the number of distinct indexed values. For indexes that enforce the `UNIQUE` and `PRIMARY KEY` constraints, this value is the same as the number of rows in the table.

`AVG_LEAF_BLOCKS_PER_KEY`

This statistic is the average number of leaf blocks in which each distinct value in the index appears, rounded to the nearest integer. For indexes that enforce the `UNIQUE` and `PRIMARY KEY` constraints, this value is always 1.

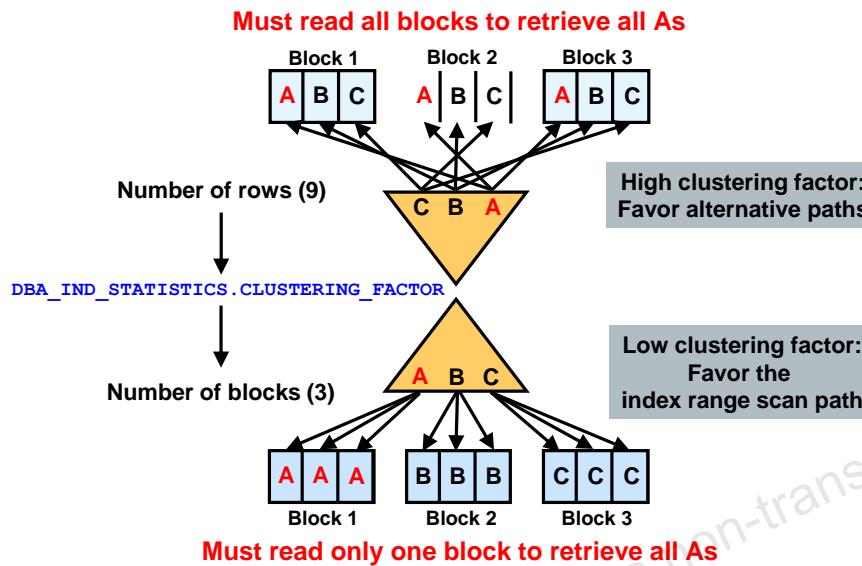
`AVG_DATA_BLOCKS_PER_KEY`

This statistic is the average number of data blocks in the table that are pointed to by a distinct value in the index, rounded to the nearest integer. This statistic is the average number of data blocks that contain rows with a given value for the indexed columns.

`NUM_ROWS`

This statistic is the number of rows in the index.

Index Clustering Factor



ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The system performs I/O by blocks. Therefore, the optimizer's decision to use full table scans is influenced by the percentage of blocks accessed, not rows.

When an index range scan is used, each selected index entry points to a block in the table. If each entry points to a different block, the accessed rows and accessed blocks are the same.

Consequently, the desired number of rows could be clustered together in a few blocks, or they could be spread out over a larger number of blocks. This is called the index clustering factor.

The cost formula of an index range scan uses the level of the B*-tree, the number of leaf blocks, the index selectivity, and the clustering factor. A clustering factor indicates that the individual rows are concentrated within fewer blocks in the table. A high clustering factor indicates that the individual rows are scattered more randomly across the blocks in the table.

Therefore, a high clustering factor means that it costs more to use an index range scan to fetch rows by `ROWID`, because more blocks in the table need to be visited to return the data. In a real-life scenarios, it appears that the clustering factor plays an important role in determining the cost of an index range scan, simply because the number of leaf blocks and the height of the B*-tree are relatively small compared to the clustering factor and the table's selectivity.

Note: If you have more than one index on a table, the clustering factor for one index might be small, whereas the clustering factor for another index might be large at the same time. An attempt to reorganize the table to improve the clustering factor for one index can cause degradation of the clustering factor for the other index.

The clustering factor is computed and stored in the `CLUSTERING_FACTOR` column of the `DBA_INDEXES` view when you gather statistics on the index. The way it is computed is relatively easy. You read the index from left to right, and for each indexed entry, you add one to the clustering factor if the corresponding row is located in a different block than the one from the previous row. Based on this algorithm, the smallest possible value for the clustering factor is the number of blocks, and the highest possible value is the number of rows.

The example in the slide shows how the clustering factor can affect cost. Assume the following situation: There is a table with nine rows; there is a non-unique index on `col1` for the table; the `col1` column currently stores the values A, B, and C; the table has only three data blocks.

- **Case 1:** If the same rows in the table are arranged so that the index values are scattered across the table blocks (rather than collocated), the index clustering factor is high.
- **Case 2:** The index clustering factor is low for the rows because they are collocated in the same block for the same value.

Note: For bitmap indexes, the clustering factor is not applicable and is not used.

Column Statistics (USER_TAB_COL_STATISTICS)

- Column statistics track information about column values and data distribution.
- The optimizer uses these statistics to generate accurate cardinality estimates and make better decisions about index usage, join orders, join methods, and so on.

The screenshot shows the Oracle SQL Developer interface. In the Worksheet tab, a SQL query is written:

```
SELECT COLUMN_NAME, NUM_DISTINCT, NUM_NULLS, NUM_BUCKETS, DENSITY
FROM USER_TAB_COL_STATISTICS
WHERE TABLE_NAME = 'CUSTOMERS'
ORDER BY COLUMN_NAME;
```

In the Query Result tab, the output is displayed in a table:

COLUMN_NAME	NUM_DISTINCT	NUM_NULLS	NUM_BUCKETS	DENSITY
1 COUNTRY_ID	19	0	1	0.0526315789473684
2 CUST_CITY	620	0	1	0.00161290322580645
3 CUST_CITY_ID	620	0	1	0.00161290322580645
4 CUST_CREDIT_LIMIT	8	0	1	0.125

ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Column statistics track information about column values and data distribution.

The optimizer uses these statistics to generate accurate cardinality estimates and make better decisions about index usage, join orders, join methods, and so on.

Index statistics in USER_TAB_COL_STATISTICS track the following:

- **NUM_DISTINCT** is used in selectivity calculations; for example, 1/Number of Distinct Values.
- **LOW_VALUE** and **HIGH_VALUE**: The cost-based optimizer (CBO) assumes uniform distribution of values between low and high values for all data types. These values are used to determine range selectivity.
- **NUM_NULLS** helps with selectivity of nullable columns and the **IS NULL** and **IS NOT NULL** predicates.
- **DENSITY** is relevant only for histograms. It is used as the selectivity estimate for nonpopular values. It can be thought of as the probability of finding one particular value in this column. The calculation depends on the histogram type.
- **NUM_BUCKETS** is the number of buckets in the histogram for the column.
- **HISTOGRAM** indicates the existence or type of the histogram: NONE, FREQUENCY, HEIGHT, and BALANCED.

Lesson Agenda

- Optimizer Statistics
 - Table Statistics
 - Index Statistics
 - Index Clustering Factor
 - Column Statistics
- **Column Statistics: Histograms**
- Column Statistics: Extended Statistics
- Session-Specific Statistics for Global Temporary Tables
- System Statistics
- Gathering and Managing Optimizer Statistics

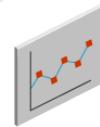


ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Column Statistics: Histograms

- The optimizer assumes uniform distributions; this may lead to suboptimal access plans in the case of data skew.
- Histograms:
 - Store additional column distribution information
 - Give better selectivity estimates in the case of non-uniform distributions
- Types of histograms:
 - Frequency
 - Top-Frequency
 - Height-Balanced (legacy)
 - Hybrid
- They are stored in DBA_TAB_HISTOGRAMS.



ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

A histogram is a special type of column statistic that provides more detailed information about the data distribution in a table column. A histogram sorts values into “buckets,” as you might sort coins into buckets. Because a histogram captures the distribution of different values in a column, it yields better selectivity estimates. Having histograms on columns that contain skewed data or values with large variations in the number of duplicates helps the query optimizer generate good selectivity estimates and make better decisions about index usage, join orders, and join methods.

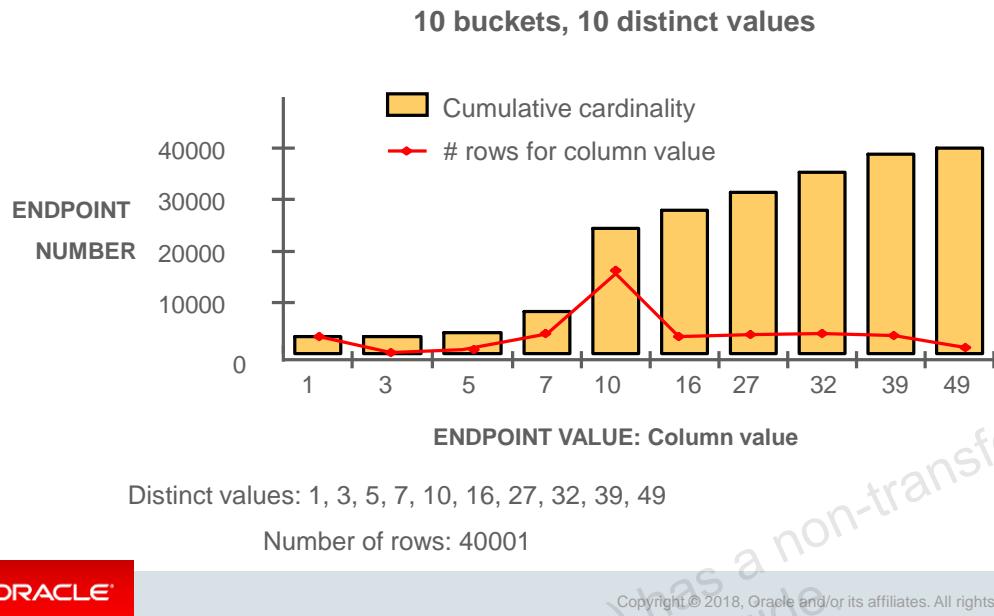
Without histograms, uniform distribution is assumed. If a histogram is available on a column, the estimator uses it instead of the number of distinct values. Creation of histograms is controlled by the METHOD_OPT parameter.

When creating histograms, Oracle Database uses different types of histogram representations, depending on the number of distinct values found in the corresponding column. When you have a data set with less than 254 distinct values, and the number of histogram buckets is not specified, the system creates a frequency histogram. If the number of distinct values is greater than the required number of histogram buckets, the system creates a height-balanced histogram.

You can find information about histograms in these dictionary views: DBA_TAB_HISTOGRAMS, DBA_PART_HISTOGRAMS, and DBA_SUBPART_HISTOGRAMS.

Note: Gathering histogram statistics is the most resource-consuming operation in gathering statistics.

Frequency Histograms



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

For the example in the slide, assume that you have a column that is populated with 40,001 numbers. You have only 10 distinct values: 1, 3, 5, 7, 10, 16, 27, 32, 39, and 49. Value 10 is the most popular value with 16,293 occurrences.

When the requested number of buckets equals (or is greater than) the number of distinct values, you can store each different value and record exact cardinality statistics. In this case, in DBA_TAB_HISTOGRAMS, the ENDPOINT_VALUE column stores the column value and the ENDPOINT_NUMBER column stores the cumulative row count, including the column value, because this can avoid some calculation for range scans. The actual row counts are derived from the endpoint values if needed. The actual number of row counts is shown by the curve in the slide for clarity; only the ENDPOINT_VALUE and ENDPOINT_NUMBER columns are stored in the data dictionary.

Viewing Frequency Histograms

```

SELECT country_subregion_id, count(*)
FROM sh.countries GROUP BY country_subregion_id
ORDER BY 1;

BEGIN DBMS_STATS.GATHER_TABLE_STATS ( ownname => 'SH' , tabname => 'COUNTRIES' , method_opt => 'FOR COLUMNS
COUNTRY_SUBREGION_ID' );
END;

SELECT TABLE_NAME, COLUMN_NAME, NUM_DISTINCT, NUM_BUCKETS, HISTOGRAM FROM USER_TAB_COL_STATISTICS WHERE
TABLE_NAME='COUNTRIES'
AND COLUMN_NAME='COUNTRY_SUBREGION_ID';

TABLE_NAME      COLUMN_NAME      NUM_DISTINCT      NUM_BUCKETS      HISTOGRAM
-----          -----          -----          -----
COUNTRIES      COUNTRY_SUBREGION_ID      8          8      FREQUENCY

SELECT ENDPOINT_NUMBER, ENDPOINT_VALUE
FROM USER_HISTOGRAMS
WHERE TABLE_NAME='COUNTRIES'
AND COLUMN_NAME='COUNTRY_SUBREGION_ID';

ENDPOINT_NUMBER ENDPOINT_VALUE
-----
1              52792
6              52793
8              52794
...

```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The example in the slide shows you how to view a frequency histogram. Here, you want to generate a frequency histogram on the sh.countries.country_subregion_id column. This table has 23 rows.

The first query shows that the country_subregion_id column contains eight distinct values that are unevenly distributed.

Because the number of distinct values in the country_subregion_id column of the COUNTRIES table is eight, and the number of requested buckets defaults to 254, the system automatically creates a frequency histogram with eight buckets. You can view this information in the USER_TAB_COL_STATISTICS view.

To view the histogram itself, you can query the USER_HISTOGRAMS view. You can see the ENDPOINT_NUMBER column that corresponds to the cumulative frequency of the corresponding ENDPOINT_VALUE column, which represents, in this case, the actual value of the column data.

For 52793, the endpoint number 6 indicates that the value appears 5 times (6 - 1).

For 52794, the endpoint number 8 indicates that the value appears 2 times (8 - 6).

Note: The DBMS_STATS package is covered later in the lesson.

Top Frequency Histogram

- Traditionally, a frequency histogram is created only if NDV < 254.
- If a column contains more than 254 distinct values, and if a small number of distinct values occupy more than 99% of the data, the database creates a Top Frequency histogram by using the small number of extremely popular distinct values.
- Creating a frequency histogram on that small set of values is very useful even though NDV is greater than 254.
- The unpopular values are ignored to create a better quality histogram for popular values.
- This is created only with `AUTO_SAMPLE_SIZE`.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Before Oracle Database 12c, a frequency histogram was created only if the number of distinct values (NDV) was less than 254. Starting with Oracle Database 12c, if a column contains more than 254 distinct values, and if a small number of distinct values occupy more than 99 percent of the data, the database creates a Top Frequency histogram by using the small number of extremely popular distinct values. It is built using the same technique that is used for frequency histograms.

A Top Frequency histogram can produce a better histogram for highly popular values by ignoring the statistically insignificant unpopular values.

A Top Frequency histogram is indicated by the `TOP-FREQUENCY` value in the `USER_TAB_COL_STATISTICS.HISTOGRAM` column.

In Oracle Database 12c, the database creates Top Frequency histograms when the following conditions are true:

- The sampling percentage is `AUTO`. In this release, the database constructs frequency histograms from a full table scan when the sampling size is `AUTO` (which is the default). For all other sampling percentage specifications, the database derives frequency histograms from a sample.
- The number of distinct values in the data set is n .
- The percentage of rows occupied by the top n frequent values is equal to or greater than threshold p , where p is $(1-(1/n)) * 100$.

Viewing Top Frequency Histograms

```
SELECT country_subregion_id, count(*)
FROM sh.countries GROUP BY country_subregion_id
ORDER BY 1;
```

```
BEGIN DBMS_STATS.GATHER_TABLE_STATS ( ownname => 'SH' , tabname => 'COUNTRIES' , method_opt => 'FOR COLUMNS
COUNTRY_SUBREGION_ID SIZE 7' ); END;
```

```
SELECT TABLE_NAME, COLUMN_NAME, NUM_DISTINCT, NUM_BUCKETS, HISTOGRAM FROM USER_TAB_COL_STATISTICS WHERE
TABLE_NAME='COUNTRIES'
AND COLUMN_NAME='COUNTRY_SUBREGION_ID';

TABLE_NAME      COLUMN_NAME          NUM_DISTINCT    NUM_BUCKETS   HISTOGRAM
-----          -----              -----          -----
COUNTRIES      COUNTRY_SUBREGION_ID      8            7        TOP-FREQUENCY
```

```
SELECT ENDPOINT_NUMBER, ENDPOINT_VALUE
FROM USER_HISTOGRAMS
WHERE TABLE_NAME='COUNTRIES'
AND COLUMN_NAME='COUNTRY_SUBREGION_ID';

ENDPOINT_NUMBER ENDPOINT_VALUE
-----
1                  52792
6                  52793
8                  52794
...
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

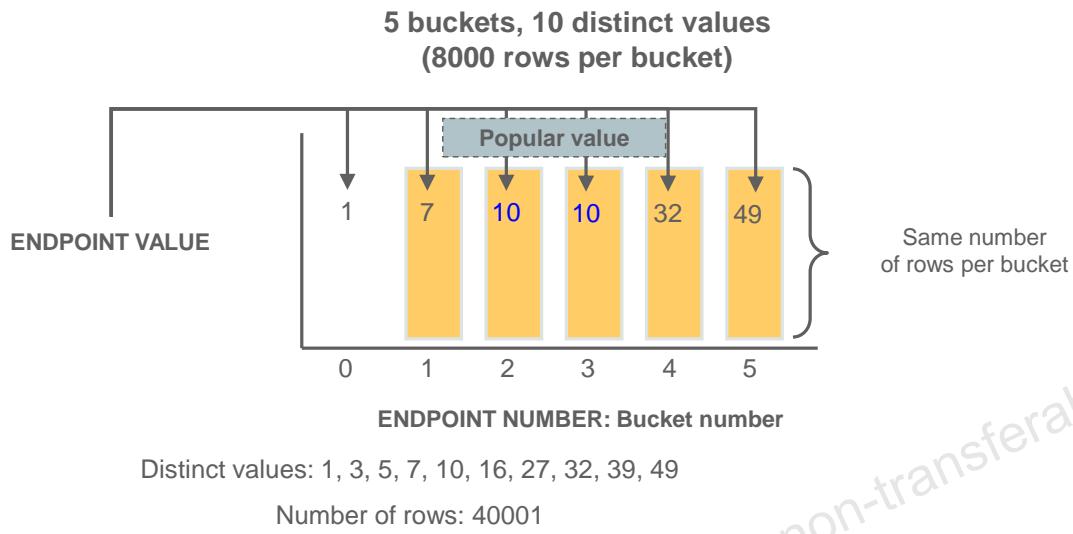
The example in the slide shows you how to view a Top Frequency histogram. Here, you want to generate a frequency histogram on the `sh.countries.country_subregion_id` column. This table has 23 rows.

The first query shows that the `country_subregion_id` column contains eight distinct values that are unevenly distributed.

Because the number of distinct values in the `country_subregion_id` column of the COUNTRIES table is eight and the number of requested buckets seven, the top seven most frequent values occupy 95.6% of the rows, which exceeds the threshold of 85.7%, thus generating a Top Frequency histogram.

When you query the endpoint number and endpoint value for the column, you see that each distinct value has its own bucket, except 52795, which is excluded from the histogram because it is nonpopular and statistically insignificant.

Height-Balanced Histograms



ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In a height-balanced histogram, the ordered column values are divided into bands so that each band contains approximately the same number of rows. The histogram tells you values of the endpoints of each band. For the example in the slide, assume that you have a column that is populated with 40,001 numbers. There will be 8,000 values in each band. You have only 10 distinct values: 1, 3, 5, 7, 10, 16, 27, 32, 39, and 49. Value 10 is the most popular value with 16,293 occurrences. When the number of buckets is less than the number of distinct values, `ENDPOINT_NUMBER` records the bucket number and `ENDPOINT_VALUE` records the column value that corresponds to this endpoint. In the example, the number of rows per bucket is one-fifth of the total number of rows; that is 8,000. Based on this assumption, value 10 appears between 8,000 and 24,000 times. So you are sure that value 10 is a popular value.

Before Oracle Database 12c Release 1 (12.1), the database created a height-balanced histogram when the NDV was greater than n .

This type of histogram is good for equality predicates on popular values and for range predicates. The number of rows per bucket is not recorded because it can be derived from the total number of values and the fact that all the buckets contain an equal number of values.

In this example, value 10 is a popular value because it spans multiple endpoint values. To save space, the histogram does not actually store duplicated buckets. For the example in the slide, bucket 2 (with endpoint value 10) would not be recorded in `DBA_TAB_HISTOGRAMS` for that reason.

Viewing Height-Balanced Histograms

```
BEGIN
  DBMS_STATS.gather_table_stats(OWNNAME =>'OE', TABNAME=>'INVENTORIES',
  METHOD_OPT => 'FOR COLUMNS SIZE 10 QUANTITY_ON_HAND', ESTIMATE_PERCENT=>100);
END;
```

```
SELECT column_name, num_distinct, num_buckets, histogram
  FROM USER_TAB_COL_STATISTICS
 WHERE table_name = 'INVENTORIES' AND column_name = 'QUANTITY_ON_HAND';

COLUMN_NAME          NUM_DISTINCT NUM_BUCKETS HISTOGRAM
-----              -----        -----
QUANTITY_ON_HAND      237           10 HEIGHT BALANCED
```

```
SELECT endpoint_number, endpoint_value
  FROM USER_HISTOGRAMS
 WHERE table_name = 'INVENTORIES' and column_name = 'QUANTITY_ON_HAND'
 ORDER BY endpoint_number;

ENDPOINT_NUMBER ENDPOINT_VALUE
-----          -----
0                  0
1                  27
2                  42
3                  57
...
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

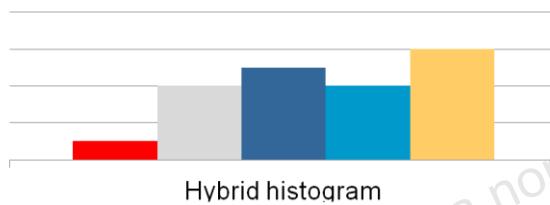
The example in the slide shows you how to view a height-balanced histogram. Because the number of distinct values in the QUANTITY_ON_HAND column of the INVENTORIES table is 237; the number of requested buckets is 10; and the estimate_percent is set to a nondefault value, the system creates a height-balanced histogram with 10 buckets.

To view the histogram itself, you can query the USER_HISTOGRAMS view. You can see that ENDPOINT_NUMBER corresponds to the bucket number and ENDPOINT_VALUE corresponds to values of the endpoint.

Note: To simulate Oracle Database 11g behavior, which is necessary to create height-based histograms, set estimate_percent to a nondefault value. If you specify a nondefault percentage, the database creates frequency or height-balanced histograms.

Hybrid Histograms

- These are similar to height-balanced histograms and are created if the NDV >254.
- They store the actual frequencies of bucket endpoints in histograms.
- No values are allowed to spill over multiple buckets.
- More endpoint values can be squeezed in a histogram.
- The same effect is achieved as increasing the number of buckets.
- They are created only with `AUTO_SAMPLE_SIZE`.



ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

A hybrid histogram combines the characteristics of both height-based histograms and frequency histograms. It is created if the number of distinct values in a column (NDV) is greater than 254 values.

The height-based histogram sometimes produces inaccurate estimates for values that are almost popular. For example, a value that occurs as an endpoint value of only one bucket but almost occupies two buckets is not considered as popular.

To solve this problem, a hybrid histogram stores the endpoint repeat count value, which is the number of times the endpoint value is repeated, for each endpoint in the histogram. By using the repeat count, the optimizer can obtain accurate estimates for almost popular values.

In Oracle Database 12c, the database creates hybrid histograms when the following conditions are true:

- The sampling percentage is `AUTO`.
 - If you specify your own percentage, the database creates frequency or height-balanced histograms.
- The criteria for Top Frequency histograms do not apply.
- n is less than NDV , where n is the user-specified number of buckets. If no number is specified, n defaults to 254.

Viewing Hybrid Histograms

```

BEGIN
  DBMS_STATS.gather_table_STATS(OWNNAME =>'OE', TABNAME=>'INVENTORIES',
  METHOD_OPT => 'FOR COLUMNS SIZE 10 QUANTITY_ON_HAND');
END;

```

```

SELECT column_name, num_distinct, num_buckets, histogram
FROM USER_TAB_COL_STATISTICS
WHERE table_name = 'INVENTORIES' AND column_name = 'QUANTITY_ON_HAND';

COLUMN_NAME          NUM_DISTINCT NUM_BUCKETS HISTOGRAM
-----              -----        -----
QUANTITY_ON_HAND      237           10 HYBRID

```

```

SELECT endpoint_number, endpoint_value
FROM USER_HISTOGRAMS
WHERE table_name = 'INVENTORIES' and column_name = 'QUANTITY_ON_HAND'
ORDER BY endpoint_number;

ENDPOINT_NUMBER ENDPOINT_VALUE
-----          -----
    0            0
  129          29
  254          46
  383          63
...

```

ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The example in the slide shows you how to view a hybrid histogram. Because the number of distinct values in the QUANTITY_ON_HAND column of the INVENTORIES table is 237, and the number of requested buckets is 10, which is less than 237, the optimizer cannot create a frequency histogram. The optimizer considers both hybrid and top frequency histograms.

To qualify for a top frequency histogram, the percentage of rows occupied by the top 10 most frequent values must be equal to or greater than threshold p , where p is $(1-(1/10))^*100$, or 90%. However, in this case, the top 10 most frequent values is less than the threshold p . Therefore, the optimizer chooses a hybrid histogram because the criteria for a top frequency histogram do not apply.

Note: If no sampling percentage is specified, Oracle Database 12c no longer creates height-balanced histograms. If you upgrade the database from Oracle Database 11g to Oracle Database 12c, any height-based histograms created before the upgrade remain in use. If Oracle Database 12c creates new histograms, and if the sampling percentage is AUTO_SAMPLE_SIZE, the histograms are either top frequency or hybrid, but not height-balanced.

Best Practices: Histogram

- Histograms are useful when you have a high degree of skew in column distribution.
- Histograms are *not* useful for:
 - Columns that do not appear in the WHERE or JOIN clause
 - Columns with uniform distributions
 - Equality predicates with unique columns
- The maximum number of buckets is the least (254, # distinct values). If possible, frequency histograms are preferred.
- Do not use histograms unless they substantially improve performance.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Histograms are useful only when they reflect the current data distribution of a given column. The data in the column can change as long as the distribution remains constant. If the data distribution of a column changes frequently, you must recompute its histogram frequently.

Histograms are useful when you have a high degree of data skew in the columns for which you want to create histograms.

However, there is no need to create histograms for columns that do not appear in the WHERE clause of a SQL statement. Similarly, there is no need to create histograms for columns with uniform distribution.

In addition, for columns that are declared as UNIQUE, histograms are useless because the selectivity is obvious. Also, the maximum number of buckets is 254, which can be lower depending on the actual number of distinct column values. If possible, frequency histograms are preferred. Histograms can affect performance and should be used only when they substantially improve query plans. For uniformly distributed data, the optimizer can make fairly accurate guesses about the cost of executing a particular statement without the use of histograms.

Note: Character columns have some exceptional behavior because histogram data is stored only for the first 32 bytes of any string.

Best Practices: Histograms

- Set `METHOD_OPT` to `FOR ALL COLUMNS AUTO`.
- Use `TRUNCATE` instead of dropping and re-creating the same table if you need to remove all the rows from a table.
- If incorrect cardinality or selectivity is observed in an execution plan, check to see if a histogram can resolve the problem.
- Make sure that statistics for objects are collected at the highest sample size you can afford and see if the plan improves.
- In earlier releases, if a query uses binds, or binds are not representative of future executions, do not consider histograms to avoid bind peeking. Starting from Oracle Database 11g, adaptive cursor sharing resolves bind/histogram issues.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

When gathering statistics, histograms are specified by using the `METHOD_OPT` argument of the `DBMS_STATS` gathering procedures. Oracle recommends setting `METHOD_OPT` to `FOR ALL COLUMNS SIZE AUTO`. With this setting, Oracle Database automatically determines which columns require histograms and the number of buckets (size) for each histogram. Column usage history is collected and monitored at `sys.col_usage$`. This information is needed by `dbms_stats` to identify candidate columns on which to build histograms when `METHOD_OPT` is set to `FOR ALL COLUMNS SIZE AUTO`.

Note: When upgrading, in some cases, the effect of a histogram is adverse to the generation of a better plan (especially in the presence of bind variables combined with small or `AUTO` sample sizes). Again, you may want to initially set this parameter to its release value before the upgrade, and later adjust to your release default value after the upgrade.

If you need to remove all the rows from a table when using `DBMS_STATS`, use `TRUNCATE` instead of dropping and re-creating the same table. When you drop a table, workload information used by the auto-histogram gathering feature and saved statistics history used by the `RESTORE_*_STATS` procedures are lost. Without this data, these features do not function properly.

Note: Statistics gathering is covered in Appendix F titled *Gathering and Managing Optimizer Statistics*.

If incorrect cardinality or selectivity is observed in an execution plan, check to see if a histogram can resolve the problem.

Make sure that statistics for objects are collected at the highest sample size you can afford and see if the plan improves. The default sample size is 100 percent.

Lesson Agenda

- Optimizer Statistics
 - Table Statistics
 - Index Statistics
 - Index Clustering Factor
 - Column Statistics
- Column Statistics: Histograms
- Column Statistics: Extended Statistics
- Session-Specific Statistics for Global Temporary Tables
- System Statistics
- Gathering and Managing Optimizer Statistics



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Column Statistics: Extended Statistics

- The optimizer estimates selectivity on *Highly Correlated Column Predicates* poorly:
 - Columns have values that are highly correlated.
 - Actual selectivity is often much lower or higher than the optimizer estimates, for example:

```
WHERE cust_state_province = 'CA'  
AND country_id=52775;
```
- The optimizer estimates *Expression on Columns* poorly:
 - WHERE upper(model)='MODEL'
 - When a function is applied to a column in the WHERE clause, the optimizer has no way of knowing how that function affects the selectivity of the column.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Extended statistics enable the gathering of statistics on a group of columns within a table as a whole, providing the optimizer with more information about any correlation that may exist between the columns. DBMS_STATS enables you to collect extended statistics, which are statistics that can improve cardinality estimates when multiple predicates exist on different columns of a table.

The query optimizer takes into account the correlation between columns when computing the selectivity of multiple predicates in the following limited cases:

- If all the columns of a conjunctive predicate match all the columns of a concatenated index key and the predicates are equalities used in equijoins, the optimizer uses the number of distinct keys (NDK) in the index for estimating selectivity, as $1/NDK$.
- When DYNAMIC_STATISTICS is set to level 4, the query optimizer uses dynamic statistics to estimate the selectivity of complex predicates that involve several columns from the same table. However, the sample size is very small and increases parsing time. As a result, the sample is likely to be statistically inaccurate and may cause more harm than good.

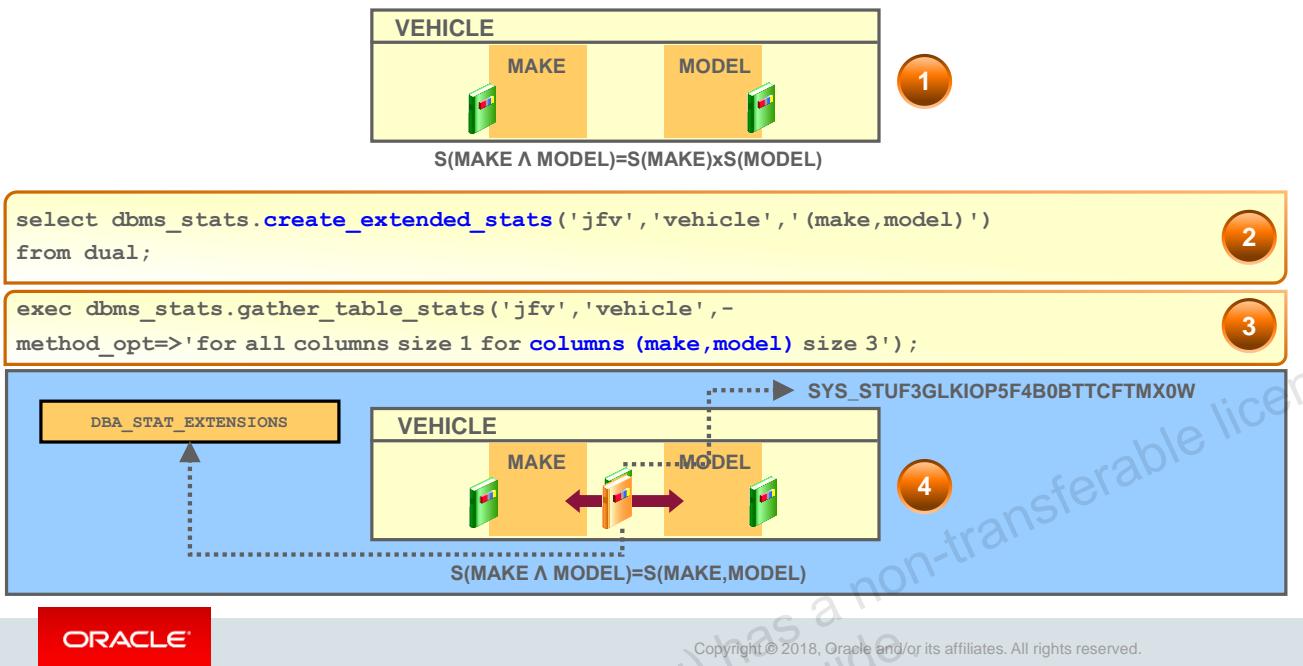
In all other cases, the optimizer assumes that the values of columns used in a complex predicate are independent of each other. It estimates the selectivity of a conjunctive predicate by multiplying the selectivity of individual predicates. This approach results in underestimation of the selectivity if there is a correlation between the columns.

To circumvent this issue, Oracle Database allows you to collect, store, and use the following statistics to capture functional dependency between two or more columns (also called groups of columns): number of distinct values, number of nulls, frequency histograms, and density.

When a function is applied to a column in the WHERE clause of a query (function(col1)=constant), the optimizer has no way of knowing how that function affects the selectivity of the column. The optimizer assumes a static selectivity value of 1 percent. This approach almost never has the correct selectivity, and it may cause the optimizer to produce suboptimal plans. By gathering expression statistics on the expression function(col1), the optimizer obtains a more accurate selectivity value.

Oracle Database gathers statistics on a group of columns within a table or an expression on a column to obtain a more accurate selectivity value.

Column Group Statistics



In the slide example, consider a **VEHICLE** table in which you store information about cars. The **MAKE** and **MODEL** columns are highly correlated, in that **MODEL** determines **MAKE**. This is a strong dependency and both columns should be considered by the optimizer as highly correlated. You can signal that correlation to the optimizer by using the **CREATE_EXTENDED_STATS** function, and then compute the statistics for all columns (including the ones for the correlated groups that you created).

Starting in this release, Oracle Database automatically determines the column groups that are required in a given workload or SQL tuning set (STS), and then creates the column groups. Thus, for any given workload, you no longer need to know which columns from each table will be used together as a workload. By monitoring a workload, Oracle Database 12c records the necessary column groups. You can then create the column groups by executing **DBMS_STATS.CREATE_EXTENDED_STATS**.

You can use **DBMS_STATS.SEED_COL_USAGE** and **REPORT_COL_USAGE** to determine the column groups that are required for a table based on a specified workload. This technique is useful when you do not know what extended statistics to create. This technique does not work for expression statistics.

The optimizer uses column group statistics for equality predicates, inlist predicates, and for estimating the group by cardinality.

Notes

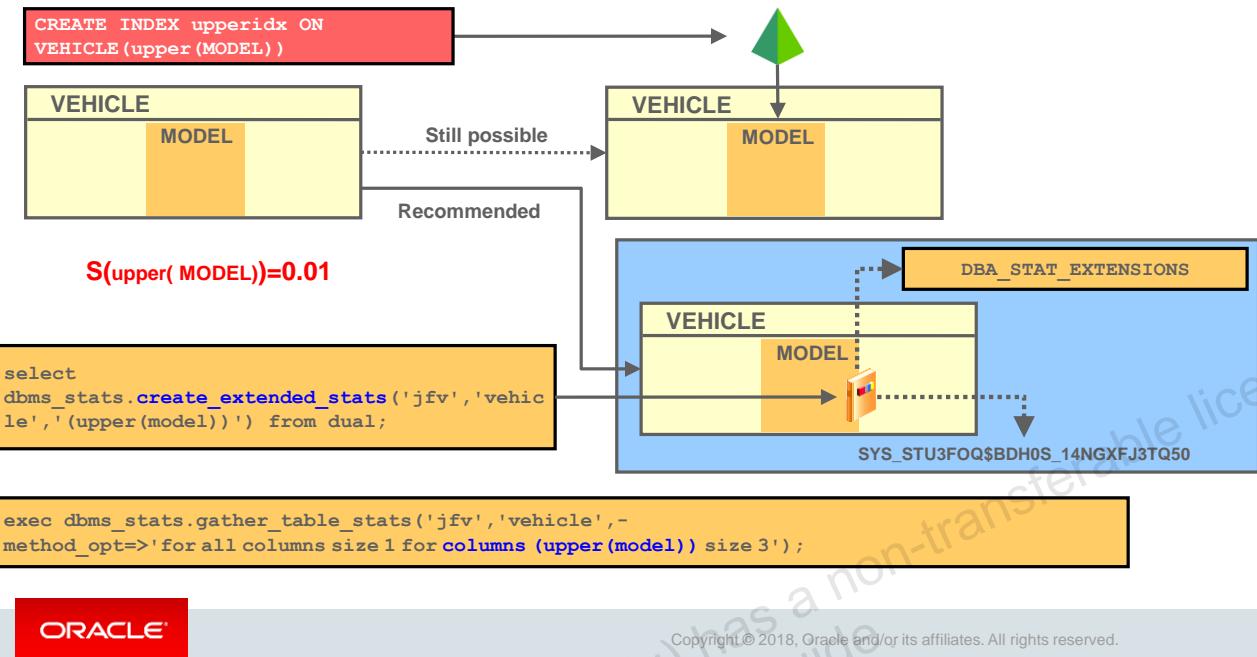
- The CREATE_EXTENDED_STATS function returns a virtual hidden column name, such as SYS_STUW_5RHLX443AN1ZCLPE_GLE4.
- Based on the example in the slide, the name can be determined by using the following SQL statement:

```
select
dbms_stats.show_extended_stats_name('jfv', 'vehicle', '(make,model)')
from dual
```
- After you create the statistics extensions, you can retrieve them by using the ALL|DBA|USER_STAT_EXTENSIONS views.

You can use the DBMS_STATS.DROP_EXTENDED_STATS function to delete a column group from a table.

You learn about detecting useful column groups for a specific workload, creating column groups manually, and gathering column group statistics in the practices for this lesson.

Expression Statistics



ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Predicates involving expressions on columns are a significant issue for the query optimizer. When computing selectivity on predicates of the form $function(Column) = constant$, the optimizer assumes a static selectivity value of 1 percent. This approach almost never has the correct selectivity, and it may cause the optimizer to produce suboptimal plans.

The query optimizer has been extended to better handle such predicates in limited cases where functions preserve the data distribution characteristics of a column and thus allow the optimizer to use column statistics. An example of such a function is `TO_NUMBER`.

Further enhancements have been made to evaluate built-in functions during query optimization to derive better selectivity by using dynamic statistics. Finally, the optimizer collects statistics on the virtual columns that are created to support function-based indexes.

However, these solutions are either limited to a certain class of functions or they work only for the expressions that are used to create function-based indexes. With expression statistics in Oracle Database, you can use a more general solution that includes arbitrary user-defined functions and does not depend on the presence of function-based indexes.

Expression statistics improve optimizer estimates when a `WHERE` clause has predicates that use expressions. As shown in the example in the slide, this feature relies on the virtual column infrastructure to create statistics on expressions of columns.

You can use DBMS_STATS to create statistics for a user-specified expression. You have the option of using either of the following program units:

- GATHER_TABLE_STATS procedure
- CREATE_EXTENDED_STATISTICS function followed by the GATHER_TABLE_STATS procedure

You can use the database view DBA_STAT_EXTENSIONS and the DBMS_STATS.SHOW_EXTENDED_STATS_NAME function to obtain information about expression statistics. You can also use views to obtain information such as the number of distinct values and whether the column group has a histogram.

Use the DBMS_STATS.DROP_EXTENDED_STATS function to delete a column group from a table.

Notes

- Extended statistics can be used only when the WHERE clause predicates are equalities or in-lists.
- Extended statistics will not be used if histograms are present on the underlying columns and no histogram is present on the column group.

Lesson Agenda

- Optimizer Statistics
 - Table Statistics
 - Index Statistics
 - Index Clustering Factor
 - Column Statistics
- Column Statistics: Histograms
- Column Statistics: Extended Statistics
- **Session-Specific Statistics for Global Temporary Tables**
- System Statistics
- Gathering and Managing Optimizer Statistics

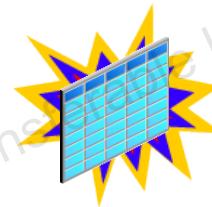


ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Session-Specific Statistics for Global Temporary Tables

- Traditionally, statistics gathered on global temporary tables (GTT) were shared by all sessions, even though data in different sessions could differ.
- Now, each session can have its own version of statistics for GTT.
- This is controlled by the new preference `GLOBAL_TEMP_TABLE_STATS`.
- The default value is `SESSION` (non-shared).
- To force sharing, set table preference to `SHARED`.



ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

A global temporary table is a special table that stores intermediate session-private data for a specific duration. Before Oracle Database 12c, it was difficult to gather statistics on global temporary tables because only one set of statistics was available for all occurrences of global temporary tables.

When you create a global temporary table, you create a definition that is visible to all sessions. No physical storage is allocated. When a session first puts data into the table, the database allocates storage space. The data in a temporary table is visible only to the current session.

In previous releases, the database did not maintain statistics for global temporary tables and non-global temporary tables differently. The database maintained one version of the statistics that was shared by all sessions, even though data in different sessions could differ.

Starting in Oracle Database 12c, you can maintain session-private statistics. You can choose to gather statistics for a global temporary table in one session and use the statistics only for this session. You can set the table-level preference `GLOBAL_TEMP_TABLE_STATS` to make statistics on a global temporary table either shared or session-specific.

Meanwhile, you can continue to maintain a shared version of the statistics that is usable by all other sessions, which have not yet gathered session-private statistics. Session-level DML monitoring is also provided to the global temporary tables. Session-specific statistics for global temporary tables allow queries to be optimized more accurately based on your own data in a global temporary table.

Session-specific statistics have the following characteristics:

- Dictionary views that track statistics show both the shared statistics and the session-specific statistics in the current session.
- The views are DBA_TAB_STATISTICS, DBA_IND_STATISTICS, DBA_TAB_HISTOGRAMS, and DBA_TAB_COL_STATISTICS (each view has a corresponding USER_ and ALL_ version). The SCOPE column shows whether statistics are session-specific or shared.
- Other sessions do not share the cursor that is using the session-specific statistics.
- Different sessions can share the cursor that is using shared statistics, as in previous releases. The same session can share the cursor that is using session-specific statistics.
- Pending statistics are not supported for session-specific statistics.
- When the GLOBAL_TEMP_TABLE_STATS preference is set to SESSION, by default, GATHER_TABLE_STATS immediately invalidates the previous cursors that were compiled in the same session. However, this procedure does not invalidate cursors compiled in other sessions.

DBMS_STATS commits changes to session-specific global temporary tables, but not to transaction-specific global temporary tables.

In previous releases, running DBMS_STATS.GATHER_TABLE_STATS on a transaction-specific temporary table (ON COMMIT DELETE ROWS) would delete all the rows in the table, making the statistics show the table as empty. Starting in Oracle Database 12c Release 1 (12.1), the following procedures do not commit for transaction-specific temporary tables, so that data in these tables is not lost:

- GATHER_TABLE_STATS
- DELETE_TABLE_STATS
- DELETE_COLUMN_STATS
- DELETE_INDEX_STATS
- SET_TABLE_STATS
- SET_COLUMN_STATS
- SET_INDEX_STATS
- GET_TABLE_STATS
- GET_COLUMN_STATS
- GET_INDEX_STATS

The preceding program units observe the GLOBAL_TEMP_TABLE_STATS preference.

Session-Specific Statistics for Global Temporary Tables: Example

- Statistics gathered on a GTT are no longer shared by all sessions.
- To restore shared statistics, change the table preference GLOBAL_TEMP_TABLE_STATS to SHARED.

The screenshot shows two sessions of SQL*Plus. Session 1 (top) creates a global temporary table 'TG' and checks its statistics preferences, which are listed as 'SESSION'. Session 2 (bottom) runs a PL/SQL block to set the preference to 'SHARED', then checks the statistics again, which are now listed as 'SHARED'. A callout box points from the 'SESSION' entry in Session 1 to a note about session-private statistics.

```
SQL> CREATE GLOBAL TEMPORARY TABLE TG(col1 NUMBER);
Table created.

SQL> SELECT DBMS_STATS.GET_PREFS('GLOBAL_TEMP_TABLE_STATS','SH','TG')from dual;
DBMS_STATS.GET_PREFS('GLOBAL_TEMP_TABLE_STATS','SH','TG')
SESSION
```

1

By default, statistics on GTT are session-private.

```
SQL> BEGIN
dbms_stats.set_table_prefs('SH','TG','GLOBAL_TEMP_TABLE_STATS','SHARED');
END;
/ 2 3 4

PL/SQL procedure successfully completed.

SQL> SELECT DBMS_STATS.GET_PREFS('GLOBAL_TEMP_TABLE_STATS','SH','TG')from dual;
DBMS_STATS.GET_PREFS('GLOBAL_TEMP_TABLE_STATS','SH','TG')
SHARED
```

2

ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

- Optimizer Statistics
 - Table Statistics
 - Index Statistics
 - Index Clustering Factor
 - Column Statistics
- Column Statistics: Histograms
- Column Statistics: Extended Statistics
- Session-Specific Statistics for Global Temporary Tables
- **System Statistics**
- Gathering and Managing Optimizer Statistics



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

System Statistics

- System statistics are used to estimate:
 - I/O performance and utilization
 - CPU performance and utilization
- System statistics enable the query optimizer to estimate I/O and CPU costs more accurately, enabling the query optimizer to choose a better execution plan.
- Procedures:
 - DBMS_STATS.GATHER_SYSTEM_STATS
 - DBMS_STATS.SET_SYSTEM_STATS
 - DBMS_STATS.GET_SYSTEM_STATS



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

System statistics allow the optimizer to consider a system's I/O and CPU performance and utilization. For each candidate plan, the optimizer computes estimates for I/O and CPU costs.

Oracle recommends that you gather system statistics when a physical change occurs in the environment, for example, the server has faster CPUs, more memory, or different disk storage.

System statistics are gathered in a user-defined time frame with the DBMS_STATS.GATHER_SYSTEM_STATS routine. You can also set system statistics values explicitly by using DBMS_STATS.SET_SYSTEM_STATS. Use DBMS_STATS.GET_SYSTEM_STATS to verify system statistics.

System Statistics: Example

Viewing System Statistics:

```
SELECT * FROM sys.aux_stats$;
```

SNAME	PNAME	PVAL1	PVAL2
1 SYSSTATS_INFO STATUS		(null) COMPLETED	
2 SYSSTATS_INFO DSTART		(null) 01-26-2017 15:27	
3 SYSSTATS_INFO DSTOP		(null) 01-26-2017 15:27	
4 SYSSTATS_INFO FLAGS		1 (null)	
5 SYSSTATS_MAIN CPUSPEEDNW	2923.27044025157	(null)	
6 SYSSTATS_MAIN IOSEEKTIM		10 (null)	
7 SYSSTATS_MAIN IOTFRSPEED		4096 (null)	
8 SYSSTATS_MAIN SREADTIM		(null) (null)	
9 SYSSTATS_MAIN MREADTIM		(null) (null)	
10 SYSSTATS_MAIN CPUSPEED		(null) (null)	
11 SYSSTATS_MAIN MBRC		(null) (null)	
12 SYSSTATS_MAIN MAXTHR		(null) (null)	
13 SYSSTATS_MAIN SLAVETHR		(null) (null)	



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

The slide shows an example of the optimizer system statistics.

- **sreadtim**: Single block read time is the average time to read a single block randomly.
- **mreadtim**: Multiblock read is the average time to read a multiblock sequentially.
- **cpuspeed**: This represents the workload CPU speed. CPU speed is the average number of CPU cycles in each second.
- **mbrc**: Multiblock count is the average multiblock read count sequentially.

For more information about these statistics, refer to *SQL Tuning Guide* (Table 13-4 Optimizer System Statistics in the DBMS_STAT Package)

Lesson Agenda

- Optimizer Statistics
 - Table Statistics
 - Index Statistics
 - Index Clustering Factor
 - Column Statistics
- Column Statistics: Histograms
- Column Statistics: Extended Statistics
- Session-Specific Statistics for Global Temporary Tables
- System Statistics
- Gathering and Managing Optimizer Statistics



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Gathering Statistics: Overview

- Optimizer statistics collection is the gathering of optimizer statistics for database objects.
- You can gather statistics by using the following:
 - Automatic optimizer statistics gathering
 - Manual statistics gathering
 - DBMS_STATS package
 - Adaptive statistics
 - Dynamic statistics
 - Automatic re-optimization
 - SQL plan directives
 - Online statistics gathering for bulk loads

Selectivity:	
Equality	1%
Inequality	5%
Other predicates	5%
Table row length	20
# of index leaf blocks	25
# of distinct values	100
Table cardinality	100
Remote table cardinality	2000



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In Oracle Database, optimizer statistics collection is the gathering of optimizer statistics for database objects. The database can collect these statistics automatically, or you can collect them manually by using the system-supplied DBMS_STATS package. These are discussed in more detail in the subsequent slides. It is recommended that you use automatic statistics gathering for objects.

The contents of tables and associated indexes change frequently, which can lead the optimizer to choose suboptimal execution plans for queries. Thus, statistics must be kept current to avoid any potential performance issues because of suboptimal plans.

To minimize DBA involvement, Oracle Database automatically gathers optimizer statistics at various times. Some automatic options are configurable, such as enabling AutoTask to run DBMS_STATS. You can manage optimizer statistics either through Oracle Enterprise Manager Cloud Control (Cloud Control) or using PL/SQL on the command line.

Notes

- When the system encounters a table with missing statistics, it dynamically gathers the necessary statistics needed by the optimizer. However, for certain types of tables (including remote tables and external tables), it does not perform dynamic sampling. In those cases, and also when dynamic sampling has been disabled, the optimizer uses default values for its statistics.
- For more information about Managing Statistics, refer to Appendix F titled “Gathering and Managing Optimizer Statistics.”

Manual Statistics Gathering

You can use Enterprise Manager and the `DBMS_STATS` package to:

- Generate and manage statistics for use by the optimizer:
 - Gather or modify
 - View or name
 - Export or import
 - Delete or lock
- Gather statistics on:
 - Indexes, tables, columns, and partitions
 - Object, schema, or database
- Gather statistics either serially or in parallel
- Gather or set system statistics (currently not possible in EM)



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Both Enterprise Manager and the `DBMS_STATS` package enable you to manually generate and manage statistics for the optimizer. You can use the `DBMS_STATS` package to gather, modify, view, export, import, lock, and delete statistics. You can also use this package to identify or name gathered statistics. You can gather statistics on indexes, tables, columns, and partitions at various granularity: object, schema, and database level.

`DBMS_STATS` gathers only those statistics needed for optimization; it does not gather other statistics. For example, the table statistics that are gathered by `DBMS_STATS` include the number of rows, number of blocks currently containing data, and average row length, but not the number of chained rows, average free space, or number of unused data blocks.

Note: Do not use the `COMPUTE` and `ESTIMATE` clauses of the `ANALYZE` statement to collect optimizer statistics. These clauses are supported solely for backward compatibility and may be removed in a future release. The `DBMS_STATS` package collects a broader, more accurate set of statistics, and gathers statistics more efficiently. You may continue to use the `ANALYZE` statement for other purposes that are not related to optimizer statistics collection, such as the following:

- To use the `VALIDATE` or `LIST CHAINED ROWS` clauses
- To collect information about free list blocks

When to Gather Statistics Manually

- Rely mostly on automatic statistics collection:
 - Change the frequency of automatic statistics collection to meet your needs.
 - Remember that `STATISTICS_LEVEL` should be set to `TYPICAL` or `ALL` for automatic statistics collection to work properly.
- Gather statistics manually for:
 - Objects that are volatile
 - Objects modified in batch operations (gather statistics as part of the batch operation)
 - External tables, system statistics, and fixed objects
 - New objects (gather statistics right after object creation)



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The automatic statistics gathering mechanism gathers statistics on schema objects in the database for which statistics are absent or stale. It is important to determine when and how often to gather new statistics. The default gathering interval is nightly, but you can change this interval to suit your business needs. You can do so by changing the characteristics of your maintenance windows. Some cases may require manual statistics gathering. For example, the statistics on tables that are significantly modified during the day may become stale. There are typically two types of such objects:

- Volatile tables that are modified significantly during the course of the day
- Objects that are the target of large bulk loads that add 10 percent or more to the object's total size between statistics-gathering intervals

For external tables, statistics are collected manually only by using `GATHER_TABLE_STATS`. Because sampling on external tables is not supported, the `ESTIMATE_PERCENT` option should be explicitly set to `null`. Because data manipulation is not allowed against external tables, it is sufficient to analyze external tables when the corresponding file changes. Other areas in which statistics need to be manually gathered are system statistics and fixed objects, such as the dynamic performance tables. These statistics are not automatically gathered.

Managing Statistics: Overview (Export / Import / Lock / Restore / Publish)

- Purpose:
 - To revert to pre-analyzed statistics if gathering statistics causes critical statements to perform badly
 - To test the new statistics before publishing
- Importing previously exported statistics (9*i*)
- Locking and unlocking statistics on a specific table (10*g*)
- Restoring statistics archived before gathering (10*g*)
- Keeping statistics pending before publishing (11*g R2*)



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The following are a few tasks to manage optimizer statistics:

- Importing previously exported statistics (9*i*)
- Locking and unlocking statistics on a specific table (10*g*)
- Restoring statistics archived before gathering (10*g*)
- Keeping statistics pending before publishing (11*g R2*)

Note: For more information about managing statistics, refer to Appendix F titled “Gathering and Managing Optimizer Statistics.”

Quiz



The optimizer depends on accurate statistics to produce the best execution plans. The automatic statistics-gathering task does not gather statistics on everything. Which objects require you to gather statistics manually?

- a. External tables
- b. Data dictionary
- c. Fixed objects
- d. Volatile tables
- e. System statistics



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Answer: a, c, d, e

Summary

In this lesson, you should have learned how to:

- Describe optimizer statistics
 - Table statistics
 - Index statistics
 - Column statistics (histogram)
 - Column statistics (extended statistics)
 - Session-specific statistics for global temporary tables
 - System statistics
- Gather and manage optimizer statistics



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Practice 11: Overview

This practice covers the following topics:

- Using the Index Clustering Factor
- Creating Expression Statistics
- Enabling Automatic Statistics Gathering (Optional)
- Using System Statistics (Optional)



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Using Bind Variables

The ORACLE logo, consisting of the word "ORACLE" in white capital letters on a red rectangular background.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- List the benefits of using bind variables
- Use bind peeking
- Use adaptive cursor sharing
- Describe common observations



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

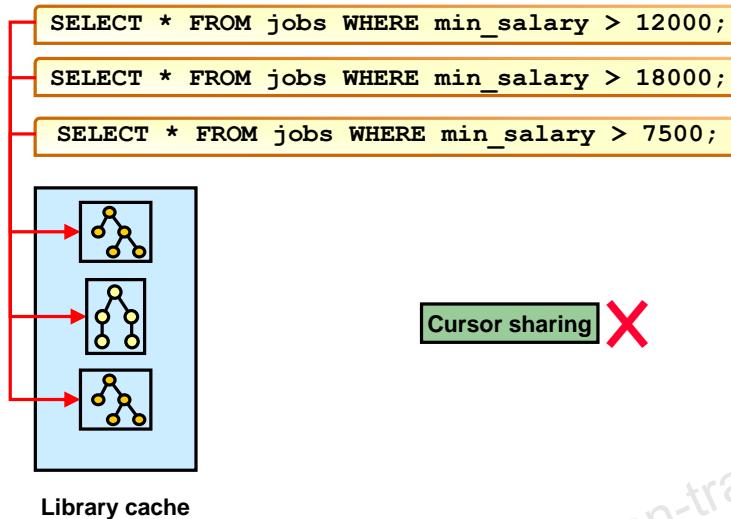
- Cursor Sharing and Different Literal Values
- Cursor Sharing and Bind Variables
 - Bind Variables in SQL*Plus
 - Bind Variables in Enterprise Manager
 - Bind Variables in SQL Developer
- Bind Variable Peeking
- Cursor Sharing
- Adaptive Cursor Sharing: Overview
 - Adaptive Cursor Sharing: Architecture
 - Adaptive Cursor Sharing: Views
 - Adaptive Cursor Sharing: Example



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Cursor Sharing and Different Literal Values



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

If your SQL statements use literal values for the WHERE clause conditions, there will be many versions of almost identical SQL stored in the library cache. For each different SQL statement, the optimizer must perform all the steps for processing a new SQL statement. This may also cause the library cache to fill up quickly because of all the different statements stored in it.

When coded this way, you are not taking advantage of cursor sharing. If the cursor is shared by using a bind variable rather than a literal variable, there will be one shared cursor, with one execution plan.

However, depending on the literal value provided, different execution plans may be generated by the optimizer. For example, there might be several JOBS, where MIN_SALARY is greater than 12000. Alternatively, there might be very few JOBS that have MIN_SALARY greater than 18000. This difference in data distribution could justify the addition of an index so that different plans can be used depending on the value provided in the query. This is illustrated in the slide. As you can see, the first and third queries use the same execution plan, but the second query uses a different one.

From a performance perspective, it is good to have separate cursors. However, this is not very economical because you could have shared cursors for the first and last queries in this example.

Note: In the case of the example in the slide, V\$SQL.PLAN_HASH_VALUE is identical for the first and third query.

Lesson Agenda

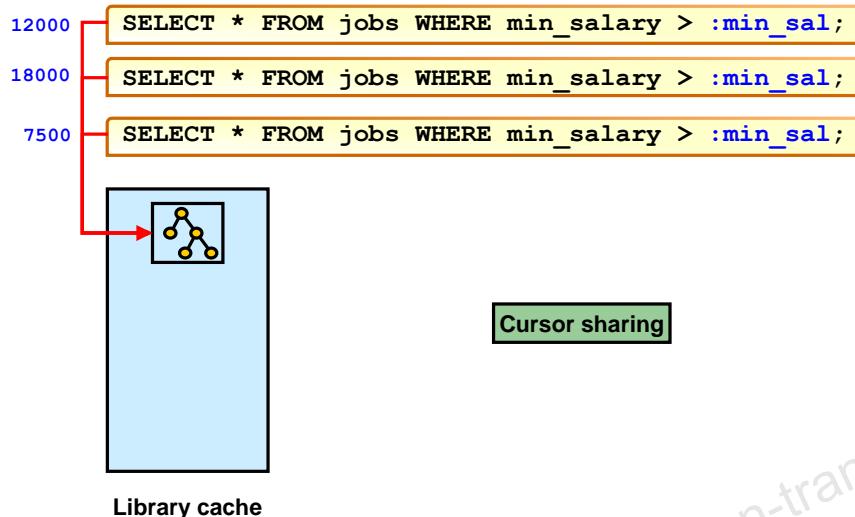
- Cursor Sharing and Different Literal Values
- Cursor Sharing and Bind Variables
 - Bind Variables in SQL*Plus
 - Bind Variables in Enterprise Manager
 - Bind Variables in SQL Developer
- Bind Variable Peeking
- Cursor Sharing
- Adaptive Cursor Sharing: Overview
 - Adaptive Cursor Sharing: Architecture
 - Adaptive Cursor Sharing: Views
 - Adaptive Cursor Sharing: Example



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Cursor Sharing and Bind Variables



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Instead of issuing different statements for each literal, if you use a bind variable, the extra parse activity is eliminated (in theory). The elimination occurs because the optimizer recognizes that the statement is already parsed and decides to reuse the same execution plan even though you specify different bind values the next time you execute the same statement.

For the example in the slide, the bind variable is called `min_sal`. It is to be compared with the `MIN_SALARY` column of the `JOB`s table. Instead of issuing three different statements, issue a single statement that uses a bind variable. At execution time, the same execution plan is used, and the given value is substituted for the variable.

However, from a performance perspective, this is not the best situation because you get best performance two times out of three. On the other hand, this is very economical because you need only one shared cursor in the library cache to execute all three statements.

Bind Variables in SQL*Plus

```
SQL> variable job_id varchar2(10)
SQL> exec :job_id := 'SA_REP';

PL/SQL procedure successfully completed.

SQL> select count(*) from employees where job_id = :job_id;

COUNT(*)
-----
      30

SQL> exec :job_id := 'AD_VP';

PL/SQL procedure successfully completed.

SQL> select count(*) from employees where job_id = :job_id;

COUNT(*)
-----
      2
```



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Bind variables can be used in Oracle SQL*Plus sessions.

In Oracle SQL*Plus, use the VARIABLE command to define a bind variable. Then, you can assign values to the variable by executing an assignment statement with the EXEC [UTE] command. Any references to that variable from then on use the assigned value.

For the example in the slide, the first count is selected while SA_REP is assigned to the variable. The result is 30.

Then, AD_VP is assigned to the variable, and the resulting count is 2.

Bind Variables in Enterprise Manager

The screenshot shows the Oracle Enterprise Manager SQL Worksheet interface. In the SQL Commands section, a query is entered:

```
select count(*) from hr.employees where salary between :low_sal and :hi_sal
```

Two red boxes highlight the bind variables `:low_sal` and `:hi_sal`. A red arrow points from the first box to the "Select Value" table, which contains two entries:

Select Value	Data Type
5000	NUMBER
10000	NUMBER
	STRING
	STRING
	STRING

The "Use bind variables for execution" checkbox is checked. Below the table are buttons for "Remove", "Move Up", "Move Down", "Add 5 Rows", and "Remove All". At the bottom of the table are checkboxes for "Auto commit" and "Allow only SELECT statements", and an "Execute" button.

In the Last Executed SQL section, the query is displayed again:

```
select count(*)  
from hr.employees  
where salary between :low_sal and :hi_sal
```

The Last Execution Details section shows the results:

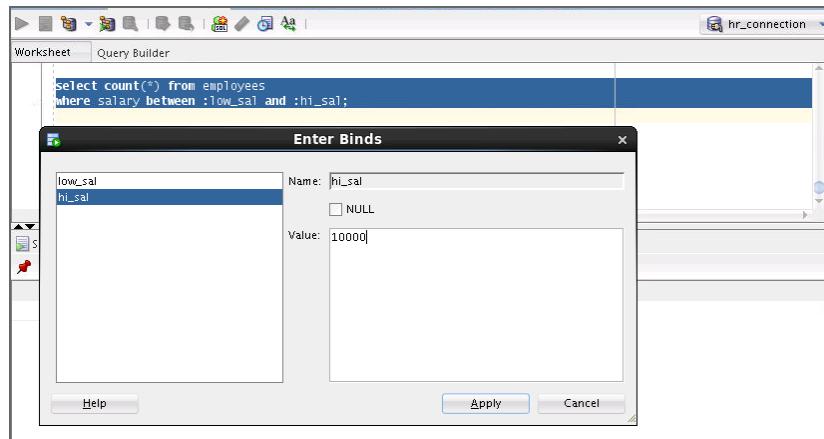
Results	Statistics	Plan
		Execution Time (seconds) 0.0010
COUNT(*)		43

Buttons for "SQL Repair Advisor", "SQL Details", and "Schedule SQL Tuning Advisor" are also present. The ORACLE logo is at the bottom left, and a copyright notice is at the bottom right.

You can access the SQL Worksheet on the EMCC by going to the **orclpdb** home page under Databases and clicking **Performances > SQL > SQL Worksheet**.

- On the SQL Worksheet page, you can specify that a SQL statement should use bind variables.
- You can do this by selecting the “Use bind variables for execution” check box.
- When you select this check box, you can enter bind variable values in the generated fields. Refer to these values in the SQL statement by using variable names that begin with a colon.
- The order in which variables are referred to defines which variable gets which value. The first variable referred to gets the first value, the second variable gets the second value, and so on.
- If you change the order in which variables are referenced in the statement, you may need to change the value list to match that order.

Bind Variables in Oracle SQL Developer



ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

On the SQL Worksheet pane of SQL Developer, you can specify that a SQL statement uses bind variables.

When you execute the statement, you can enter bind variable values in the *Enter Binds* dialog box. Refer to these values in the SQL statement by using variable names that begin with a colon.

Select each bind variable in turn to enter a value for that variable.

Lesson Agenda

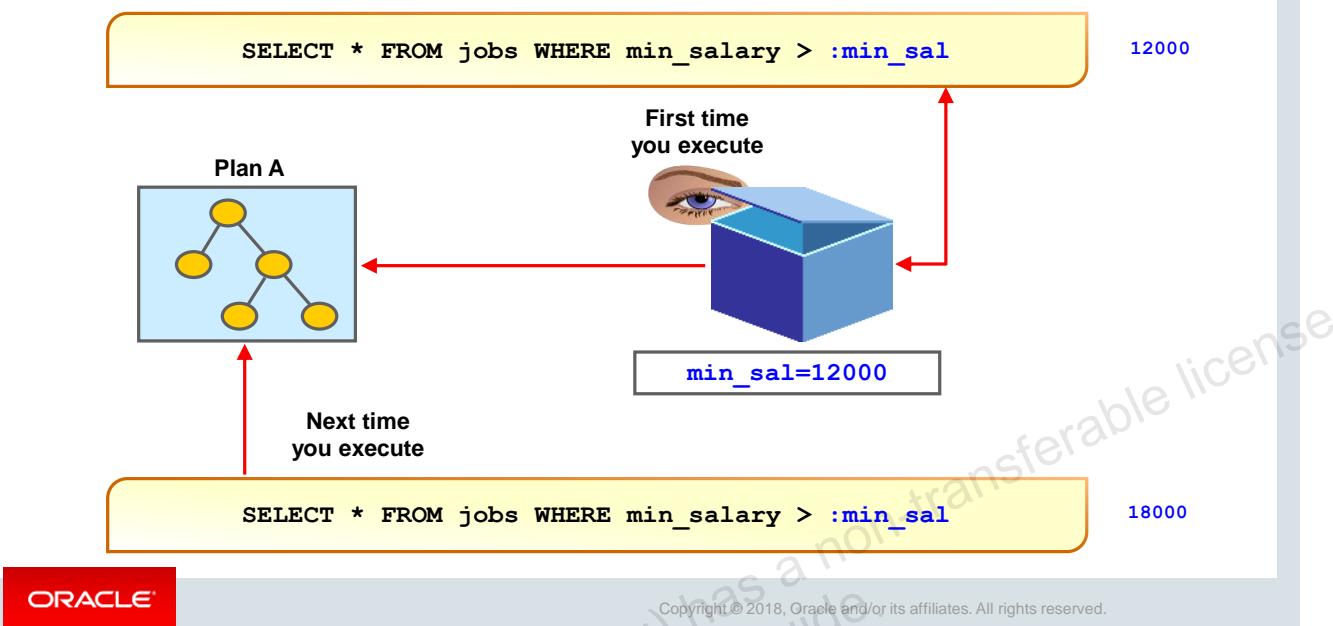
- Cursor Sharing and Different Literal Values
- Cursor Sharing and Bind Variables
 - Bind Variables in SQL*Plus
 - Bind Variables in Enterprise Manager
 - Bind Variables in SQL Developer
- Bind Variable Peeking
- Cursor Sharing
- Adaptive Cursor Sharing: Overview
 - Adaptive Cursor Sharing: Architecture
 - Adaptive Cursor Sharing: Views
 - Adaptive Cursor Sharing: Example



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Bind Variable Peeking



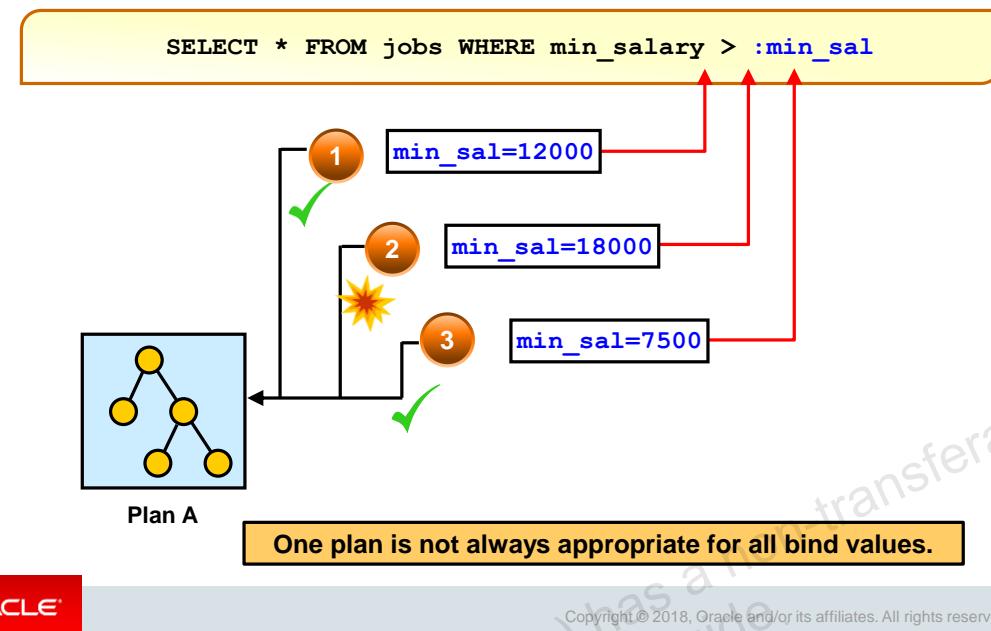
When literals are used in a query, those literal values can be used by the optimizer to decide on the best plan.

However, when bind variables are used, the optimizer still needs to select the best plan based on the values of the conditions in the query, but cannot readily see those values in the SQL text. This means that as a SQL statement is parsed, the system needs to be able to see the value of the bind variables to ensure that a good plan that would suit those values is selected.

The optimizer does this by *peeking* at the value in the bind variable. By peeking at bind values, the optimizer can determine the selectivity of a `WHERE` clause condition as if literals *had* been used, thereby improving the plan.

When the SQL statement is hard parsed, the optimizer evaluates the value for each bind variable, and uses that as input in determining the best plan. After the execution is determined the first time you parsed the query, it is reused when you execute the same statement regardless of the bind values used.

Bind Variable Peeking



Under some conditions, bind variable peeking can cause the optimizer to select a suboptimal plan. This occurs because the first value of the bind variable is used to determine the plan for all subsequent executions of the query. Therefore, even though subsequent executions provide different bind values, the same plan is used.

It is possible that a different plan would be better for executions that have different bind variable values. An example is where the selectivity of a particular index varies extremely depending on the column value. For low selectivity, a full table scan may be faster. For high selectivity, an index range scan may be more appropriate.

As shown in the slide, plan A may be good for the first and third values of `min_sal`, but it may not be the best for the second one. Suppose there are very few `MIN_SALARY` values that are above 18000 and plan A is a full table scan. It is probable that a full table scan is not a good plan for the second execution, in that case.

Bind variables are beneficial in that they cause more cursor sharing to happen, and thus reduce parsing of SQL. But, as in this case, it is possible that they cause a suboptimal plan to be chosen for some of the bind variable values. This is a good reason for not using bind variables for decision support system (DSS) environments, where the parsing of a query is a very small percentage of the work done when submitting the query. The parsing may take fractions of a second, but the execution may take minutes or hours. To execute with a slower plan is not worth the savings gained in parse time.

Lesson Agenda

- Cursor Sharing and Different Literal Values
- Cursor Sharing and Bind Variables
 - Bind Variables in SQL*Plus
 - Bind Variables in Enterprise Manager
 - Bind Variables in SQL Developer
- Bind Variable Peeking
- **Cursor Sharing**
- Adaptive Cursor Sharing: Overview
 - Adaptive Cursor Sharing: Architecture
 - Adaptive Cursor Sharing: Views
 - Adaptive Cursor Sharing: Example



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Cursor Sharing Enhancements

- Oracle 8*i* introduced the possibility of sharing SQL statements that differ only in literal values.
- Oracle 9*i* extended this feature by limiting it to similar statements, instead of forcing it.
- Similar: Regardless of the literal value, same execution plan

```
SQL> SELECT * FROM employees  
2 WHERE employee_id = 153;
```

- Not similar: Possible different execution plans for different literal values

```
SQL> SELECT * FROM employees  
2 WHERE department_id = 50;
```



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Oracle 8*i* introduced the possibility of sharing SQL statements that differ only in literal values. Rather than developing an execution plan every time the same statement—with a different literal value—is executed, the optimizer generates a common execution plan that is used for all subsequent executions of the statement.

Because only one execution plan is used instead of potential different ones, this feature should be tested against your applications before you decide whether to enable it or not. That is why Oracle 9*i* extends this feature by sharing only statements that are considered as similar, that is, only when the optimizer has the guarantee that the execution plan is independent of the literal value used. For example, consider a query where `EMPLOYEE_ID` is the primary key:

```
SQL> SELECT * FROM employees WHERE employee_id = 153;
```

The substitution of any value would produce the same execution plan. It would, therefore, be safe for the optimizer to generate only one plan for different occurrences of the same statement that is executed with different literal values.

On the other hand, assume that the same `EMPLOYEES` table has a wide range of values in its `DEPARTMENT_ID` column. For example, department 50 could contain more than one-third of all employees, and department 70 could contain only one or two.

Observe the two queries:

```
SQL> SELECT * FROM employees WHERE department_id = 50;  
SQL> SELECT * FROM employees WHERE department_id = 70;
```

Using only one execution plan for sharing the same cursor would not be safe if you have histogram statistics (and there is skew in the data) on the DEPARTMENT_ID column.

In this case, depending on which statement was executed first, the execution plan could contain a full table (or fast full index) scan, or it could use a simple index range scan.

CURSOR_SHARING Parameter

- CURSOR_SHARING parameter values:
 - FORCE
 - EXACT (default)
- CURSOR_SHARING can be changed by using:
 - ALTER SYSTEM
 - ALTER SESSION
 - Initialization parameter files
- CURSOR_SHARING_EXACT hint



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The value of the CURSOR_SHARING initialization parameter determines how the optimizer processes statements with bind variables:

- EXACT: Allows only statements with identical text to share the same cursor
- FORCE: Allows the creation of a new cursor if sharing an existing cursor, or if the cursor plan is not optimal

The best practice is to write sharable SQL and use the default of EXACT for CURSOR_SHARING. By default, Oracle Database uses adaptive cursor sharing to enable a single SQL statement that contains bind variables to use multiple execution plans.

However, for applications with many similar statements, setting CURSOR_SHARING to FORCE can significantly improve cursor sharing, resulting in reduced memory usage, faster parses, and reduced latch contention.

The value of CURSOR_SHARING in the initialization file can be overridden with an ALTER SYSTEM SET CURSOR_SHARING or an ALTER SESSION SET CURSOR_SHARING command.

The CURSOR_SHARING_EXACT hint causes the system to execute the SQL statement without any attempt to replace literals with bind variables.

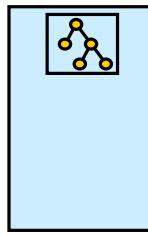
Note: Starting in Oracle Database 12c, the SIMILAR value for CURSOR_SHARING is deprecated. Use FORCE instead.

Forcing Cursor Sharing: Example

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

```
SELECT * FROM jobs WHERE min_salary > 12000;
SELECT * FROM jobs WHERE min_salary > 18000;
SELECT * FROM jobs WHERE min_salary > 7500;
```

```
SELECT * FROM jobs WHERE min_salary > :"SYS_B_0"
```



System-generated bind variable

ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

For the example in the slide, because you forced cursor sharing with the ALTER SESSION command, all your queries that differ only in literal values are automatically rewritten to use the same system-generated bind variable called SYS_B_0. As a result, you end up with only one child cursor instead of three.

When CURSOR_SHARING is set to FORCE, the database uses one parent cursor and one child cursor for each distinct SQL statement. The database uses the same plan for each execution of the same statement, for example:

```
SELECT * FROM hr.employees WHERE employee_id = 101;
```

If you use FORCE, the database optimizes this statement as if it contained a bind variable and uses bind peeking to estimate cardinality. Statements that differ only in the bind variable share the same execution plan.

Setting CURSOR_SHARING to FORCE has the following drawbacks:

- The database must perform extra work during soft parse to find a similar statement in the shared pool.
- There is an increase in the maximum lengths (as returned by DESCRIBE) of any selected expressions that contain literals in a SELECT statement. However, the actual length of the data returned does not change.
- Star transformation is not supported.

Note: Adaptive cursor sharing may also apply, and might generate a second child cursor in this case.

Lesson Agenda

- Cursor Sharing and Different Literal Values
- Cursor Sharing and Bind Variables
 - Bind Variables in SQL*Plus
 - Bind Variables in Enterprise Manager
 - Bind Variables in SQL Developer
- Bind Variable Peeking
- Cursor Sharing
- Adaptive Cursor Sharing: Overview
 - Adaptive Cursor Sharing: Architecture
 - Adaptive Cursor Sharing: Views
 - Adaptive Cursor Sharing: Example



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Adaptive Cursor Sharing: Overview

- Cursor sharing is adaptive because the cursor adapts its behavior so that the database does not always use the same plan for each execution or bind variable value.
- This allows intelligent cursor sharing for statements that use bind variables.
- It is used to compromise between cursor sharing and optimization.
- It has the following benefits:
 - Automatically detects when different executions would benefit from different execution plans
 - Limits the number of generated child cursors to a minimum
 - Provides an automated mechanism that cannot be turned off



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Bind variables were designed to allow Oracle Database to share a single cursor for multiple SQL statements to reduce the amount of shared memory used to parse SQL statements.

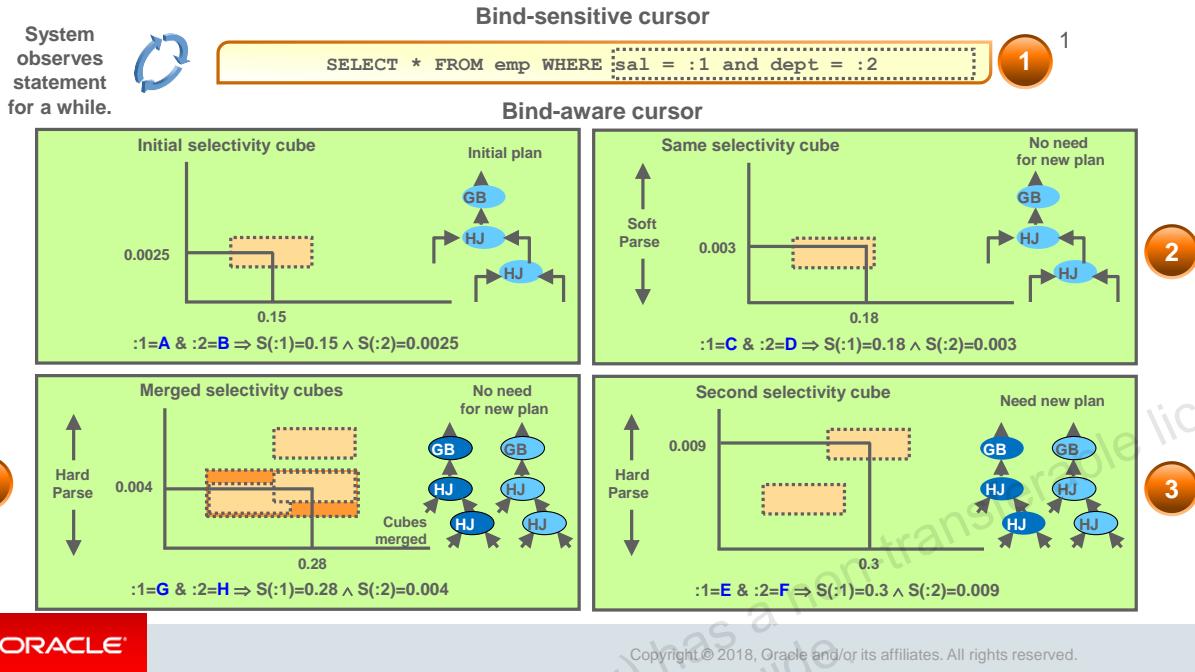
However, cursor sharing and SQL optimization are conflicting goals. Writing a SQL statement with literals provides more information for the optimizer and naturally leads to better execution plans, while increasing memory and CPU overhead caused by excessive hard parses.

Oracle Database 9*i* was the first attempt to introduce a compromise solution by allowing similar SQL statements that use different literal values to be shared. For statements using bind variables, Oracle Database 9*i* also introduced the concept of bind peeking. To benefit from bind peeking, it is assumed that cursor sharing is intended and that different invocations of the statement are supposed to use the same execution plan. If different invocations of the statement would significantly benefit from different execution plans, bind peeking is of no use in generating good execution plans.

To address this issue, Oracle Database 11*g* introduced adaptive cursor sharing. This feature is a more sophisticated strategy that is designed not to share a cursor blindly, but generate multiple plans per SQL statement with bind variables, if the benefit of using multiple execution plans outweighs the parse time and memory usage overhead. Adaptive cursor sharing is enabled for the database by default and cannot be disabled. Adaptive cursor sharing does not apply to SQL statements containing more than 14 bind variables.

Note: Adaptive cursor sharing is independent of the CURSOR_SHARING initialization parameter.

Adaptive Cursor Sharing: Architecture



When you use adaptive cursor sharing, the following steps take place in the scenario illustrated in the slide:

1. The cursor starts its life with a hard parse, as usual. If bind peeking takes place and a histogram is used to compute selectivity of the predicate containing the bind variable, the cursor is marked as a bind-sensitive cursor. In addition, some information is stored about the predicate containing the bind variables, including the predicate selectivity. In the slide example, the predicate selectivity that would be stored is a cube that is centered around (0.15, 0.0025). Because of the initial hard parse, an initial execution plan is determined by using the peeked binds. After the cursor is executed, the bind values and the execution statistics of the cursor are stored in that cursor.
During the next execution of the statement, when a new set of bind values is used, the system performs a usual soft parse, and finds a matching cursor for execution. At the end of execution, execution statistics are compared with the ones currently stored in the cursor. The system then observes the pattern of the statistics over all the previous runs (see the `V$SQL_CS_...` views in the next slide) and decides whether or not to mark the cursor as bind aware.
2. On the next soft parse of this query, if the cursor is now bind aware, bind-aware cursor matching is used. Suppose the selectivity of the predicate with the new set of bind values is now (0.18, 0.003). Because selectivity is used as part of bind-aware cursor matching, and because the selectivity is within an existing cube, the statement uses the existing child cursor's execution plan to run.

3. On the next soft parse of this query, suppose that the selectivity of the predicate with the new set of bind values is now (0.3,0.009). Because this selectivity is not within an existing cube, no child cursor match is found; the system does a hard parse, which generates a new child cursor with a second execution plan in that case. In addition, the new selectivity cube is stored as part of the new child cursor. After the new child cursor executes, the system stores the bind values and execution statistics in the cursor.
4. On the next soft parse of this query, suppose the selectivity of the predicate with the new set of bind values is now (0.28,0.004). Because this selectivity is not within one of the existing cubes, the system does a hard parse. Suppose that this time, the hard parse generates the same execution plan as the first one. Because the plan is the same as the first child cursor, both child cursors are merged. That is, both cubes are merged into a new bigger cube, and one of the child cursors is deleted. The next time there is a soft parse, if the selectivity falls within the new cube, the child cursor matches.

Adaptive Cursor Sharing: Views

The following views provide information about adaptive cursor sharing usage:

V\$SQL	Two new columns show whether a cursor is bind sensitive or bind aware.
V\$SQL_CS_HISTOGRAM	The distribution of the execution count is shown across the execution history histogram.
V\$SQL_CS_SELECTIVITY	The selectivity cubes stored for every predicate containing a bind variable and whose selectivity is used in the cursor sharing checks are shown.
V\$SQL_CS_STATISTICS	Execution statistics of a cursor using different bind sets are shown.



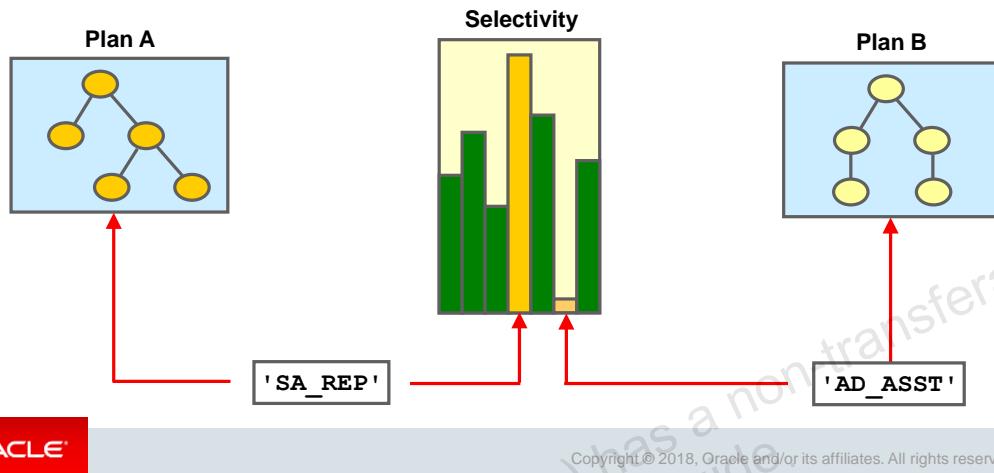
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

These views determine whether a query is bind aware or not and whether it is handled automatically, without any user input. However, information about what goes on is exposed through V\$ views so that you can diagnose problems, if any. New columns have been added to V\$SQL:

- **IS_BIND_SENSITIVE:** This indicates if a cursor is bind sensitive; value YES | NO. A query for which the optimizer peeked at bind variable values when computing predicate selectivity and where a change in a bind variable value may lead to a different plan is called *bind sensitive*.
- **IS_BIND_AWARE:** This indicates if a cursor is bind aware; value YES | NO. A cursor in the cursor cache that has been marked to use bind-aware cursor sharing is called *bind aware*.
- **V\$SQL_CS_HISTOGRAM:** This shows the distribution of the execution count across a three-bucket execution history histogram
- **V\$SQL_CS_SELECTIVITY:** This shows the selectivity cubes or ranges stored in a cursor for every predicate containing a bind variable and whose selectivity is used in the cursor sharing checks. It contains the text of the predicates and the low and high values of the selectivity range.
- **V\$SQL_CS_STATISTICS:** Adaptive cursor sharing monitors the execution of a query, collects information about it for a while, and uses this information to decide whether to switch to bind-aware cursor sharing for the query. This view summarizes the information that it collects to make this decision. For a sample of executions, it keeps track of the rows processed, buffer gets, and CPU time. The PEEKED column has the value YES if the bind set was used to build the cursor; it has the value NO otherwise.

Adaptive Cursor Sharing: Example

```
SQL> variable job varchar2(10)
SQL> exec :job := 'AD_ASST'
SQL> select count(*), max(salary) from employees where job_id=:job;
```



Consider the data in the slide. Histogram statistics on the `JOB_ID` column show that the occurrence of `SA_REP` is many thousands of times more than that of `AD_ASST`. In this case, if literals were used instead of a bind variable, the query optimizer would see that the `AD_ASST` value occurs in less than 1 percent of the rows, whereas the `SA_REP` value occurs in approximately one-third of the rows.

If the table has over a million rows, the execution plans are different for each of these values' queries. The `AD_ASST` query results in an index range scan because there are so few rows with that value. The `SA_REP` query results in a full table scan because so many of the rows have that value that it is more efficient to read the entire table.

But, as it is, using a bind variable causes the same execution plan to be used at first for both the values. So, even though there exist different and better plans for each of these values, they use the same plan.

After several executions of this query using a bind variable, the system considers the query bind aware, at which point it changes the plan based on the bound value. This means the best plan is used for the query, based on the bind variable value.

Interacting with Adaptive Cursor Sharing

- **CURSOR_SHARING:**
 - If CURSOR_SHARING <> EXACT, statements containing literals may be rewritten by using bind variables.
 - If statements are rewritten, adaptive cursor sharing may apply to them.
- SQL Plan Management (SPM):
 - If OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES is set to TRUE, only the first generated plan is used.
 - As a workaround, set this parameter to FALSE and run your application until all plans are loaded in the cursor cache.
 - Manually load the cursor cache into the corresponding plan baseline.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Adaptive cursor sharing is independent of the CURSOR_SHARING parameter. The setting of this parameter determines whether literals are replaced by system-generated bind variables. If they are, adaptive cursor sharing behaves just as it would if the user supplied binds to begin with.

When using the SPM automatic plan capture, the first plan captured for a SQL statement with bind variables is marked as the corresponding SQL plan baseline. If another plan is found for that same SQL statement (which may be the case with adaptive cursor sharing), it is added to the SQL statement's plan history and marked for verification. It will not be used immediately. Although adaptive cursor sharing has come up with a new plan based on a new set of bind values, SPM does not let it be used until the plan has been verified. Thus reverting to Oracle Database 10g behavior, only the plan generated based on the first set of bind values is used by all subsequent executions of the statement.

One possible workaround is to run the system for some time with automatic plan capture set to False. After the cursor cache has been populated with all the plans for a SQL statement with bind, load the entire plan directly from the cursor cache into the corresponding SQL plan baseline. By doing this, all the plans for a single SQL statement are marked as SQL baseline plans by default. Refer to the lesson titled “SQL Plan Management” for more information.

Common Observations

Consider the following areas to resolve excessive parsing time as well:

- CPU time dominates the parse time.
- Wait time dominates the parse time.

SELECT * FROM								
call	count	cpu	elapsed	disk	query	current	rows	
Parse	555	100.09	300.83	0	0	0	0	
Execute	555	0.42	0.78	0	0	0	0	
Fetch	555	14.04	85.03	513	1448514	0	11724	
total	1665	114.55	386.65	513	1448514	0	11724	



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

You might also need to consider the following areas to resolve excessive parsing problems:

- **CPU time dominates the parse time:** If high CPU usage (in general more than 50 percent of parse elapsed time) is observed during the hard parses, you would need to review the following common causes:
 - **Dynamic statistics:** Dynamic statistics is performed via hint, parameters, or missing statistics. Because dynamic statistics is performed at parse time, depending on the level of the dynamic statistics, it may impact the parse time.
 - **Many IN-LIST parameters or OR clauses:** It may take a long time to estimate the cost of a statement with many IN-LIST or OR clauses. You could use the NO_EXPAND hint to avoid query transformation, eventually to save the parse time.
- **Wait time dominates the parse time:** High wait time during hard parses are usually due to contention for resources or are related to very large queries. You would need to examine the waits ("SQL*Net more data from client") in the 10046 trace for the query. One of the common causes is seen when a large query containing lots of text is sent. Because it may take several round trips to be sent from the client to the server, each trip takes time, especially on slow networks.

Quiz



Which three statements are true about applications that are coded with literals rather than bind variables in the SQL statements?

- a. More shared pool space is required for cursors.
- b. Less shared pool space is required for cursors.
- c. Histograms are used if they are available.
- d. Histograms are not used.
- e. No parsing is required for literal values.
- f. Every different literal value requires parsing.



ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Answer: a, c, f

Quiz



The CURSOR_SHARING parameter should be set to _____ for systems with large tables and long-running queries, such as a data warehouse.

- a. Force
- b. Exact
- c. Literal
- d. True
- e. False



ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Answer: b

Quiz



Adaptive cursor sharing can be turned off by setting the CURSOR_SHARING parameter to FALSE.

- a. True
- b. False



ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Answer: b

Summary

In this lesson, you should have learned how to:

- List the benefits of using bind variables
- Use bind peeking
- Use adaptive cursor sharing
- Describe common observations



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Practice 12: Overview

This practice covers the following topics:

- Using Adaptive Cursor Sharing and Bind Peeking
- Using the `CURSOR_SHARING` Initialization Parameter (Optional)



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

SQL Plan Management

The ORACLE logo, consisting of the word "ORACLE" in white capital letters on a red rectangular background.

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Manage SQL performance through changes
- Set up SQL Plan Management
- Set up various SQL Plan Management scenarios
- Load hinted plans into SQL Plan Management



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

- Maintaining SQL Performance
- SQL Plan Management: Overview
- SQL Plan Baseline: Architecture
- Basic Tasks in SQL Plan Management
- Adaptive SQL Plan Management
- Possible SQL Plan Manageability Scenarios
- Enterprise Manager and SQL Plan Baselines
- Loading Hinted Plans into SPM: Example



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Maintaining SQL Performance

Maintaining performance may require using SQL plan baselines.



ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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

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

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

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

SQL outlines have been used in past versions. They are still available for backward compatibility, but outlines are deprecated in favor of SPM.

Lesson Agenda

- Maintaining SQL Performance
- SQL Plan Management: Overview
- SQL Plan Baseline: Architecture
- Basic Tasks in SQL Plan Management
- Adaptive SQL Plan Management
- Possible SQL Plan Manageability Scenarios
- Enterprise Manager and SQL Plan Baselines
- Loading Hinted Plans into SPM: Example



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

SQL Plan Management: Overview

- SQL plan management is a preventative mechanism that records and evaluates the execution plans of SQL statements over time.
- The optimizer automatically manages execution plans.
 - Only known and verified plans are used.
- Plan changes are automatically verified.
 - Only comparable or better plans are subsequently used.
- SQL plan management uses a mechanism called a SQL plan baseline.
- The primary goal of SQL plan management is to prevent performance regressions caused by plan changes.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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

SQL plan management is a preventative mechanism that enables the optimizer to automatically manage execution plans, ensuring that the database uses only known or verified plans. In this context, a plan includes all plan-related information (for example, SQL plan identifier, set of hints, bind values, and optimizer environment) that the optimizer needs to reproduce an execution plan.

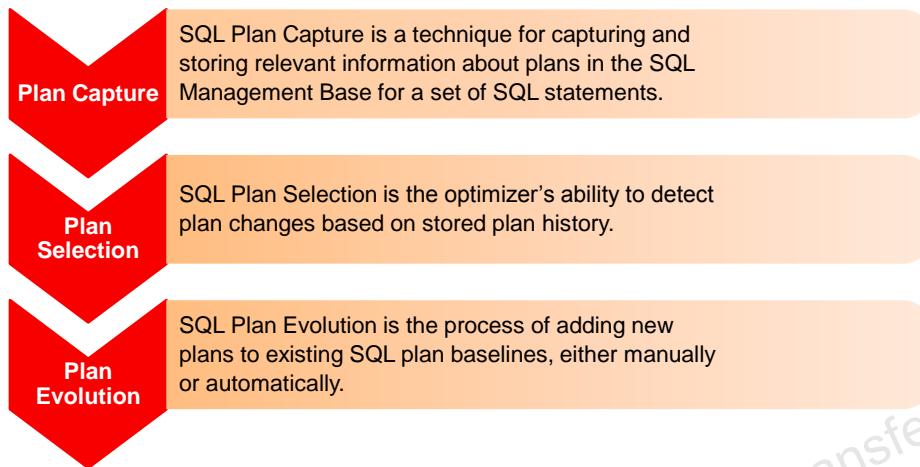
Various plan control techniques are available in Oracle Database to address performance regressions due to plan changes. The oldest is the use of hints in SQL code to force a specific access path. Stored outlines allowed the hints to be stored separately from the code and modified. Both techniques focused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

Note: A **SQL profile** is a database object that contains auxiliary statistics specific to a SQL statement. SQL profiles are created when a DBA invokes SQL Tuning Advisor.

During the execution of a SQL statement, only a plan that is part of the SQL plan baseline can be used. As described later in this lesson, SQL plan baselines can be automatically loaded or they can be seeded by using SQL tuning sets. Various scenarios are covered later in this lesson.

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

Components of SQL Plan Management



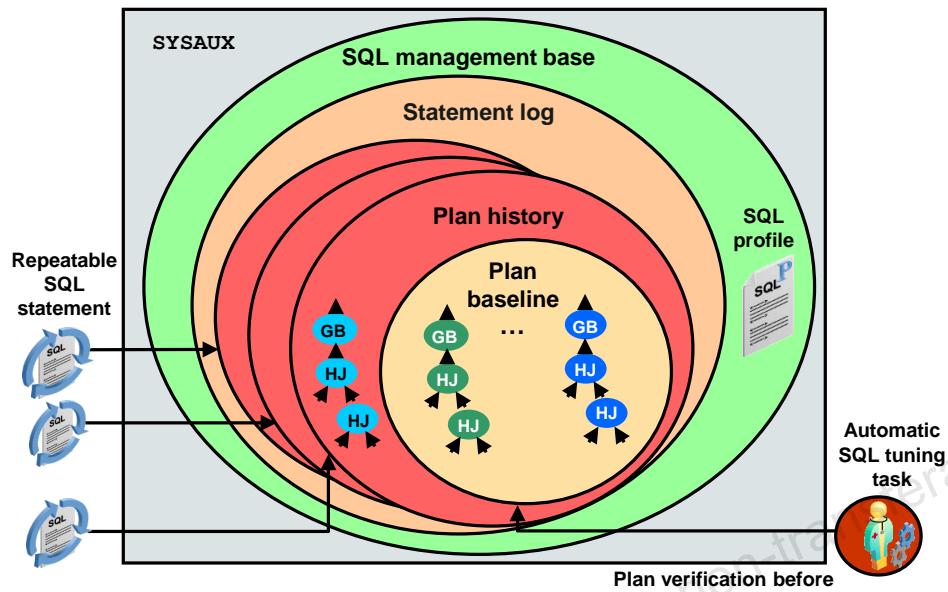
ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

SQL Plan Management has three main components:

- **SQL plan baseline capture:** SQL plan capture refers to techniques for capturing and storing relevant information about plans in the SQL Management Base for a set of SQL statements. Capturing a plan means making SQL plan management aware of this plan. You can configure initial plan capture to occur automatically by setting an initialization parameter, or you can capture plans manually by using the `DBMS_SPM` package. The SQL plan baselines are stored in a plan history in the SQL Management Base in the `SYSAUX` tablespace.
- **SQL plan baseline selection:** Ensure that only accepted execution plans are used for statements with a SQL plan baseline and track all new execution plans in the plan history for a statement. The plan history consists of accepted and unaccepted plans. An unaccepted plan can be unverified (newly found but not verified) or rejected (verified but not found to be performant).
- **SQL plan baseline evolution:** Evaluate all unverified execution plans for a given statement in the plan history to become either accepted or rejected. SQL plan evolution is the process of adding new plans to existing SQL plan baselines, either manually or automatically.

SQL Plan Baseline: Architecture



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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

SQL Management Base

The SQL management base (SMB) is a logical repository in the data dictionary that contains the SQL statement log, plan history, plan baselines, and SQL profiles. It also stores information that the optimizer can use to maintain or improve SQL performance.

The SMB is stored in the `SYSAUX` tablespace and has automatic space management (for example, periodic purging of unused plans). You can configure the SMB to change the plan retention policy and set space size limits. The database does not use SQL plan management and SQL tuning features when the `SYSAUX` tablespace is unavailable because the SMB is located entirely within this tablespace.

SQL Statement Log

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

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

SQL Profiles

The database creates SQL profiles when you invoke SQL Tuning Advisor, which you do typically only after a SQL statement has shown high-load symptoms. SQL profiles are primarily useful in providing ongoing resolution of optimizer mistakes that have led to suboptimal plans. Because the SQL profile mechanism is reactive, it cannot guarantee stable performance as drastic database changes occur.

SQL profiles are also implemented using hints, but these hints do not specify any specific plan. Rather, the hints correct miscalculations in the optimizer estimates that lead to suboptimal plans. Because a profile does not constrain the optimizer to any one plan, a SQL profile is more flexible than a SQL plan baseline.

SQL Plan History

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

In the latest Oracle Database release, the SMB stores the rows for the new plans that are added to the plan history of a SQL statement. The `DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE` function fetches and displays the plan from the SMB. Previously, when the plans were created, the function used to compile the SQL statement and generate the plan because the SMB did not store the rows.

SQL Plan Baselines

SQL Plan Management uses a mechanism called a SQL plan baseline. It is a subset of SQL Plan History that includes only the set of accepted plans generated for a SQL statement. In the typical use case, the database accepts a plan into the plan baseline only after verifying that the plan performs well. SQL plan baselines help improve the performance of SQL statements by ensuring that the optimizer uses only optimal plans. Each plan is implemented using a set of outline hints that fully specify a particular plan. SQL plan baselines prevent the optimizer from using suboptimal plans in the future.

Accepted Plans

A plan is accepted if and only if it is in the plan baseline. The plan history for a statement contains all plans, both accepted and unaccepted. After the optimizer generates the first accepted plan in a plan baseline, every subsequent unaccepted plan is added to the plan history, awaiting verification, but it is not added to the SQL plan baseline.

Enabled Plans

A plan is enabled only if it is used by the optimizer. The database automatically marks all plans in the plan history as enabled even if they are still unaccepted. You can manually change an enabled plan to a disabled plan, which means that the optimizer can no longer use the plan even if it is accepted.

Automatic SQL Tuning Task

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

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

Lesson Agenda

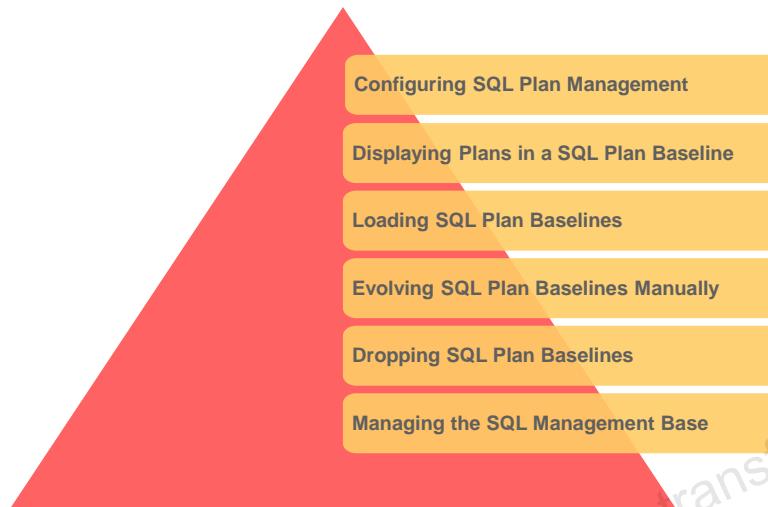
- Maintaining SQL Performance
- SQL Plan Management: Overview
- **Basic Tasks in SQL Plan Management**
- Adaptive SQL Plan Management
- Possible SQL Plan Manageability Scenarios
- Enterprise Manager and SQL Plan Baselines
- Loading Hinted Plans into SPM: Example



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Basic Tasks in SQL Plan Management



ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The basic tasks in using SQL plan management to prevent plan regressions and allow the optimizer to consider new plans are:

- **Configuring SQL Plan Management:** You can set initialization parameters to control whether the database captures and uses SQL plan baselines, and whether it evolves new plans.
- **Displaying Plans in SQL Plan Baselines:** You can use the `DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE` function to view the plans stored in the SQL plan baseline for a specific statement.
- **Loading SQL Plan Baselines:** You can manually load plans into SQL plan baselines. You can load plans from SQL tuning sets, the shared SQL area, or a staging table.
- **Evolving SQL Plan Baselines:** You can manually evolve plans into SQL plan baselines. You can use PL/SQL to verify the performance of specified plans and add them to plan baselines.
- **Dropping SQL Plan Baselines:** You can drop all or some plans in SQL plan baselines.
- **Managing the SQL Management Base:** You can alter disk space limits and change the length of the plan retention policy.

Configuring SQL Plan Management

- You control SQL plan management with initialization parameters.
 - `OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES=false`
 - `OPTIMIZER_USE_SQL_PLAN_BASELINES=true`
- You can configure the Automatic SPM Evolve Advisor Task using the `SET_EVOLVE_TASK_PARAMETER` procedure.
- You can set `ACCEPT_PLANS` to `true` (default), so that SQL plan management automatically accepts all plans recommended by the task.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

If you set the `OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES` initialization parameter to `true`, the database automatically creates an initial SQL plan baseline for any SQL statement that is not already in the plan history. This parameter does not control the automatic addition of newly discovered plans to a previously created SQL plan baseline.

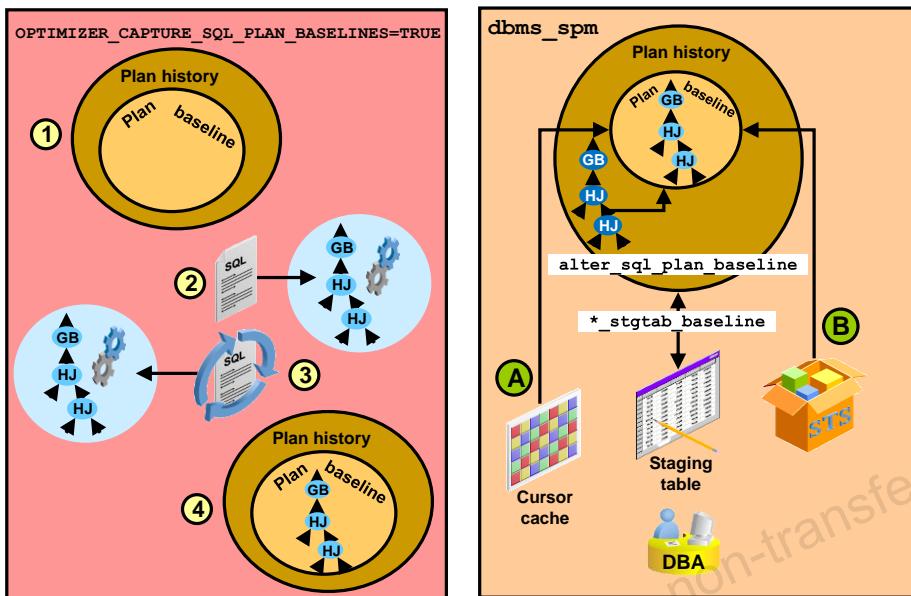
When you set the `OPTIMIZER_USE_SQL_PLAN_BASELINES` initialization parameter to `false`, the database does not use any plan baselines in the database.

The `DBMS_SPM` package enables you to configure automatic plan evolution by specifying the task parameters using the `SET_EVOLVE_TASK_PARAMETER` procedure. Because the task is owned by `SYS`, only `SYS` can set task parameters.

The `ACCEPT_PLANS` tuning task parameter specifies whether to accept recommended plans automatically. When `ACCEPT_PLANS` is `true` (default), SQL plan management automatically accepts all plans recommended by the task. When set to `false`, the task verifies the plans and generates a report of its findings, but does not evolve the plans.

Note: Automatic SPM Evolve Advisor Task is discussed in later slides.

Loading SQL Plan Baselines

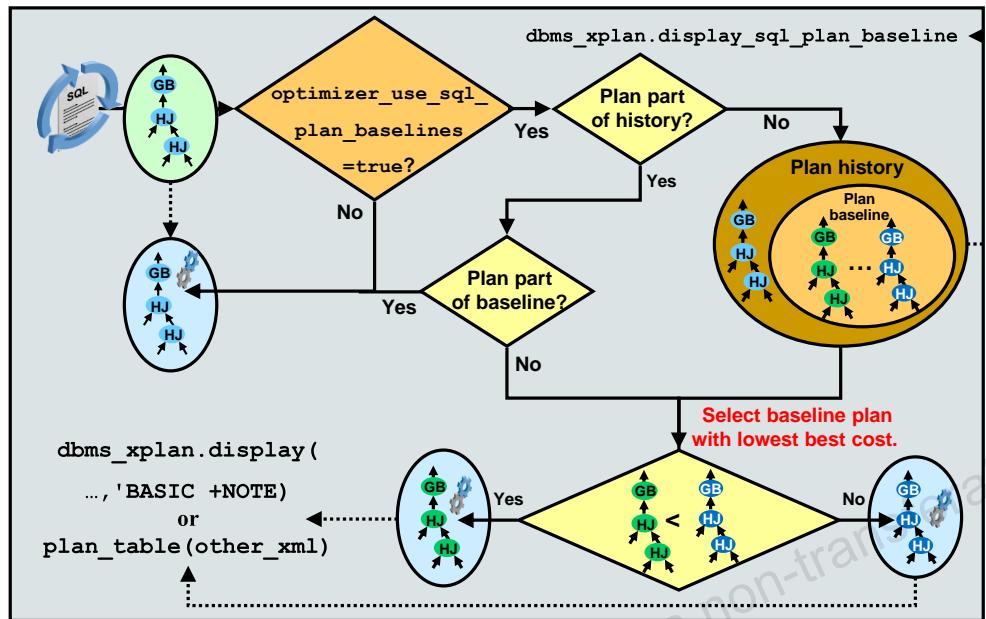


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

There are two ways to load SQL plan baselines in SQL Management Base:

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

SQL Plan Selection



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

The optimizer uses a comparative plan selection policy when a plan baseline exists for a SQL statement and the `OPTIMIZER_USE_SQL_PLAN_BASELINES` initialization parameter is set to `TRUE` (default value). SQL plan selection is the optimizer ability to detect plan changes based on stored plan history, and the use of SQL plan baselines to select plans to avoid potential performance regressions.

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

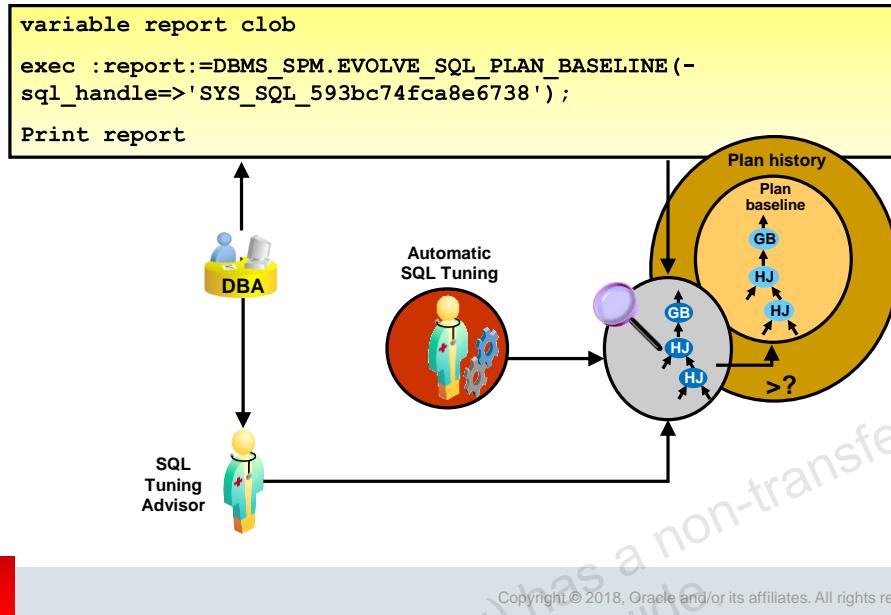
With SPM, the optimizer can produce a plan that could be either a best-cost plan or a baseline plan. This information is dumped in the `other_xml` column of the `plan_table` upon `EXPLAIN PLAN`. However, the optimizer can use only an accepted and enabled baseline plan.

In addition, you can use the new `dbms_xplan.display_sql_plan_baseline` function to display one or more execution plans for the specified `sql_handle` of a plan baseline. If `plan_name` is also specified, the corresponding execution plan is displayed.

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

Stored outlines can be migrated to SQL plan baselines by using the `MIGRATE_STORED_OUTLINE` procedure from the `DBMS_SPM` package. When the migration is complete, you should disable or drop the original stored outline by using the `DROP_MIGRATED_STORED_OUTLINE` procedure of the `DBMS_SPM` package.

Evolving SQL Plan Baselines



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

There are two ways to evolve SQL plan baselines:

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

Lesson Agenda

- Maintaining SQL Performance
- SQL Plan Management: Overview
- Basic Tasks in SQL Plan Management
- **Adaptive SQL Plan Management**
- Possible SQL Plan Manageability Scenarios
- Enterprise Manager and SQL Plan Baselines
- Loading Hinted Plans into SPM: Example



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Adaptive SQL Plan Management

- The new evolve auto task runs in the nightly maintenance window.
- It ranks all non-accepted plans and runs the evolve process for them. Newly found plans are ranked the highest.
- The database purges plans that have not been used for longer than the plan retention period. The default retention period is 53 weeks.
- The new task is `SYS_AUTO_SPM_EVOLVE_TASK`.
- Information about the task is in `DBA_ADVISOR_TASKS`.
- Use `DBMS_SPM.REPORT_AUTO_EVOLVE_TASK` to view the results of the auto job.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Starting in Oracle Database 12c, the database can use adaptive SPM. With adaptive SQL plan management, DBAs no longer have to manually run the verification or evolve process for non-accepted plans. They can go back days or weeks later and review the plans that were evolved during each nightly maintenance window. When automatic SQL tuning is in `COMPREHENSIVE` mode, it runs the verification or evolve process for all SQL statements that have non-accepted plans during the nightly maintenance window. All non-accepted plans are ranked and the evolve process is run for them. The newly found plans are ranked the highest.

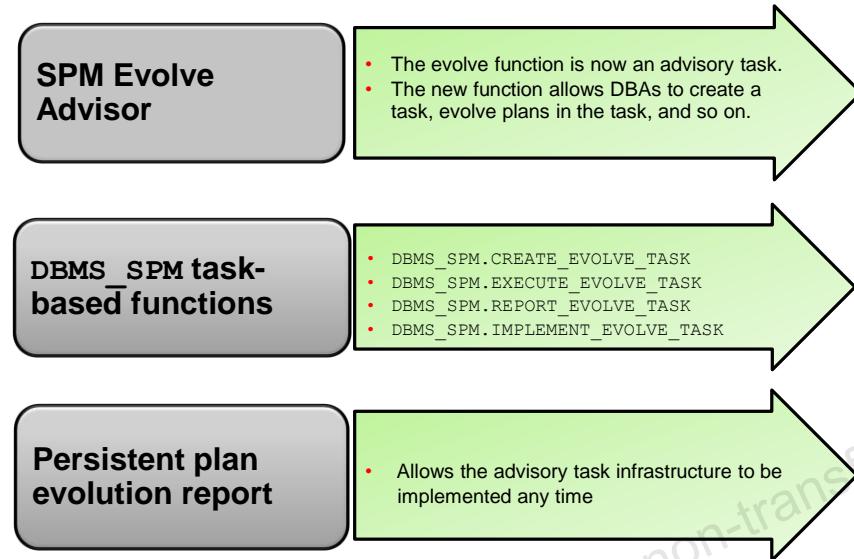
If the non-accepted plan performs better than the existing accepted plan (or plans) in the SQL plan baseline, the plan is automatically accepted and becomes usable by the optimizer.

After the verification is complete, a persistent report is generated detailing how the non-accepted plan performs compared to the accepted plan performance.

Because the evolve process is now an `AUTOTASK`, DBAs can also schedule their own evolve job at end time and use `DBMS_SPM.REPORT_AUTO_EVOLVE_TASK` to view the results of the auto job.

Automatic Plan Evolution of SQL Plan Management is a feature of the Tuning Pack.

Managing the SPM Evolve Advisor Task



ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

SPM Evolve Advisor is a SQL advisor that evolves plans that have recently been added to the SQL plan baseline. The advisor simplifies plan evolution by eliminating the manual chore of evolution.

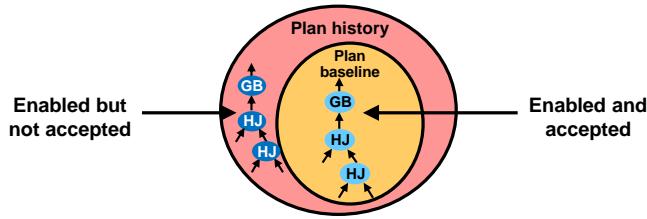
New functions introduced in the `DBMS_SPM` package work on the advisory task infrastructure. The new functions such as `DBMS_SPM.CREATE_EVOLVE_TASK`, `DBMS_SPM.EXECUTE_EVOLVE_TASK`, and so on allow database administrators (DBAs) to create a task, evolve one or more plans in the task, fetch the report of the task, and implement the results of the task. The task can be executed multiple times, and the report of a given task can be fetched multiple times. SPM can schedule an evolve task, as well as rerun an evolve task. There is also a function to implement the results of a task, which in this case would be to accept the currently non-accepted plans.

Using the task infrastructure allows the report of the evolution to be persistently stored in the advisory framework repository. It also allows Enterprise Manager to schedule SQL plan baselines evolution and fetch reports on demand. The new function supports HTML and XML reports in addition to TEXT. In Oracle Database 12c, the new task-based functions in `DBMS_SPM` retain the `time_limit` specification as a task parameter. It has the same default values as Oracle Database 11g. As for commit, the new evolve task does not accept any plans by default. Users can view the report by using the appropriate function, and then execute a new implement function to accept the successful plans.

The DBMS_SPM procedures and functions are used for managing plan evolution. Execute evolution tasks manually or schedule them to run automatically.

- ACCEPT_SQL_PLAN_BASELINE: This function accepts one recommendation to evolve a single plan into a SQL plan baseline.
- CREATE_EVOLVE_TASK: This function creates an advisor task to prepare the plan evolution of one or more plans for a specified SQL statement. The input parameters can be a SQL handle, plan name or a list of plan names, time limit, task name, and description.
- EXECUTE_EVOLVE_TASK: This function executes an evolution task. The input parameters can be the task name, execution name, and execution description. If not specified, the advisor generates the name, which is returned by the function.
- IMPLEMENT_EVOLVE_TASK: This function implements all recommendations for an evolve task. Essentially, this function is equivalent to using ACCEPT_SQL_PLAN_BASELINE for all recommended plans. The input parameters include task name, plan name, owner name, and execution name.
- REPORT_EVOLVE_TASK: This function displays the results of an evolve task as a CLOB. The input parameters include the task name and section of the report to include.
- SET_EVOLVE_TASK_PARAMETER: This function updates the value of an evolve task parameter. In this release, the only valid parameter is TIME_LIMIT.

Important SQL Plan Baseline Attributes



```
select signature, sql_handle, sql_text, plan_name, origin, enabled,
       accepted, fixed, autopurge
  from dba_sql_plan_baselines;
```

SIGNATURE	SQL_HANDLE	SQL_TEXT	PLAN_NAME	ORIGIN	ENA	ACC	FIX	AUT
8.062E+18	SYS_SQL_6fe2	select..	SQL_PLAN_6zsn...	AUTO-CAPTURE	YES	NO	NO	YES
8.062E+18	SYS_SQL_e23f	select..	SQL_PLAN_f4gy...	AUTO-CAPTURE	YES	YES	NO	YES
...								

```
exec :cnt := dbms_spm.alter_sql_plan_baseline(
      sql_handle => 'SYS_SQL_6fe28d438dfc352f',
      plan_name      => 'SQL_PLAN_6zsnd8f6zsd9g54bc8843',
      attribute_name  => 'ENABLED', attribute_value => 'NO');
```



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

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

- SIGNATURE, SQL_HANDLE, SQL_TEXT, and PLAN_NAME are important identifiers for search operations.
- ORIGIN allows you to determine whether the plan was automatically captured (AUTO-CAPTURE), manually evolved (MANUAL-LOAD), automatically evolved by SQL Tuning Advisor (MANUAL-SQLTUNE), or automatically evolved by Automatic SQL Tuning (AUTO-SQLTUNE).
- The ENABLED and ACCEPTED attributes must be set to YES or else the plan is not considered by the optimizer. The ENABLED attribute means that the plan is enabled for use by the optimizer. The ACCEPTED attribute means that the plan was validated as a good plan, either automatically by the system or manually when the user changes it to ACCEPTED. When a plan status changes to ACCEPTED, it will continue to be ACCEPTED until DBMS_SPM.ALTER_SQL_PLAN_BASELINE() is used to change its status. An ACCEPTED plan can be temporarily disabled by removing the ENABLED setting.

- **FIXED** means that the optimizer considers only those plans and not other plans. For example, if you have 10 baseline plans and three are marked **FIXED**, the optimizer uses the best plan only from these three, ignoring all the others. A SQL plan baseline is said to be **FIXED** if it contains at least one enabled fixed plan. If new plans are added to a fixed SQL plan baseline, they cannot be used until they are manually declared as **FIXED**. You can look at each plan's attributes by using the `DBA_SQL_PLAN_BASELINES` view, as shown in the slide. You can then use the `DBMS_SPM.ALTER_SQL_PLAN_BASELINE` function to change some of them. You can also remove plans or a complete plan history by using the `DBMS_SPM.DROP_SQL_PLAN_BASELINE` function. The example shown in the slide changes the **ENABLED** attribute of `SQL_PLAN_6zsnd8f6zsd9g54bc8843` to **NO**.

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

Lesson Agenda

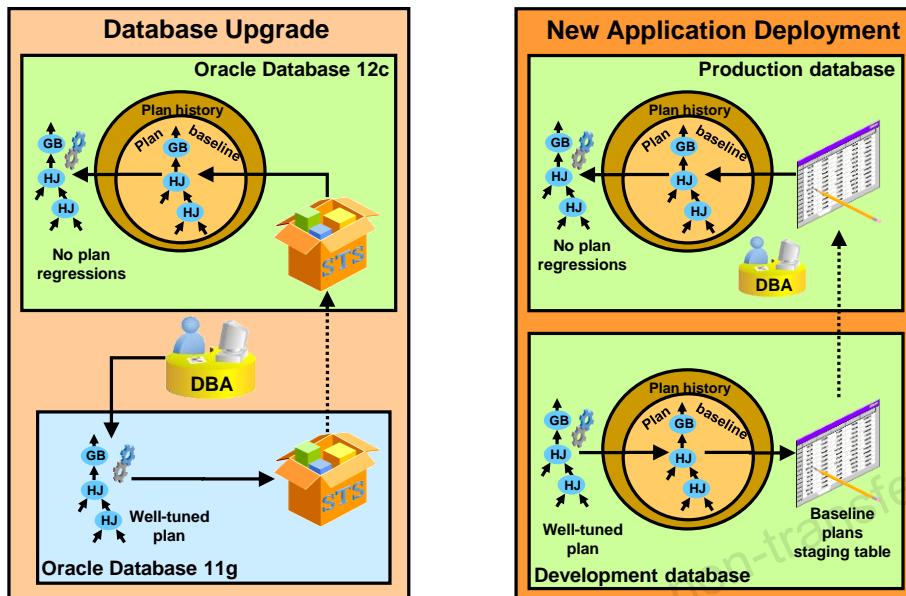
- Maintaining SQL Performance
- SQL Plan Management: Overview
- Basic Tasks in SQL Plan Management
- Adaptive SQL Plan Management
- **Possible SQL Plan Manageability Scenarios**
- Enterprise Manager and SQL Plan Baselines
- Loading Hinted Plans into SPM: Example



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Possible SQL Plan Manageability Scenarios



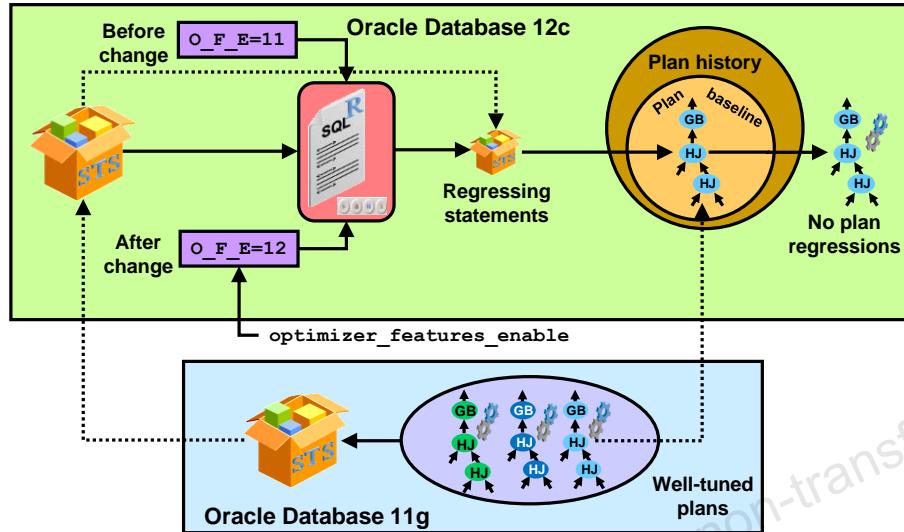
Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

- **Database upgrade:** Bulk SQL plan loading is especially useful when the system is being upgraded from an earlier version to Oracle Database 12c. For this, you can capture plans for a SQL workload into an SQL tuning set (STS) before the upgrade, and then load these plans from the STS into the SQL plan baseline immediately after the upgrade. This strategy can minimize plan regressions resulting from the use of the new optimizer version.
- **New application deployment:** The deployment of a new application module means the introduction of new SQL statements into the system. The software vendor can ship the application software along with the appropriate SQL plan baselines for the new SQL that is being introduced. Because of the plan baselines, the new SQL statements will initially run with the plans that are known to give good performance under a standard test configuration. However, if the customer system configuration is very different from the test configuration, the plan baselines can be evolved over time to produce better performance.

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

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

SQL Performance Analyzer and SQL Plan Baseline Scenario

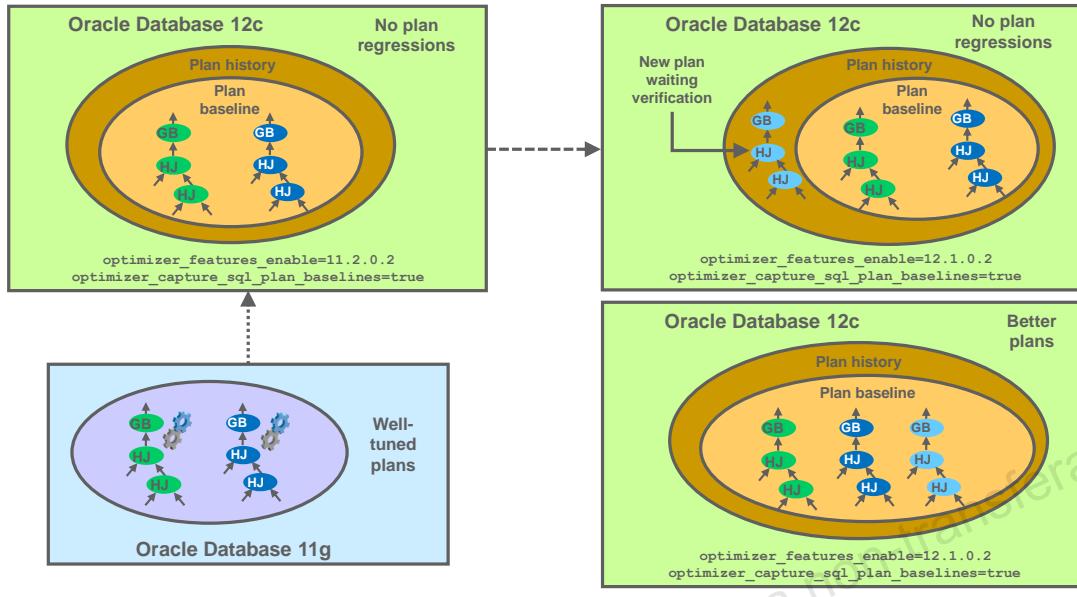


A variation of the first method described in the previous slide is through the use of SQL Performance Analyzer. You can capture pre-Oracle Database 12c plans in an STS and import them into Oracle Database 12c. Then set the `optimizer_features_enable (O_F_E)` initialization parameter to 11.2.0 to make the optimizer behave as if this were Oracle Database 11g. Next run SQL Performance Analyzer for the STS. When that is complete, set the `optimizer_features_enable` initialization parameter back to 12.1.0 and rerun SQL Performance Analyzer for the STS.

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

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

Loading a SQL Plan Baseline Automatically



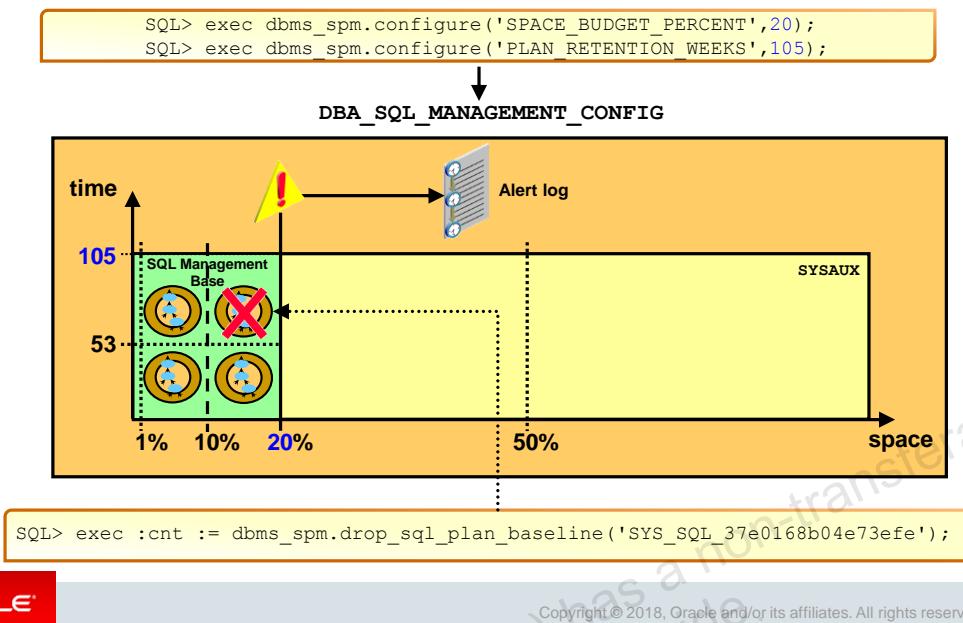
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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

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

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

Purging SQL Management Base Policy



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

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

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

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

Lesson Agenda

- Maintaining SQL Performance
- SQL Plan Management: Overview
- Basic Tasks in SQL Plan Management
- Adaptive SQL Plan Management
- Possible SQL Plan Manageability Scenarios
- Enterprise Manager and SQL Plan Baselines
- Loading Hinted Plans into SPM: Example



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Enterprise Manager and SQL Plan Baselines

The screenshot shows the Oracle Enterprise Manager Cloud Control 13c interface. In the top navigation bar, the database name is 'orcl / ORCLPDB'. The left sidebar has a 'Performance' section with various monitoring tools like Performance Home, Top Activity, ASH Analytics, and SQL Monitoring. Under the 'SQL' section, 'SQL Plan Control' is highlighted with a red box and a red arrow pointing to it. The main content area is titled 'SQL Plan Control' and shows a table for 'Jobs for SQL Plan Baselines'. The table has columns for 'Enabled', 'Accepted', 'Reproduced', 'Fixed', 'Auto Purge', 'Origin', 'Created', and 'Last Modified'. There are three rows in the table, each corresponding to a SQL plan baseline entry. At the bottom of the page, there is a note: 'TIP The table will display maximum of 2000 rows. Use search criteria to get the desired results.'

	SQL Text	Enabled	Accepted	Reproduced	Fixed	Auto Purge	Origin	Created	Last Modified
<input type="checkbox"/> SQL_PLAN_bvfhuhdg33rbed1a00	select signature, sql_handle, sql_text_plan_name,...	YES	YES	NO	NO	AUTO-CAPTURE	Feb 16, 2018 9:32:18 AM	Feb 16, 2018 9:32:18 AM	
<input type="checkbox"/> SQL_PLAN_4zzsm0f4zsdg0e080c0	select "LOAD_AUTO" from sh.sales where quantit...	YES	NO	YES	NO	AUTO-CAPTURE	Feb 16, 2018 9:31:47 AM	Feb 16, 2018 9:31:47 AM	
<input type="checkbox"/> SQL_PLAN_4zzsm0f4zsdg4e08843	select "LOAD_AUTO" from sh.sales where quantit...	YES	YES	NO	YES	AUTO-CAPTURE	Feb 16, 2018 9:23:53 AM	Feb 16, 2018 9:23:53 AM	

Use the SQL Plan Management page to manage SQL profiles, SQL patches, and SQL plan baselines from one location rather than from separate locations in Enterprise Manager.

You can also enable, disable, drop, pack, unpack, load, and evolve selected baselines.

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

To navigate to this page, click the Performance tab under the **orclpdb** database home page, and then click SQL Plan Control entry in the SQL section.

Lesson Agenda

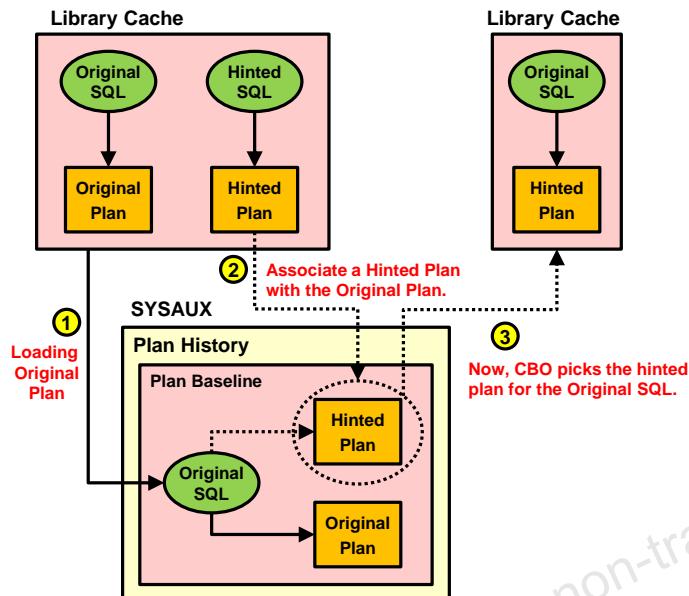
- Maintaining SQL Performance
- SQL Plan Management: Overview
- Basic Tasks in SQL Plan Management
- Adaptive SQL Plan Management
- Possible SQL Plan Manageability Scenarios
- Enterprise Manager and SQL Plan Baselines
- Loading Hinted Plans into SPM: Example



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Loading Hinted Plans into SPM: Example



ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

You can create SQL plan baselines for SQL statements:

- Coming from an application where the SQL statement cannot be modified
- That need hints to run a good execution plan

To load hinted plans into SPM, perform the following:

1. Capture the SQL plan baseline for the original SQL, which is the problem SQL identified.

```
var res number ;
exec :res := dbms_spm.load_plans_from_cursor_cache(sql_id =>
    '&original_sql_id', plan_hash_value =>
    '&original_plan_hash_value');
```

2. Add hints into the problem SQL and execute the hinted SQL.

3. Find the `SQL_ID` and `plan_hash_value` for the hinted SQL.

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

4. Verify that the original SQL baseline exists.

```
select sql_text, sql_handle, plan_name, enabled, accepted from
dba_sql_plan_baselines;
```

5. Associate the hinted execution plan with the original `sql_handle`.

```
var res number
exec :res := dbms_spm.load_plans_from_cursor_cache( -
    sql_id => '&hinted_SQL_ID', -
    plan_hash_value => &hinted_plan_hash_value, -
    sql_handle => '&sql_handle_for_original');
```

6. Verify that the new baseline was added.

```
select sql_text, sql_handle, plan_name, enabled, accepted from  
dba_sql_plan_baselines;
```

7. If the original plan capture is not needed initially, drop or disable it.

```
exec :res :=DBMS_SPM.DROP_SQL_PLAN_BASELINE  
('&original_sql_handle','&original_plan_name');
```

8. Re-execute the original SQL to verify that the SQL statement is now using the SQL plan baseline.

```
select SQL_PLAN_BASELINE from V$SQL where SQL_ID='&original_SQL_ID'
```

Quiz



When the OPTIMIZER_USE_SQL_PLAN_BASELINES parameter is set to TRUE, the optimizer:

- a. Does not develop an execution plan; it uses an accepted plan in the baselines
- b. Compares the plan that it develops with accepted plans in the baselines
- c. Compares the plan that it develops with enabled plans in the baselines
- d. Does not develop an execution plan; it uses enabled plans in the baselines
- e. Develops plans and adds them to the baselines as verified



ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Answer: b

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

Summary

In this lesson, you should have learned how to:

- Manage SQL performance through changes
- Set up SQL Plan Management
- Set up various SQL Plan Management scenarios
- Load hinted plans into SQL Plan Management



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Practice 13: Overview

This practice covers the use of SQL Plan Management.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Workshops

ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Objectives

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

- Identify some common problems in writing SQL statements
- Determine the common causes and observations
- Describe the different tuning approaches
- Describe the behavior of the optimizer
- Describe the effects of functions on index usage
- Tune sort operations for the ORDER BY clause
- Tune high parse time using a parse time reduction strategy
- Maintain stable execution plans over time



ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Overview

- This lesson is made up of nine workshops based on the SQL tuning techniques covered in preceding lessons.
- All workshops are based on problem solving.
- You must find ways to enhance performance through several tasks in each workshop.
- You also review concepts learned in each workshop through quizzes.
- Finally, by resolving the problems, you reinforce your learning.



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Workshop 1

Introduction:

- In this workshop, you must find a workaround to enhance performance.
- You analyze a poorly written SQL statement and perform additional tasks such as creating a function-based index, redesigning a simple table, and rewriting the SQL statement.



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Workshop 2

Introduction:

- This workshop is about reviewing the *execution steps of a SQL statement* section.
- You review the SQL statement that uses a bind variable in the indexed column.
- You analyze two execution plans of the SQL statement by using the EXPLAIN PLAN command and V\$ view.
- You should be able to describe why the two execution plans are different for the same SQL statement.



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Workshop 3

Introduction:

- This workshop shows the possible use of an index in the ORDER BY clause to tune a sort operation.
- You perform several tasks to be able to use the indexed ORDER BY column.
- After all the tasks are complete, you verify if the index always produces a better plan cost.



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Workshop 4

Introduction:

- In this workshop, you assume that you have identified a poorly running SQL statement.
- You already noticed that the root cause of the issue is the table design. However, re-creating the table is not an option at this point. The table is already in use.
- Try to tune the SQL statement by rewriting it.



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Workshop 5

Introduction:

- In this workshop, you write several SQL statements that return the same output to see which SQL statement is more efficient.
- You also examine the effects of changing column order in a composite index.



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Workshop 6 and 7 (Optional)

Introduction:

- In these workshops, you review the execution plan and several sections in an event 10053 trace file.
- You interpret the information to understand the optimizer's decision.
- There are several questions on cost model, selectivity, and cardinality. You answer these questions.
- Finally, you learn how to use the information in the 10053 file to tune the SQL statement.



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Workshop 8

Introduction:

- You have identified a slow-running SQL statement. It was run 534 times in a certain time period. You noticed that a different literal was used in each execution. This caused the system to parse the same statement 534 times with a high parse time, which is not efficient.
- You perform several tasks to tune the high parse time.
- What tuning strategy could be used?



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Workshop 9

Introduction:

- Oracle has introduced several methods to maintain the stable execution plans of SQL statements such as hints, stored outlines, and SQL profile. In Oracle 11g, you can use a better way to manage the execution plan, called the SQL Plan Baseline. This feature allows the optimizer to make the right decision in choosing a verified or tested execution plan in SQL Plan Baseline even if the optimizer generates a new plan during hard parse.
- In this workshop, you learn how to use this feature to associate a hinted execution plan with a hard-coded SQL statement.

ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Summary

In this lesson, you should have learned how to:

- Identify some common problems in writing SQL statements
- Determine the common causes and observations
- Describe the different tuning approaches
- Describe the behavior of the optimizer
- Describe the effects of functions on index usage
- Tune sort operations for the ORDER BY clause
- Tune high parse time using a parse time reduction strategy
- Maintain stable execution plans over time



ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

A

Using SQL Developer

ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- List the key features of Oracle SQL Developer
- Identify the menu items of Oracle SQL Developer
- Create a database connection
- Manage database objects
- Use SQL Worksheet
- Save and run SQL scripts
- Create and save reports
- Browse the data modeling options in SQL Developer



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

What Is Oracle SQL Developer?

- Oracle SQL Developer is a graphical tool that enhances productivity and simplifies database development tasks.
- You can connect to any target Oracle database schema by using standard Oracle database authentication.



SQL Developer

ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Oracle SQL Developer is a free graphical tool designed to improve your productivity and simplify the development of everyday database tasks. With just a few clicks, you can easily create and debug stored procedures, test SQL statements, and view optimizer plans.

SQL Developer, which is the visual tool for database development, simplifies the following tasks:

- Browsing and managing database objects
- Executing SQL statements and scripts
- Editing and debugging PL/SQL statements
- Creating reports

You can connect to any target Oracle database schema by using standard Oracle database authentication. When connected, you can perform operations on objects in the database.

SQL Developer is the interface to administer the Oracle Application Express Listener. The new interface enables you to specify global settings and multiple database settings with different database connections for the Application Express Listener. SQL Developer provides the option to drag objects by table or column name onto the worksheet. It provides improved DB Diff comparison options, GRANT statement support in the SQL editor, and DB Doc reporting. Additionally, SQL Developer includes support for Oracle Database 12c features.

SQL Developer: Specifications

- Is shipped along with Oracle Database 12c Release 2
- Is developed in Java
- Supports Windows, Linux, and Mac OS X platforms
- Enables default connectivity by using the JDBC Thin driver
- Connects to Oracle Database version 9.2.0.1 and later



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

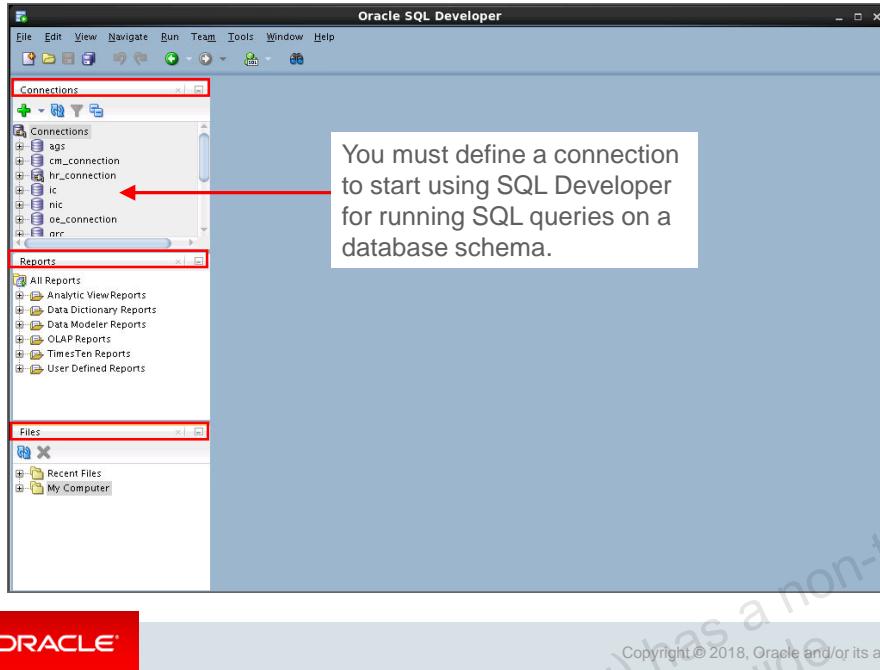
Oracle SQL Developer is shipped along with Oracle Database 12c Release 2 by default. SQL Developer is developed in Java, leveraging the Oracle JDeveloper integrated development environment (IDE). Therefore, it is a cross-platform tool. The tool runs on Windows, Linux, and Mac operating system (OS) X platforms.

The default connectivity to the database is through the Java Database Connectivity (JDBC) Thin driver, and therefore, no Oracle Home is required. SQL Developer does not require an installer; you can simply unzip the downloaded file. With SQL Developer, users can connect to Oracle Databases 9.2.0.1 and later, and all Oracle database editions, including Express Edition.

Note: For the latest version of Oracle SQL Developer, download the ZIP file from the following location: <http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html>

For instructions on how to install SQL Developer, see the following website:
<http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html>

SQL Developer 17.2 Interface



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The SQL Developer interface contains three main navigation tabs, from left to right:

- **Connections:** By using this tab, you can browse database objects and users to which you have access.
- **Reports:** Identified by the Reports icon, this tab enables you to run predefined reports or create and add your own reports.
- **Files:** Identified by the Files folder icon, this tab enables you to access files from your local machine without having to use the File > Open menu.

General Navigation and Use

SQL Developer uses the left side for navigation to find and select objects, and the right side to display information about selected objects. You can customize many aspects of the appearance and behavior of SQL Developer by setting preferences.

Note: You need to define at least one connection to be able to connect to a database schema and issue SQL queries or run procedures and functions.

Menus

The following menus contain standard entries, plus entries for features that are specific to SQL Developer:

- **View:** Contains options that affect what is displayed in the SQL Developer interface
- **Navigate:** Contains options for navigating to panes and for executing subprograms
- **Run:** Contains the Run File and Execution Profile options that are relevant when a function or procedure is selected, and also debugging options
- **Versioning:** Provides integrated support for the following versioning and source control systems: Concurrent Versions System (CVS) and Subversion
- **Tools:** Invokes SQL Developer tools such as SQL*Plus, Preferences, and SQL Worksheet. It also contains options related to migrating third-party databases to Oracle.

Note: The Run menu also contains options that are relevant when a function or procedure is selected for debugging.

Creating a Database Connection

- You must have at least one database connection to use SQL Developer.
- You can create and test connections for:
 - Multiple databases
 - Multiple schemas
- SQL Developer automatically imports any connections defined in the `tnsnames.ora` file on your system.
- You can export connections to an Extensible Markup Language (XML) file.
- Each additional database connection created is listed in the Connections Navigator hierarchy.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

A connection is a SQL Developer object that specifies the necessary information for connecting to a specific database as a specific user of that database. To use SQL Developer, you must have at least one database connection, which may be existing, created, or imported.

You can create and test connections for multiple databases and for multiple schemas.

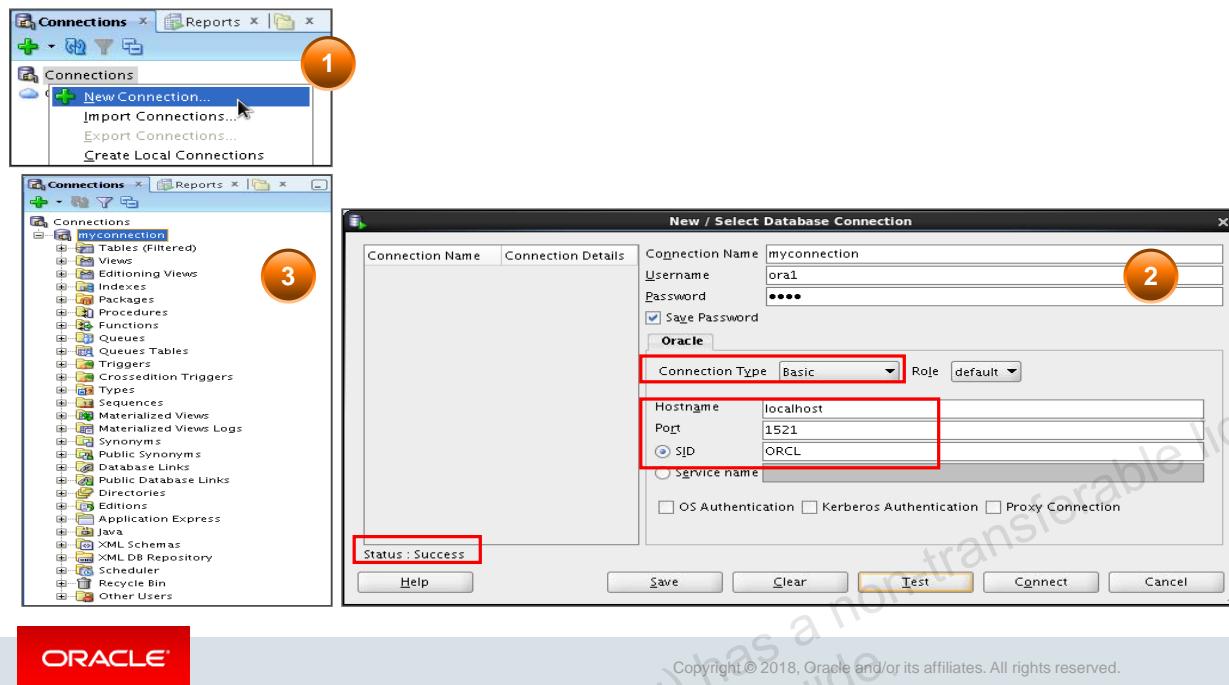
By default, the `tnsnames.ora` file is located in the `$ORACLE_HOME/network/admin` directory, but it can also be in the directory specified by the `TNS_ADMIN` environment variable or registry value. When you start SQL Developer and open the Database Connections dialog box, SQL Developer automatically imports any connections defined in the `tnsnames.ora` file on your system.

Note: With Windows, if the `tnsnames.ora` file exists, but its connections are not being used by SQL Developer, define `TNS_ADMIN` as a system environment variable.

You can export connections to an XML file so that you can reuse it.

You can create additional connections as different users to the same database or to connect to the different databases.

Creating a Database Connection



To create a database connection, perform the following steps:

1. On the Connections tabbed page, right-click Connections and select New Connection.
2. In the New/Select Database Connection window, enter the connection name. Enter the username and password of the schema that you want to connect to.
 - a. From the Role drop-down list, you can select either *default* or *SYSDBA*. (You choose *SYSDBA* for the *sys* user or any user with database administrator privileges.)
 - b. You can select the connection type as:
 - **Basic:** In this type, enter host name and SID for the database that you want to connect to. Port is already set to 1521. You can also choose to enter the Service name directly if you use a remote database connection.
 - **TNS:** You can select any one of the database aliases imported from the *tnsnames.ora* file.
 - **LDAP:** You can look up database services in Oracle Internet Directory, which is a component of Oracle Identity Management.
 - **Advanced:** You can define a custom Java Database Connectivity (JDBC) URL to connect to the database.
 - **Local/Bequeath:** If the client and database exist on the same computer, a client connection can be passed directly to a dedicated server process without going through the listener.

- c. Click Test to ensure that the connection has been set correctly.
- d. Click Connect.

If you select the Save Password check box, the password is saved to an XML file. So, after you close the SQL Developer connection and open it again, you are not prompted for the password.

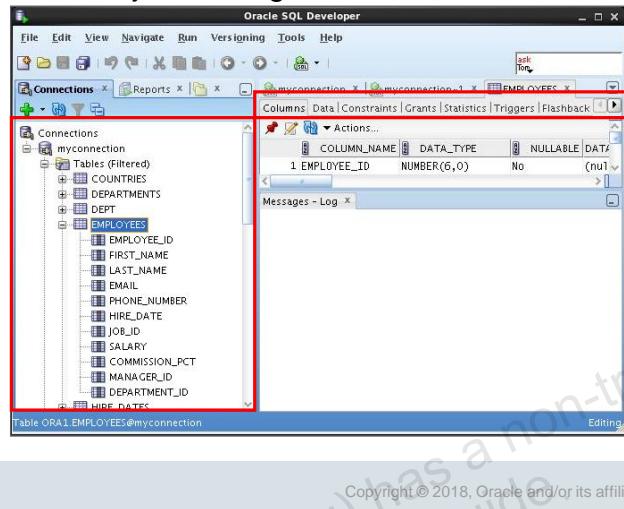
3. The connection gets added in the Connections Navigator. You can expand the connection to view the database objects and view object definitions (dependencies, details, statistics, and so on).

Note: From the same New>Select Database Connection window, you can define connections to non-Oracle data sources by using the Access, MySQL, and SQL Server tabs. However, these connections are read-only connections that enable you to browse objects and data in that data source.

Browsing Database Objects

Use the Connections Navigator to:

- Browse through many objects in a database schema
- Review the definitions of objects at a glance



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

After you create a database connection, you can use the Connections Navigator to browse through many objects in a database schema, including tables, views, indexes, packages, procedures, triggers, and types.

SQL Developer uses the left side for navigation to find and select objects, and the right side to display information about the selected objects. You can customize many aspects of the appearance of SQL Developer by setting preferences.

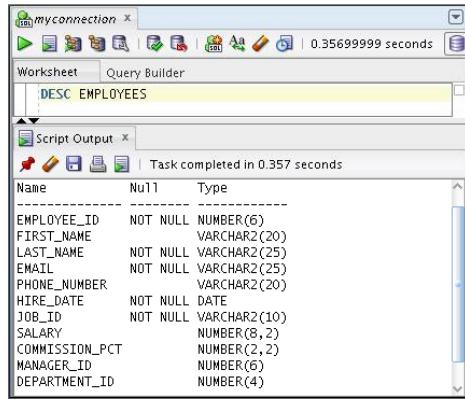
You can see the definitions of the objects broken up into tabs of information that is pulled out of the data dictionary. For example, if you select a table in the Navigator, details about columns, constraints, grants, statistics, triggers, and so on are displayed on an easy-to-read tabbed page.

If you want to see the definition of the EMPLOYEES table as shown in the slide, perform the following:

1. Expand the Connections node in the Connections Navigator.
2. Expand Tables.
3. Click EMPLOYEES. By default, the Columns tab is selected. It shows the column description of the table. Using the Data tab, you can view the table data and also enter new rows, update data, and commit these changes to the database.

Displaying the Table Structure

Use the `DESCRIBE` command to display the structure of a table:



The screenshot shows the Oracle SQL Developer interface. In the top-left corner, there's a red button with the word "ORACLE". The main window has two tabs: "Worksheet" and "Query Builder". The "Worksheet" tab is active and contains the command `DESC EMPLOYEES`. Below the command, the output shows the structure of the `EMPLOYEES` table:

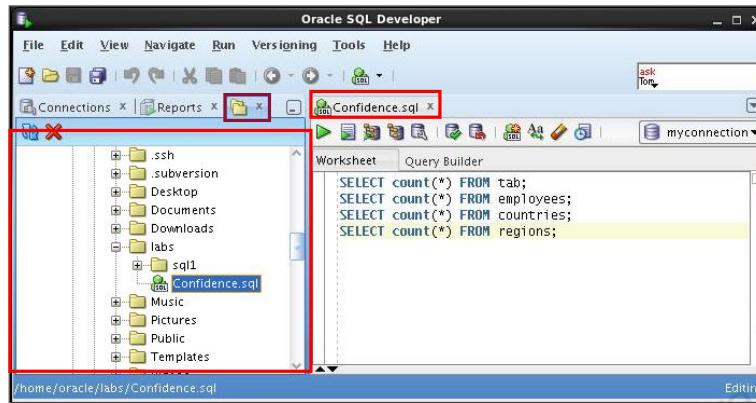
Name	Null	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

In SQL Developer, you can also display the structure of a table by using the `DESCRIBE` command. The result of the command is a display of column names and data types, as well as an indication of whether a column must contain data.

Browsing Files

Use the File Navigator to explore the file system and to open system files.



ORACLE®

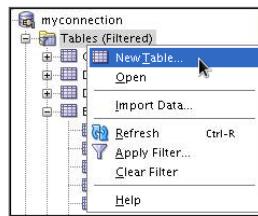
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

You can use the File Navigator to browse and open system files.

- To view the File Navigator, click the View tab and select Files, or select View > Files.
- To view the contents of a file, double-click a file name to display its contents in the SQL Worksheet area.

Creating a Schema Object

- SQL Developer supports the creation of any schema object by:
 - Executing a SQL statement in SQL Worksheet
 - Using the context menu
- Edit the objects by using an edit dialog box or one of the many context-sensitive menus.
- View the data definition language (DDL) for adjustments such as creating a new object or editing an existing schema object.



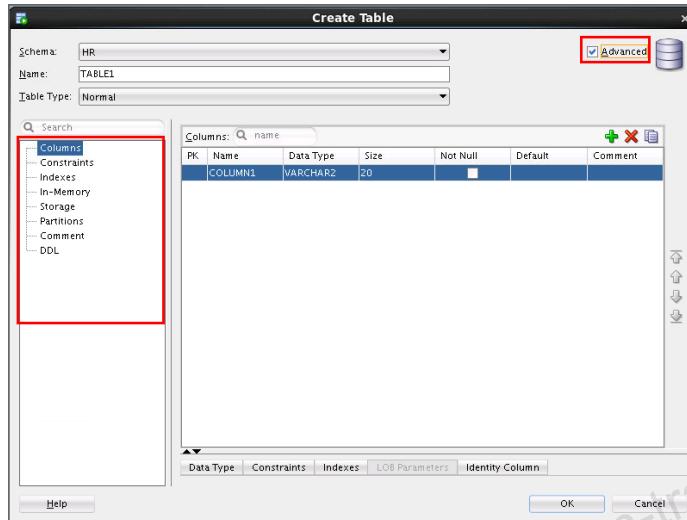
ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

As new objects are created or existing objects are edited, the DDL for those adjustments is available for review. An Export DDL option is available if you want to create the full DDL for one or more objects in the schema.

The slide shows how to create a table by using the context menu. To open a dialog box for creating a new table, right-click Tables and select New Table. The dialog boxes to create and edit database objects have multiple tabs, each reflecting a logical grouping of properties for that type of object.

Creating a New Table: Example



ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In the Create Table dialog box, if you do not select the Advanced check box, you can create a table quickly by specifying columns and some frequently used features.

If you select the Advanced check box, the Create Table dialog box changes to one with multiple options, in which you can specify an extended set of features while you create the table.

The example in the slide shows how to create the DEPENDENTS table by selecting the Advanced check box.

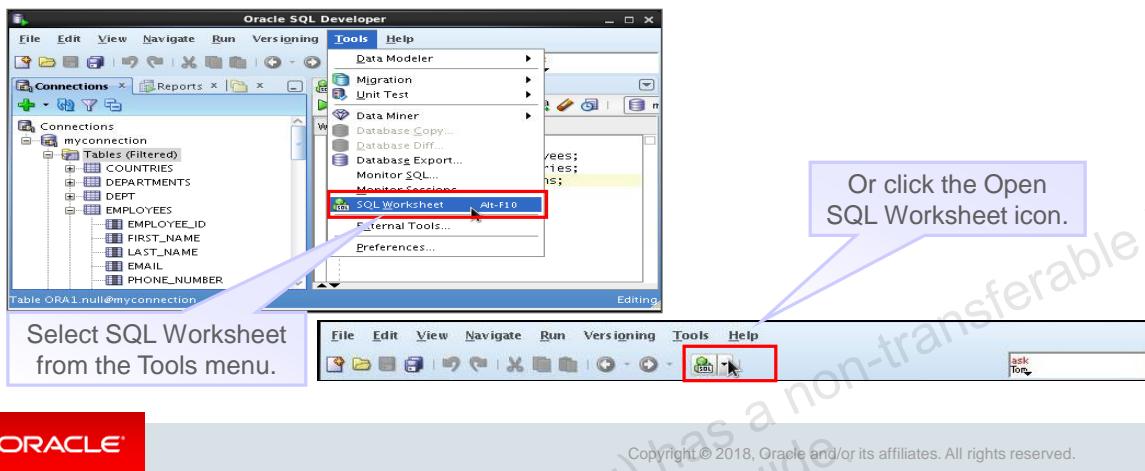
To create a new table, perform the following steps:

1. In the Connections Navigator, right-click Tables and select Create TABLE.
2. In the Create Table dialog box, select Advanced.
3. Specify the column information.
4. Click OK.

Although it is not required, you should also specify a primary key by using the Primary Key tab in the dialog box. Sometimes, you may want to edit the table that you have created; to do so, right-click the table in the Connections Navigator and select Edit.

Using the SQL Worksheet

- Use the SQL Worksheet to enter and execute SQL, PL/SQL, and SQL *Plus statements.
- Specify any actions that can be processed by the database connection associated with the worksheet.



When you connect to a database, a SQL Worksheet window for that connection automatically opens. You can use the SQL Worksheet to enter and execute SQL, PL/SQL, and SQL*Plus statements. The SQL Worksheet supports SQL*Plus statements to a certain extent. SQL*Plus statements that are not supported by the SQL Worksheet are ignored and not passed to the database.

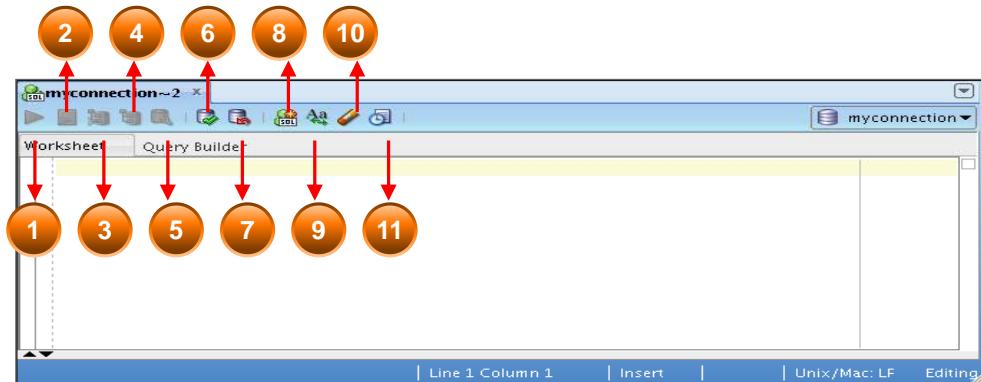
You can specify the actions that can be processed by the database connection associated with the worksheet, such as:

- Creating a table
- Inserting data
- Creating and editing a trigger
- Selecting data from a table
- Saving the selected data to a file

You can display a SQL Worksheet by using one of the following:

- Select Tools > SQL Worksheet.
- Click the Open SQL Worksheet icon.

Using the SQL Worksheet



ORACLE®

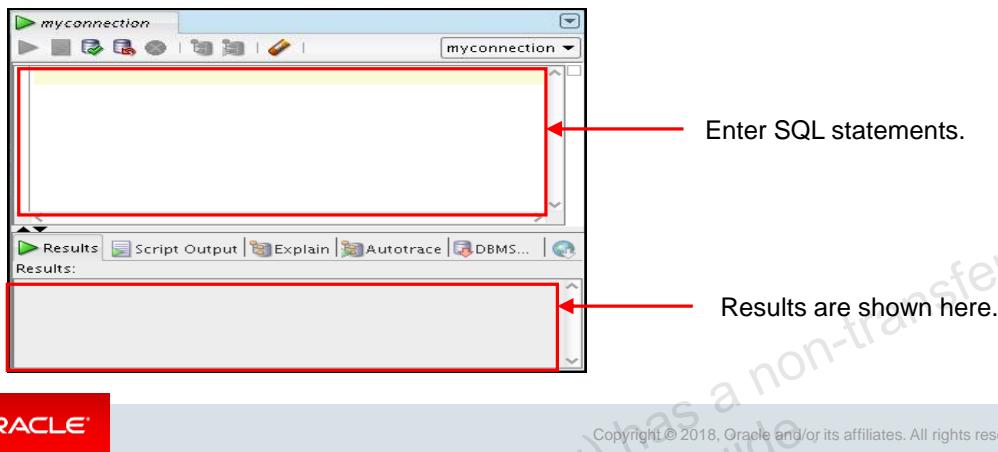
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

You may want to use the shortcut keys or icons to perform certain tasks such as executing a SQL statement, running a script, and viewing the history of the SQL statements that you have executed. You can use the SQL Worksheet toolbar that contains icons to perform the following tasks:

1. **Run Statement:** Executes the statement where the cursor is located in the Enter SQL Statement box. You can use bind variables in the SQL statements, but not substitution variables.
2. **Run Script:** Executes all the statements in the Enter SQL Statement box by using the Script Runner. You can use substitution variables in the SQL statements, but not bind variables.
3. **Autotrace:** Generates trace information for the statement
4. **Explain Plan:** Generates an execution plan, which you can see by clicking the Explain tab
5. **SQL Tuning Advisory:** Analyzes high-volume SQL statements and offers tuning recommendations
6. **Commit:** Writes any changes to the database and ends the transaction
7. **Rollback:** Discards any changes to the database without writing them to the database, and ends the transaction
8. **Unshared SQL Worksheet:** Creates a separate unshared SQL Worksheet for a connection
9. **To Upper/Lower/InitCap:** Changes the selected text to uppercase, lowercase, or sentence case respectively
10. **Clear:** Erases the statement or statements in the Enter SQL Statement box
11. **SQL History:** Displays a dialog box with information about the SQL statements that you have executed

Using the SQL Worksheet

- Use the SQL Worksheet to enter and execute SQL, PL/SQL, and SQL*Plus statements.
- Specify any actions that can be processed by the database connection associated with the worksheet.

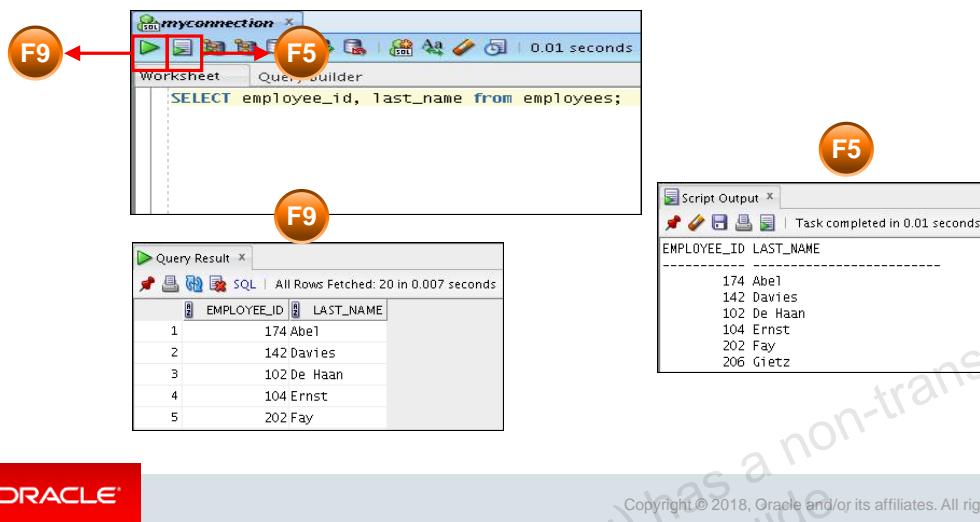


When you connect to a database, a SQL Worksheet window for that connection automatically opens. You can use the SQL Worksheet to enter and execute SQL, PL/SQL, and SQL*Plus statements. All SQL and PL/SQL commands are supported as they are passed directly from the SQL Worksheet to the Oracle database. The SQL*Plus commands that are used in SQL Developer must be interpreted by the SQL Worksheet before being passed to the database.

The SQL Worksheet currently supports a number of SQL*Plus commands. Commands that are not supported by the SQL Worksheet are ignored and not sent to the Oracle database. Through the SQL Worksheet, you can execute the SQL statements and some of the SQL*Plus commands.

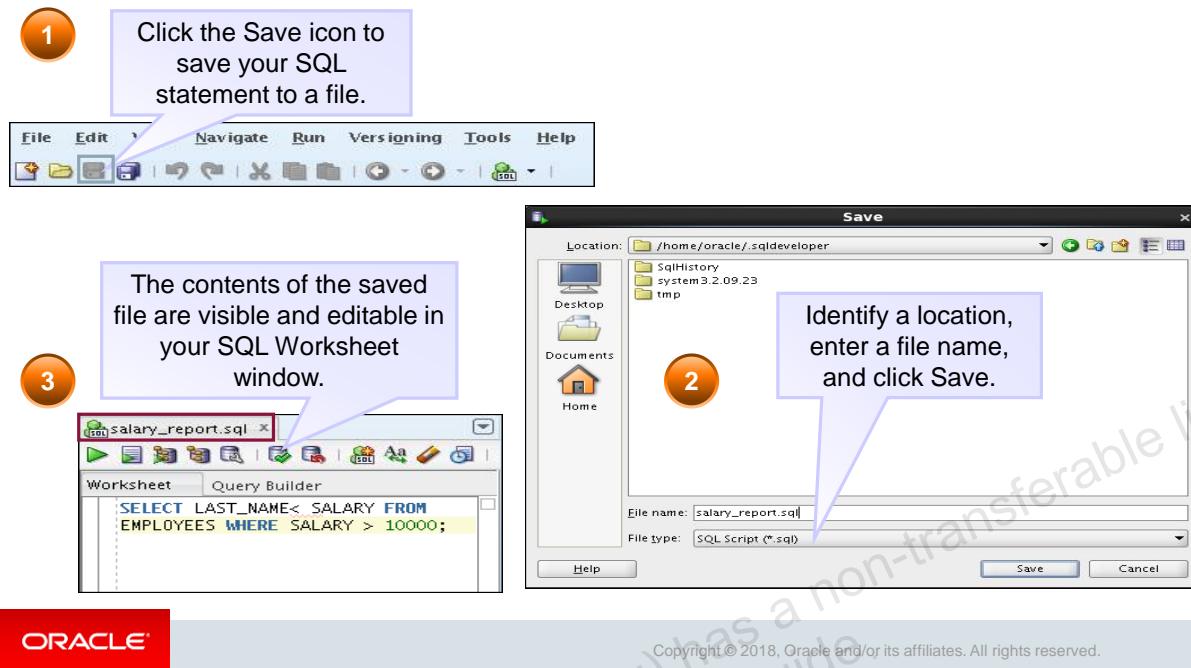
Executing SQL Statements

Use the Enter SQL Statement box to enter single or multiple SQL statements.



The example in the slide shows the difference in output for the same query when the F9 key or Execute Statement is used versus the output when F5 or Run Script is used.

Saving SQL Scripts



You can save your SQL statements from the SQL Worksheet to a text file. To save the contents of the Enter SQL Statement box, perform the following steps:

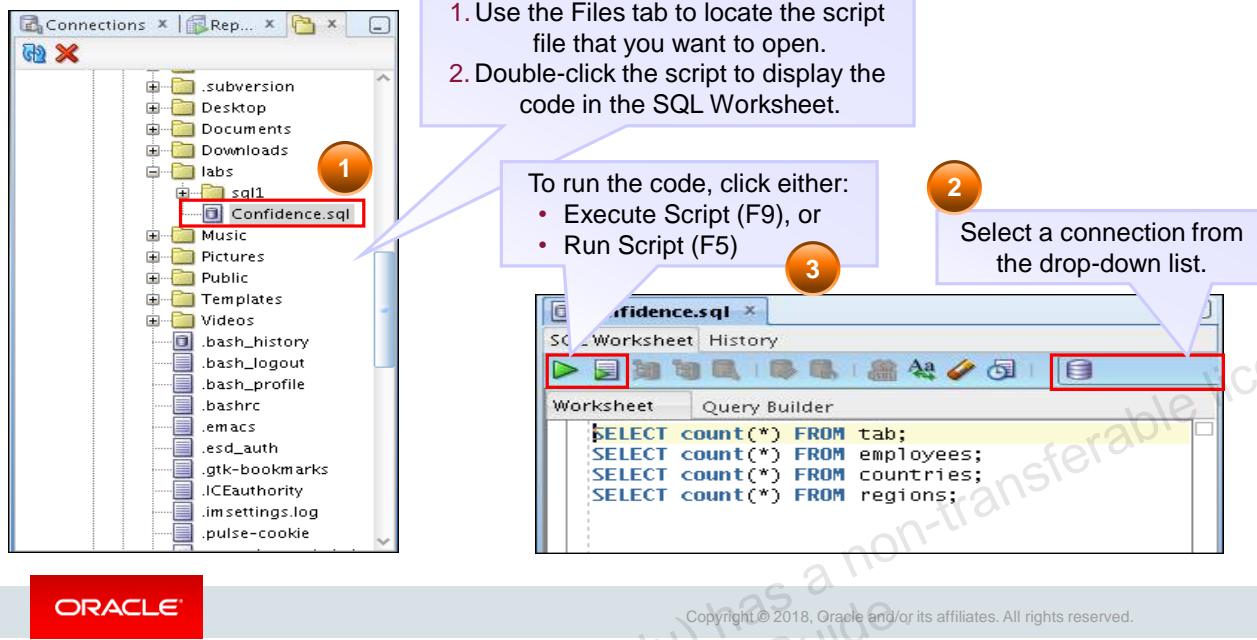
1. Click the Save icon or use the File > Save menu item.
2. In the Save dialog box, enter a file name and the location where you want the file saved.
3. Click Save.

After you save the contents to a file, the Enter SQL Statement window displays a tabbed page of your file contents. You can have multiple files open at the same time. Each file is displayed as a tabbed page.

Script Pathing

You can select a default path to look for scripts and to save scripts. Under Tools > Preferences > Database > Worksheet Parameters, enter a value in the “Select default path to look for scripts” field.

Executing Saved Script Files: Method 1



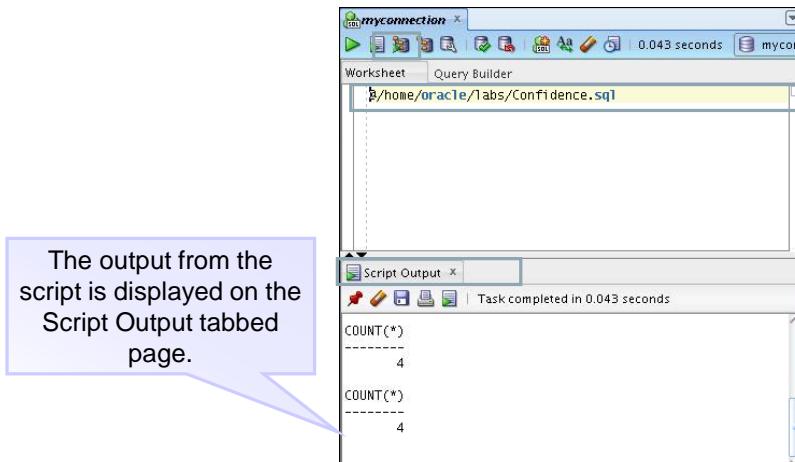
To open a script file and display the code in the SQL Worksheet area, perform the following:

1. In the Files Navigator, select (or navigate to) the script file that you want to open.
2. Double-click the file to open it. The code of the script file is displayed in the SQL Worksheet area.
3. Select a connection from the Connection drop-down list.
4. To run the code, click the Run Script (F5) icon on the SQL Worksheet toolbar. If you have not selected a connection from the Connection drop-down list, a Connection dialog box appears. Select the connection that you want to use for the script execution.

Alternatively, you can also do the following:

1. Select File > Open. The Open dialog box is displayed.
2. In the Open dialog box, select (or navigate to) the script file that you want to open.
3. Click Open. The code of the script file is displayed in the SQL Worksheet area.
4. Select a connection from the Connection drop-down list.
5. To run the code, click the Run Script (F5) icon on the SQL Worksheet toolbar. If you have not selected a connection from the Connection drop-down list, a Connection dialog box appears. Select the connection that you want to use for the script execution.

Executing Saved Script Files: Method 2



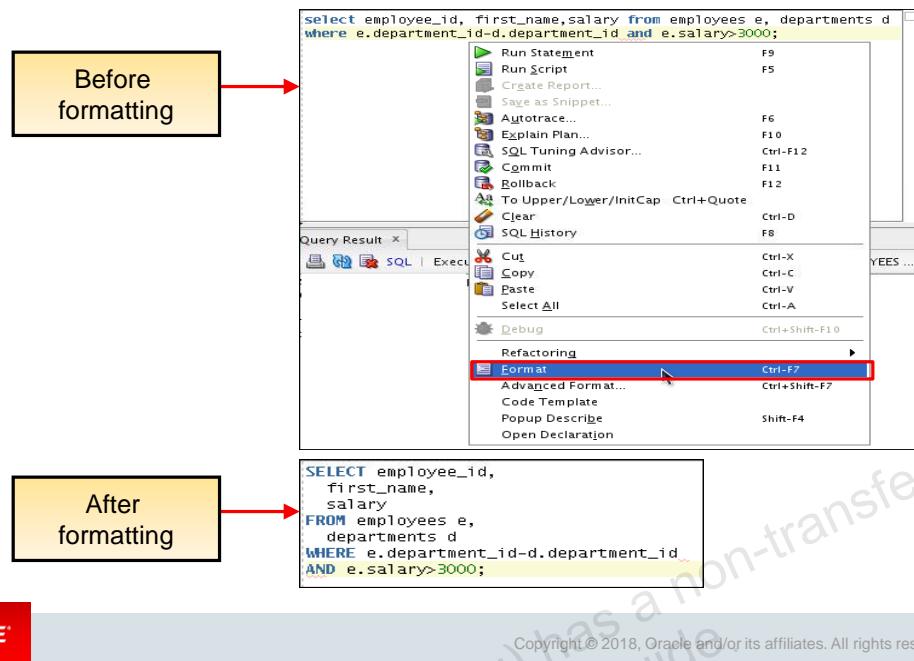
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

To run a saved SQL script, perform the following:

1. Use the @ command, followed by the location, and name of the file you want to run, in the Enter SQL Statement window.
2. Click the Run Script icon.

The results from running the file are displayed on the Script Output tabbed page. You can also save the script output by clicking the Save icon on the Script Output tabbed page. The Windows Save dialog box appears and you can identify a name and location for your file.

Formatting the SQL Code



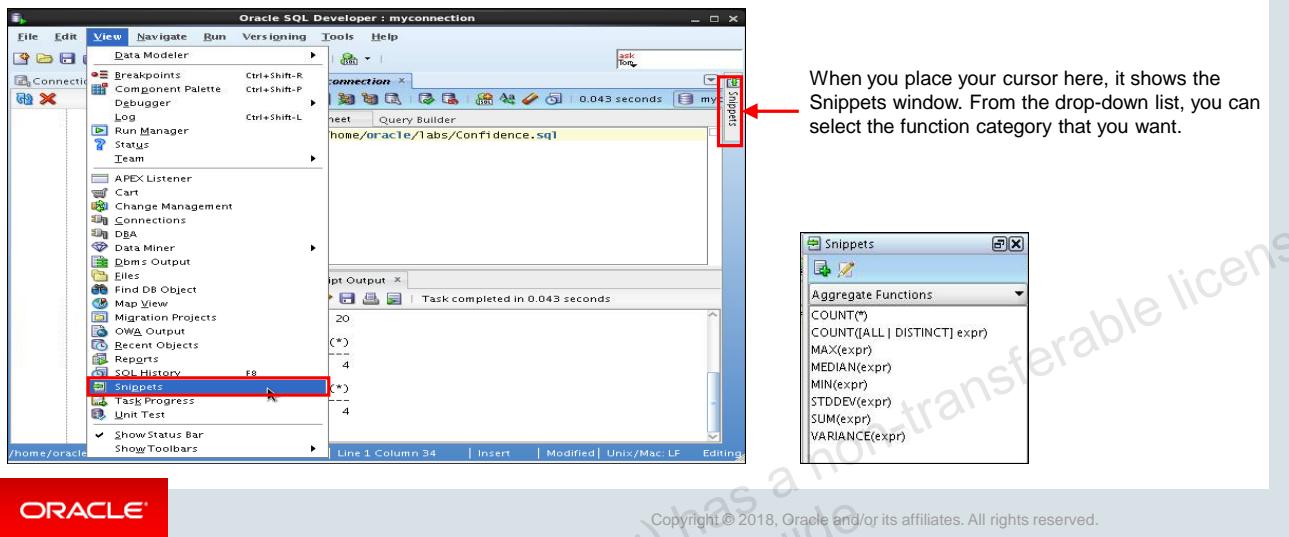
You may want to format the indentation, spacing, capitalization, and line separation of the SQL code. SQL Developer has a feature for formatting SQL code.

To format the SQL code, right-click in the statement area and select Format.

In the example in the slide, before formatting, the SQL code has the keywords not capitalized and the statement not properly indented. After formatting, the SQL code is beautified with the keywords capitalized and the statement properly indented.

Using Snippets

Snippets are code fragments that may be just syntax or examples.



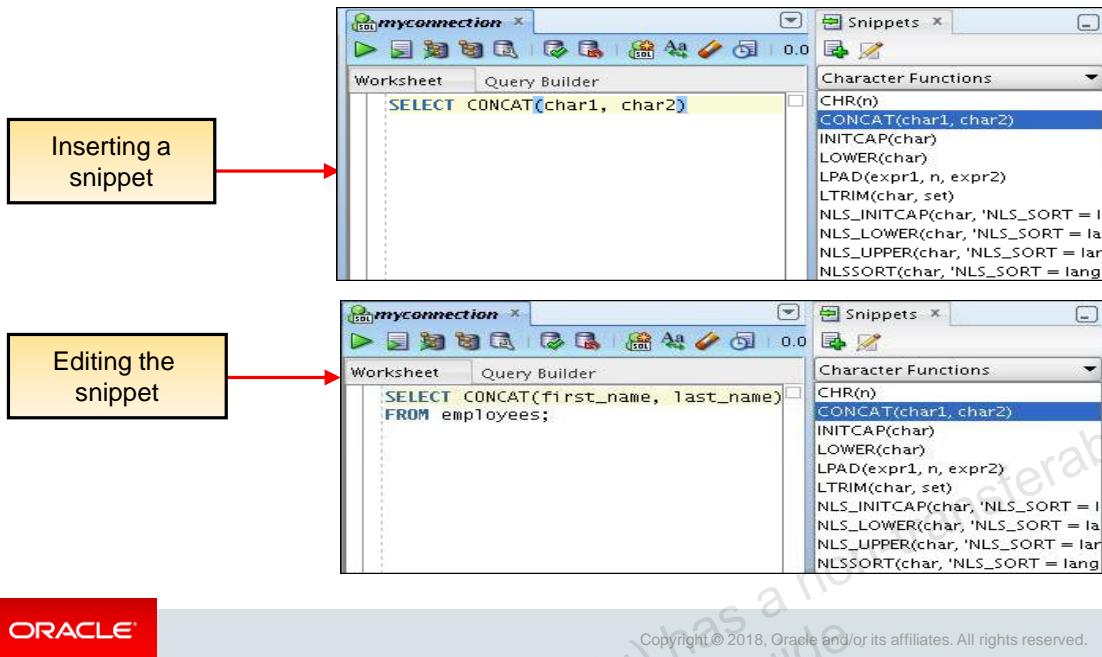
When you place your cursor here, it shows the Snippets window. From the drop-down list, you can select the function category that you want.

You may want to use certain code fragments when you use the SQL Worksheet or create or edit a PL/SQL function or procedure. SQL Developer has a feature called Snippets. Snippets are code fragments such as SQL functions, optimizer hints, and miscellaneous PL/SQL programming techniques. You can drag snippets to the Editor window.

To display Snippets, select View > Snippets.

The Snippets window is displayed on the right. You can use the drop-down list to select a group. A Snippets button is placed in the right window margin, so that you can display the Snippets window if it becomes hidden.

Using Snippets: Example



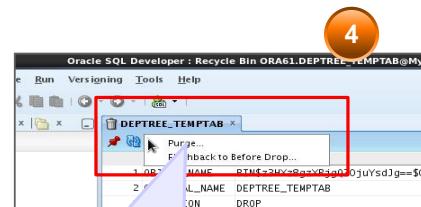
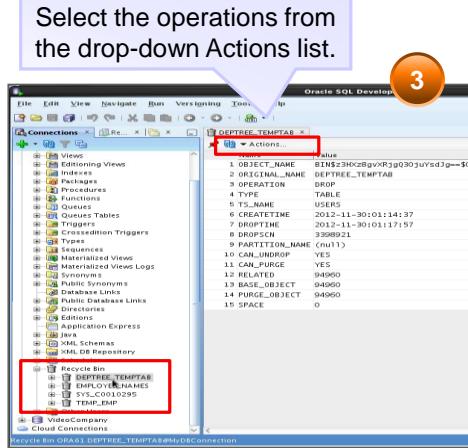
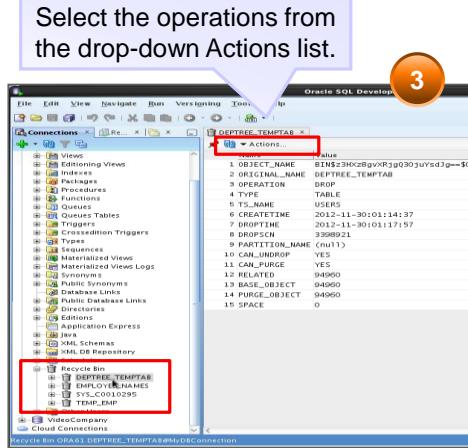
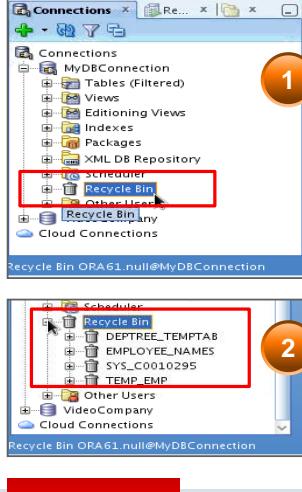
To insert a snippet into your code in a SQL Worksheet or in a PL/SQL function or procedure, drag the snippet from the Snippets window to the desired place in your code. Then you can edit the syntax so that the SQL function is valid in the current context. To see a brief description of a SQL function in a tool tip, place the cursor over the function name.

The example in the slide shows that `CONCAT (char1, char2)` is dragged from the Character Functions group in the Snippets window. Then the `CONCAT` function syntax is edited and the rest of the statement is added as in the following:

```
SELECT CONCAT(first_name, last_name) FROM employees;
```

Using the Recycle Bin

The recycle bin holds objects that have been dropped.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

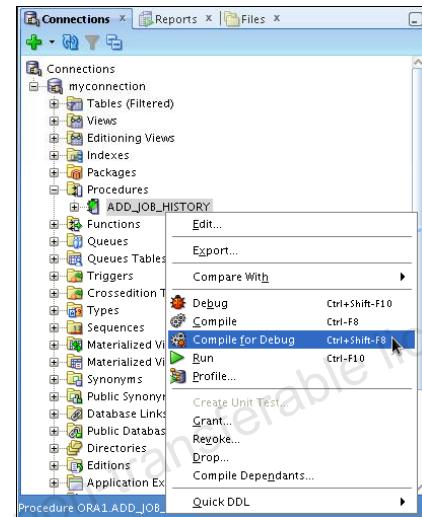
The recycle bin is a data dictionary table containing information about dropped objects. Dropped tables and any associated objects such as indexes, constraints, nested tables are not removed and still occupy space. They continue to count against user space quotas until specifically purged from the recycle bin or the unlikely situation where they must be purged by the database because of tablespace space constraints.

To use the recycle bin, perform the following:

1. In the Connections navigator, select (or navigate to) Recycle Bin.
2. Expand Recycle Bin and click the object name. The object details are displayed in the SQL Worksheet area.
3. Click the Actions drop-down list and select the operation you want to perform on the object.

Debugging Procedures and Functions

- Use SQL Developer to debug PL/SQL functions and procedures.
- Use the Compile for Debug option to perform a PL/SQL compilation so that the procedure can be debugged.
- Use the Debug menu options to set breakpoints, and to perform step-into and step-over tasks.



ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

In SQL Developer, you can debug PL/SQL procedures and functions. Using the Debug menu options, you can perform the following debugging tasks:

- **Find Execution Point:** Goes to the next execution point
- **Resume:** Continues execution
- **Step Over:** Bypasses the next method and goes to the next statement after the method
- **Step Into:** Goes to the first statement in the next method
- **Step Out:** Leaves the current method and goes to the next statement
- **Step to End of Method:** Goes to the last statement of the current method
- **Pause:** Halts execution, but does not exit, thus allowing you to resume execution
- **Terminate:** Halts and exits the execution. You cannot resume execution from this point; instead, to start running or debugging from the beginning of the function or procedure, click the Run or Debug icon on the Source tab toolbar.
- **Garbage Collection:** Removes invalid objects from the cache in favor of more frequently accessed and more valid objects

These options are also available as icons on the Debugging tab of the output window.

Database Reporting

SQL Developer provides several predefined reports about the database and its objects.

Owner	Name	Type	Referenced_Owner	Referenced_Name	Referenced_Type
APEX_040100	APEX	PROCEDURE	APEX_040100	WW_FLOW	PACKAGE
APEX_040100	APEX	PROCEDURE	APEX_040100	WW_FLOW_ISC	PACKAGE
APEX_040100	APEX	PROCEDURE	APEX_040100	WW_FLOW_SECURITY	PACKAGE
APEX_040100	APEX	PROCEDURE	SYS	STANDARD	PACKAGE
APEX_040100	APEX	PROCEDURE	SYS	SYS_STUB_FOR_PURITY_ANALYSIS	PACKAGE
APEX_040100	APEXWS	PACKAGE	SYS	STANDARD	PACKAGE
APEX_040100	APEXADMIN	PROCEDURE	APEX_040100	F	PROCEDURE
APEX_040100	APEXADMIN	PROCEDURE	SYS	STANDARD	PACKAGE
APEX_040100	APEXADMIN	PROCEDURE	SYS	SYS_STUB_FOR_PURITY_ANALYSIS	PACKAGE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	NV	FUNCTION
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WW_FLOWWS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WW_FLOW_APPLICATION_GROUPS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WW_FLOW_AUTHENTIFICATIONS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WW_FLOW_COMPANIES	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WW_FLOW_COMPANY_SCHEMAS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WW_FLOW_COMPUTATIONS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WW_FLOW_ICON_BAR	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WW_FLOW_INSTALL_SCRIPTS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WW_FLOW_ITEMS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WW_FLOW_LANGUAGE_MAP	TABLE

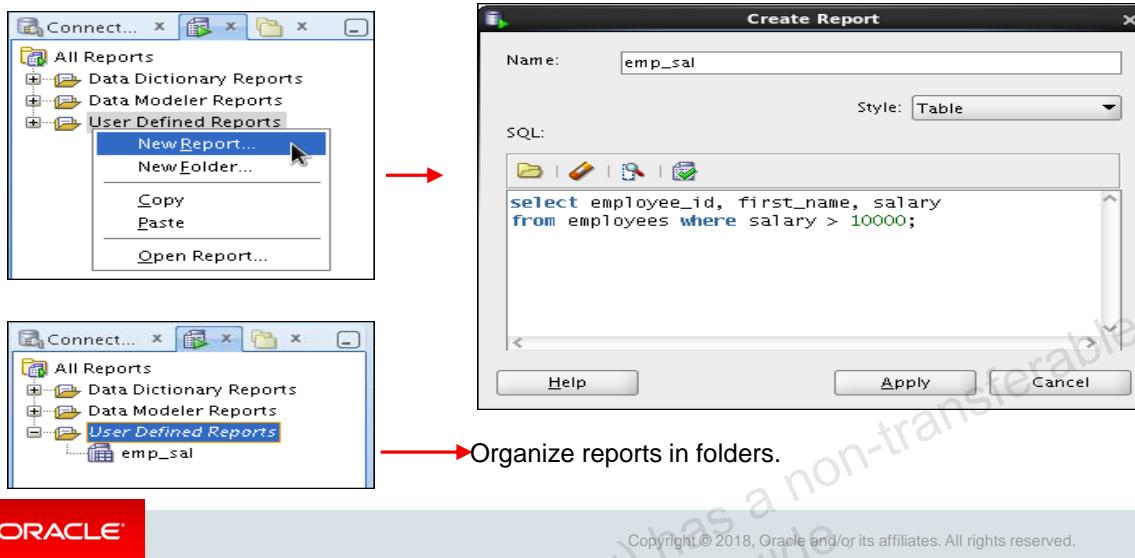
SQL Developer provides many reports about the database and its objects. These reports can be grouped into the following categories:

- About Your Database reports
- Database Administration reports
- Table reports
- PL/SQL reports
- Security reports
- XML reports
- Jobs reports
- Streams reports
- All Objects reports
- Data Dictionary reports
- User-Defined reports

To display reports, click the Reports tab on the left of the window. Individual reports are displayed in tabbed panes on the right of the window; for each report, you can select (using a drop-down list) the database connection for which to display the report. For reports about objects, the objects shown are only those visible to the database user associated with the selected database connection, and the rows are usually ordered by Owner. You can also create your own user-defined reports.

Creating a User-Defined Report

Create and save user-defined reports for repeated use.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

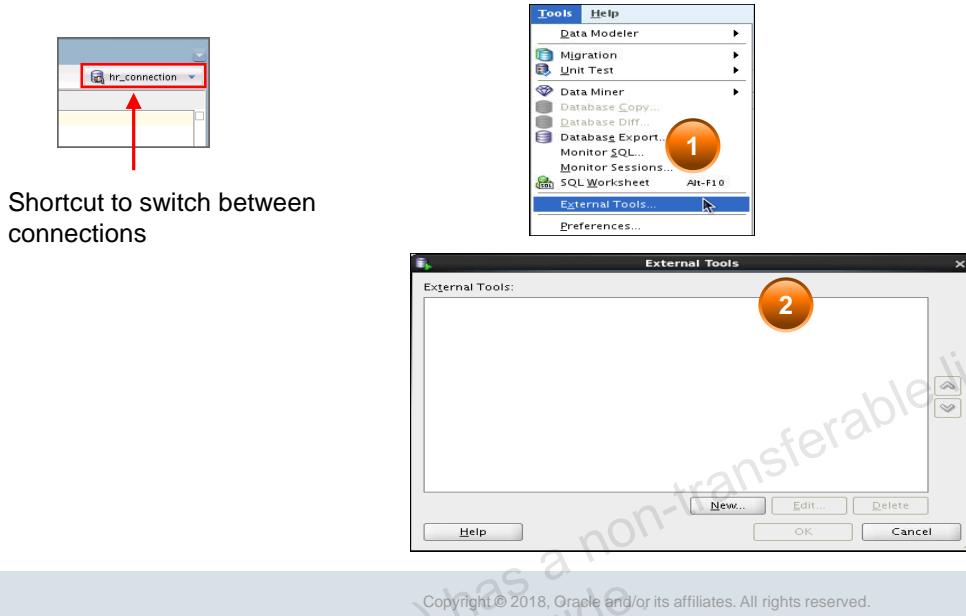
User-defined reports are reports created by SQL Developer users. To create a user-defined report, perform the following:

1. Right-click the User Defined Reports node under Reports and select Add Report.
2. In the Create Report dialog box, specify the report name and the SQL query to retrieve information for the report. Then click Apply.

In the example in the slide, the report name is specified as `emp_sal`. An optional description is provided indicating that the report contains details of employees with `salary >= 10000`. The complete SQL statement for retrieving the information to be displayed in the user-defined report is specified in the SQL box. You can also include an optional tool tip to be displayed when the cursor stays briefly over the report name in the Reports navigator display.

You can organize user-defined reports in folders and you can create a hierarchy of folders and subfolders. To create a folder for user-defined reports, right-click the User Defined Reports node or any folder name under that node and select Add Folder. Information about user-defined reports, including any folders for these reports, is stored in a file named `UserReports.xml` in the directory for user-specific information.

Search Engines and External Tools



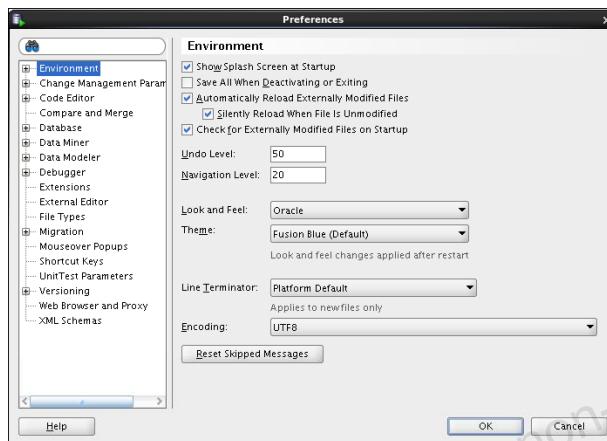
To enhance the productivity of developers, you have shortcut icons to some of the frequently used tools such as Notepad, Microsoft Word, and Dreamweaver available to you.

You can add external tools to the existing list or even delete shortcuts to the tools that you do not use frequently. To do so, perform the following:

1. From the Tools menu, select External Tools.
2. In the External Tools dialog box, select New to add new tools. Select Delete to remove any tool from the list.

Setting Preferences

- Customize the SQL Developer interface and environment.
- In the Tools menu, select Preferences.



ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

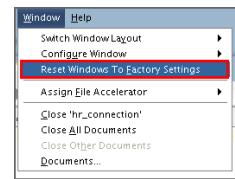
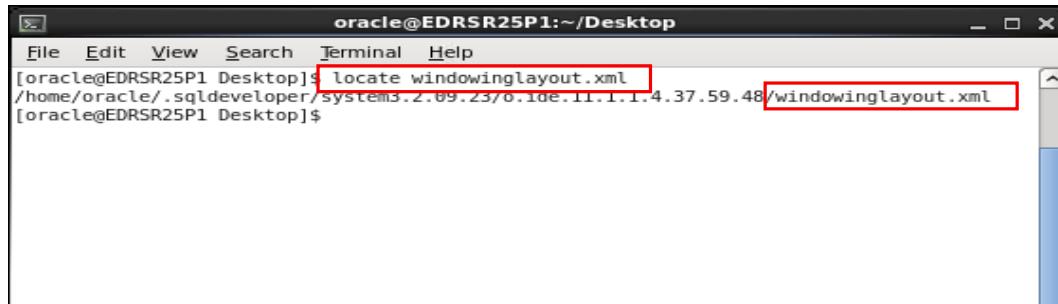
You can customize many aspects of the SQL Developer interface and environment by modifying SQL Developer preferences according to your needs. To modify SQL Developer preferences, select Tools and then Preferences.

The preferences are grouped into the following categories:

- Environment
- Change Management parameter
- Code Editors
- Compare and Merge
- Database
- Data Miner
- Data Modeler
- Debugger
- Extensions
- External Editor
- File Types
- Migration

- Mouseover Popups
- Shortcut Keys
- Unit Test Parameters
- Versioning
- Web Browser and Proxy
- XML Schemas

Resetting the SQL Developer Layout



ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

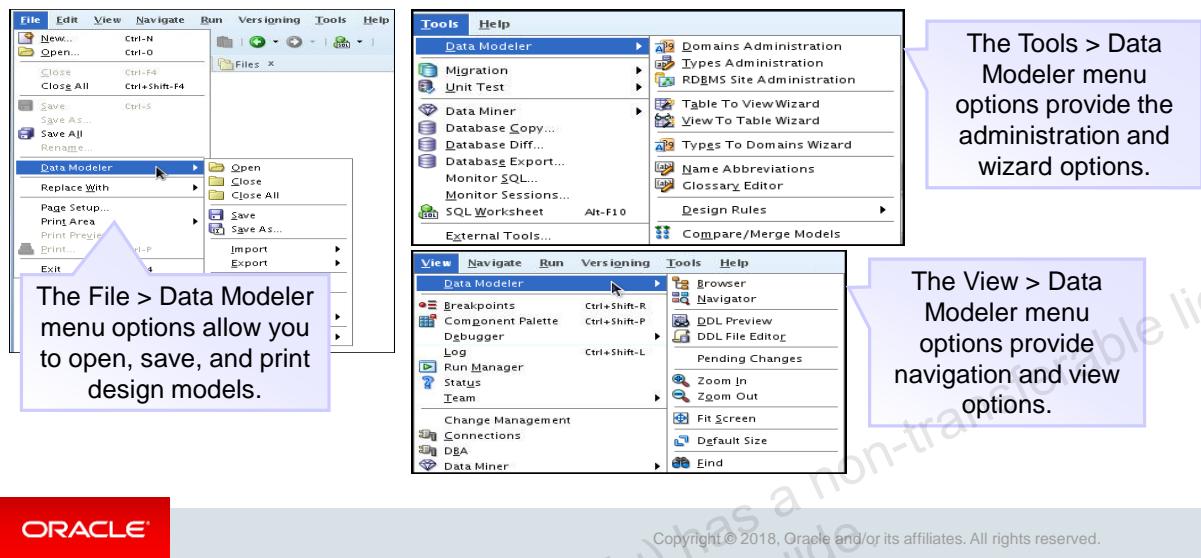
While working with SQL Developer, if the Connections Navigator disappears or if you cannot dock the Log window in its original place, perform the following steps to fix the problem:

1. Exit SQL Developer.
2. Open a terminal window and use the `locate` command to find the location of `windowinglayout.xml`.
3. Go to the directory that has `windowinglayout.xml` and delete it.
4. Restart SQL Developer.

Or, click Window > Reset Windows to Factory Settings.

Data Modeler in SQL Developer

SQL Developer includes an integrated version of SQL Developer Data Modeler.



Using the integrated version of the SQL Developer Data Modeler, you can:

- Create, open, import, and save a database design
- Create, modify, and delete Data Modeler objects

To display Data Modeler in a pane, click Tools and then Data Modeler. The Data Modeler menu under Tools includes additional commands that enable you to, for example, specify design rules and preferences.

Summary

In this appendix, you should have learned how to use SQL Developer to:

- Browse, create, and edit database objects
- Execute SQL statements and scripts in SQL Worksheet
- Create and save custom reports
- Browse the data modeling options in SQL Developer



ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

SQL Developer is a free graphical tool to simplify database development tasks. Using SQL Developer, you can browse, create, and edit database objects. You can use SQL Worksheet to run SQL statements and scripts. SQL Developer enables you to create and save your own special set of reports for repeated use.

B

SQL Tuning Advisor

ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe statement profiling
- Use SQL Tuning Advisor



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Tuning SQL Statements Automatically

- Tuning SQL statements automatically eases the entire SQL tuning process and replaces manual SQL tuning.
- The optimizer has two modes:
 - Normal mode
 - Tuning mode or Automatic Tuning Optimizer (ATO)
- You use SQL Tuning Advisor to access tuning mode.
- You should use tuning mode only for high-load SQL statements.



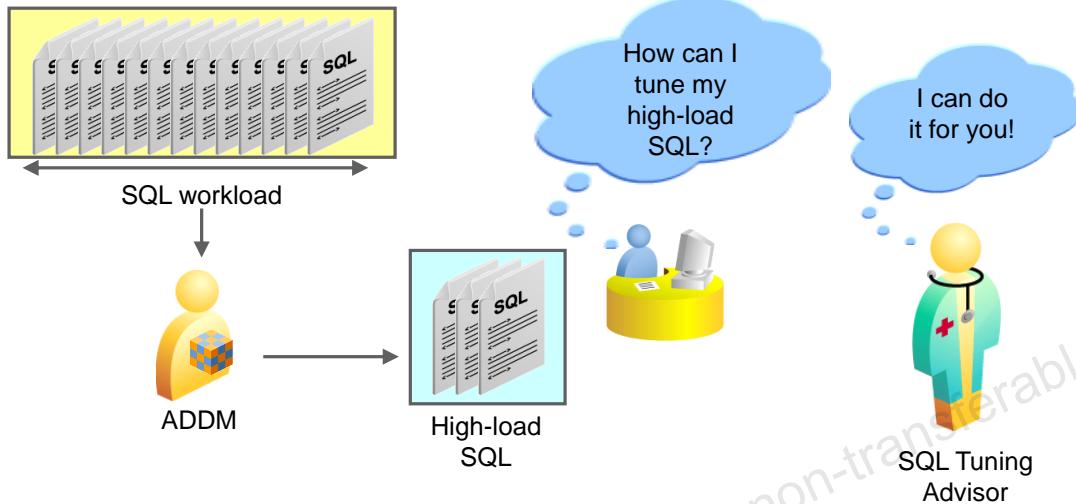
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The query optimizer can automate the entire SQL tuning process. This automatic process replaces manual SQL tuning, which is a complex, repetitive, and time-consuming function. SQL Tuning Advisor exposes the features of SQL tuning to the user. The enhanced query optimizer has two modes:

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

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

Application Tuning Challenges



ORACLE®

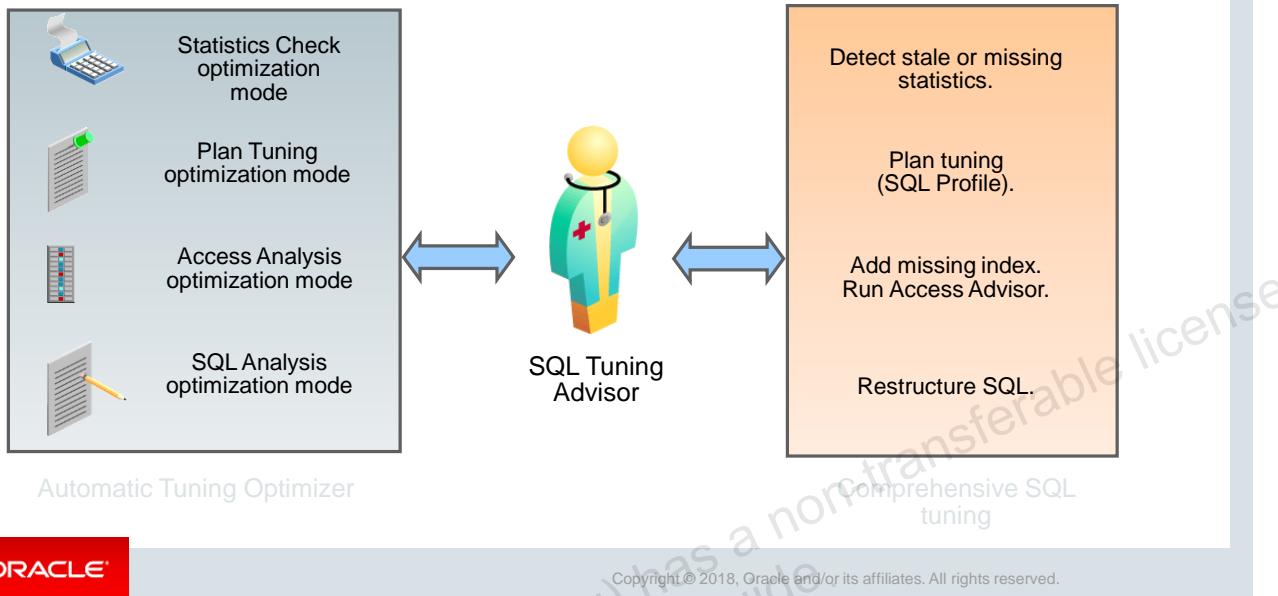
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The process of identifying high-load SQL statements and tuning them is very challenging even for an expert. SQL tuning is not only one of the most critical aspects of managing the performance of a database server, but also one of the most difficult tasks to accomplish. Starting with Oracle Database 10g, the task of identifying high-load SQL statements has been automated by Automatic Database Diagnostic Monitor (ADDM). Even though the number of high-load SQL statements that are identified by ADDM may represent a very small percentage of the total SQL workload, the task of tuning them is still highly complex and requires a high level of expertise.

Also, SQL tuning activity is a continuous task because the SQL workload can change relatively often when new application modules are deployed.

SQL Tuning Advisor, introduced with Oracle Database 10g, is designed to replace the manual tuning of SQL statements. SQL statements that consume high resources (such as CPU, I/O, and temporary space) are good candidates for SQL Tuning Advisor. The advisor receives one or more SQL statements as input and then provides advice on how to optimize the execution plan, a rationale for the advice, estimated performance benefits, and the actual command to implement the advice. You accept the advice, thereby tuning the SQL statements. With the introduction of SQL Tuning Advisor, you can now let the Oracle optimizer tune the SQL code for you.

SQL Tuning Advisor: Overview



SQL Tuning Advisor is primarily the driver of the tuning process. It calls ATO to perform the following four specific types of analyses:

- **Statistics Analysis:** ATO checks each query object for missing or stale statistics and makes a recommendation to gather relevant statistics. It also collects auxiliary information to supply missing statistics or correct stale statistics in case recommendations are not implemented.
- **SQL Profiling:** ATO verifies its own estimates and collects auxiliary information to remove estimation errors. It also collects auxiliary information in the form of customized optimizer settings, such as *first rows* and *all rows*, based on the past execution history of the SQL statement. It builds a SQL Profile by using the auxiliary information and makes a recommendation to create it. When a SQL Profile is created, the profile enables the query optimizer, under normal mode, to generate a well-tuned plan.
- **Access Path Analysis:** ATO explores whether a new index can be used to significantly improve access to each table in the query and, when appropriate, makes recommendations to create such indexes.
- **SQL Structure Analysis:** Here, ATO tries to identify SQL statements that lend themselves to bad plans, and makes relevant suggestions to restructure them. The suggested restructuring can be syntactic as well as semantic changes to the SQL code.

Stale or Missing Object Statistics

- Object statistics are key inputs to the optimizer.
- ATO verifies object statistics for each query object.
- ATO uses dynamic sampling and generates:
 - Auxiliary object statistics to compensate for missing or stale object statistics
 - Recommendations to gather object statistics where appropriate

```
EXEC DBMS_STATS.GATHER_TABLE_STATS(
    ownname=>'SH', tabname=>'CUSTOMERS',
    estimate_percent=>DBMS_STATS.AUTO_SAMPLE_SIZE);
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The query optimizer relies on object statistics to generate execution plans. If these statistics are stale or missing, the optimizer does not have the necessary information and can generate suboptimal execution plans.

ATO checks each query object for missing or stale statistics and produces two types of outputs:

- Auxiliary information in the form of statistics for objects with no statistics, and statistic adjustment factor for objects with stale statistics
- Recommendations to gather relevant statistics for objects with stale or no statistics

For optimal results, you gather statistics when recommended and then rerun ATO. However, you may be hesitant to accept this recommendation immediately because of the impact it could have on other queries in the system.

SQL Statement Profiling

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

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



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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

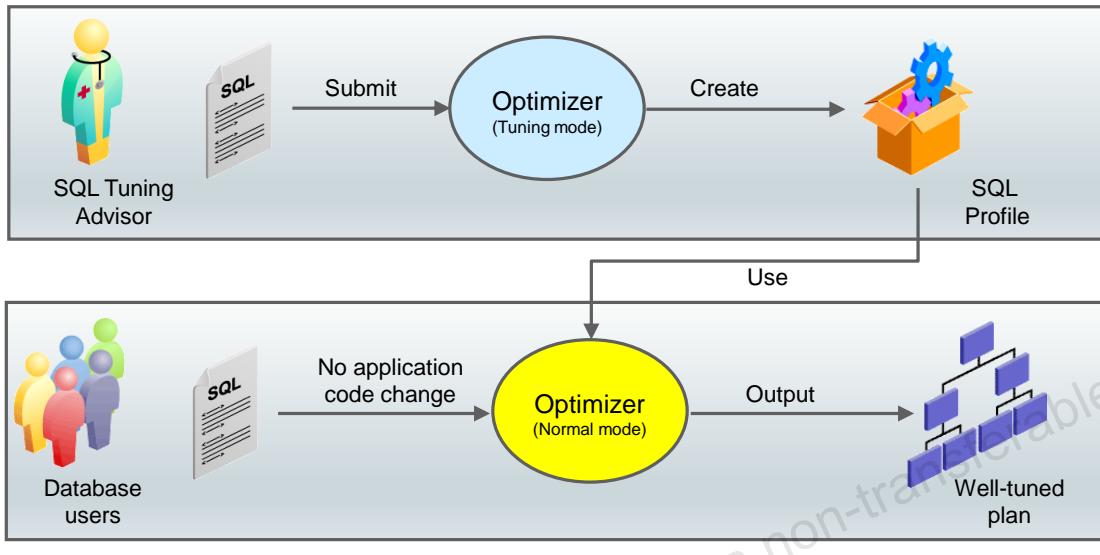
During SQL Profiling, ATO performs verification steps to validate its own estimates. The validation consists of taking a sample of data and applying appropriate predicates to the sample. The new estimate is compared to the regular estimate, and if the difference is large enough, a correction factor is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

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

ATO builds a SQL Profile if it has generated auxiliary information either during Statistics Analysis or during SQL Profiling. When a SQL Profile is built, it generates a user recommendation to create a SQL Profile.

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

Plan Tuning Flow and SQL Profile Creation

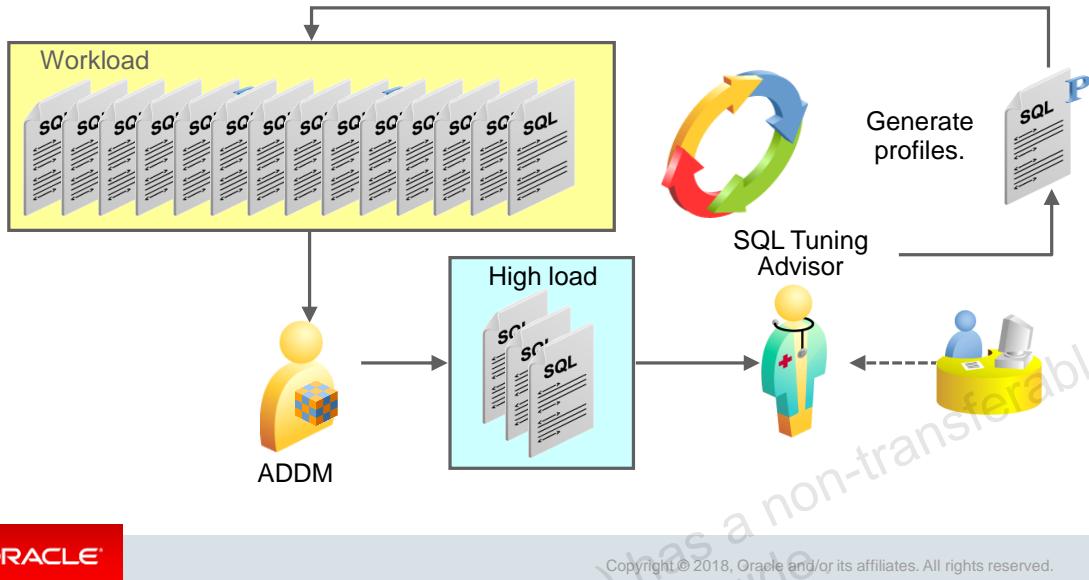


A SQL Profile is a collection of auxiliary information that is built during automatic tuning of a SQL statement. Thus, a SQL Profile is to a SQL statement what statistics are to a table or index. After it is created, a SQL Profile is used in conjunction with the existing statistics by the query optimizer, in normal mode, to produce a well-tuned plan for the corresponding SQL statement. A SQL Profile is stored persistently in the data dictionary. However, the SQL profile information is not exposed through regular dictionary views. After creation of a SQL Profile, every time the corresponding SQL statement is compiled in normal mode, the query optimizer uses the SQL Profile to produce a well-tuned plan.

The slide shows the process flow of the creation and use of a SQL Profile. The process consists of two phases: system SQL tuning phase and regular optimization phase.

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

SQL Tuning Loop

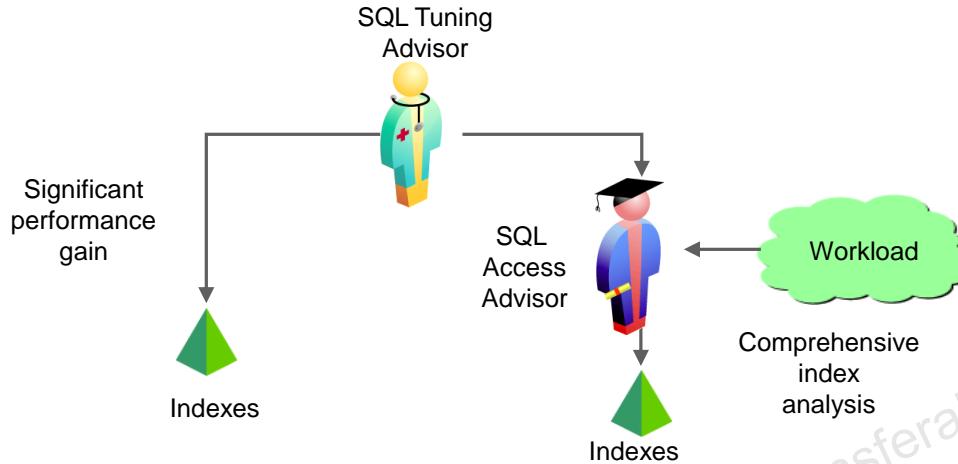


The auxiliary information contained in a SQL Profile is stored in such a way that it stays relevant after database changes, such as addition or removal of indexes, growth in the size of tables, and periodic collection of database statistics. Therefore, when a profile is created, the corresponding plan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

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

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

Access Path Analysis



```
CREATE INDEX JFV.IDX$_00002 ON JFV.TEST("C");
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

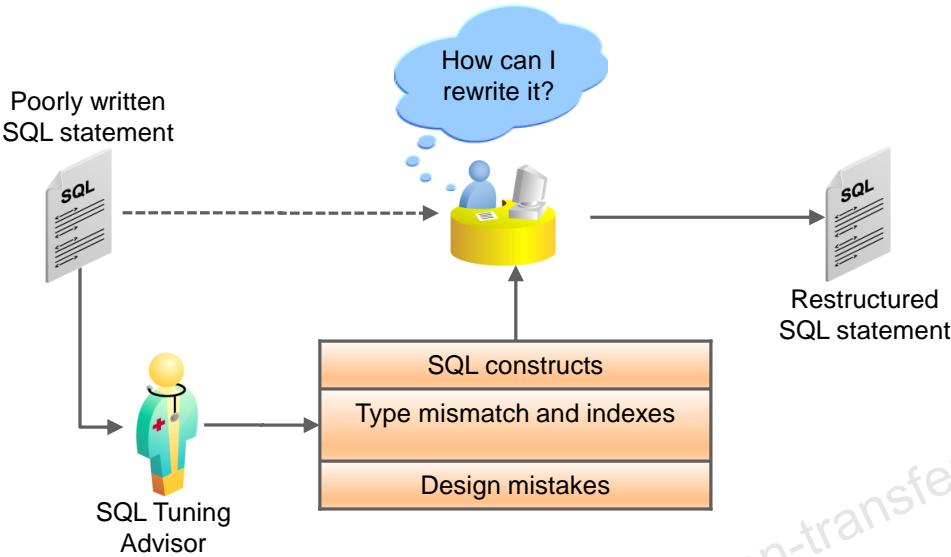
ATO also provides advice on indexes. Effective indexing is a well-known tuning technique that can significantly improve the performance of SQL statements by reducing the need for full table scans. Any index recommendations generated by ATO are specific to the SQL statement being tuned. Therefore, it provides a quick solution to the performance problem associated with a single SQL statement.

Because ATO does not perform an analysis of how its index recommendations affect the entire SQL workload, it recommends running the Access Advisor on the SQL statement along with a representative SQL workload. The Access Advisor collects advice given on each statement of a SQL workload and consolidates it into global advice for the entire SQL workload.

Access Path Analysis can make the following recommendations:

- Create new indexes if they provide significantly superior performance.
- Run SQL Access Advisor to perform a comprehensive index analysis based on application workload.

SQL Structure Analysis



ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

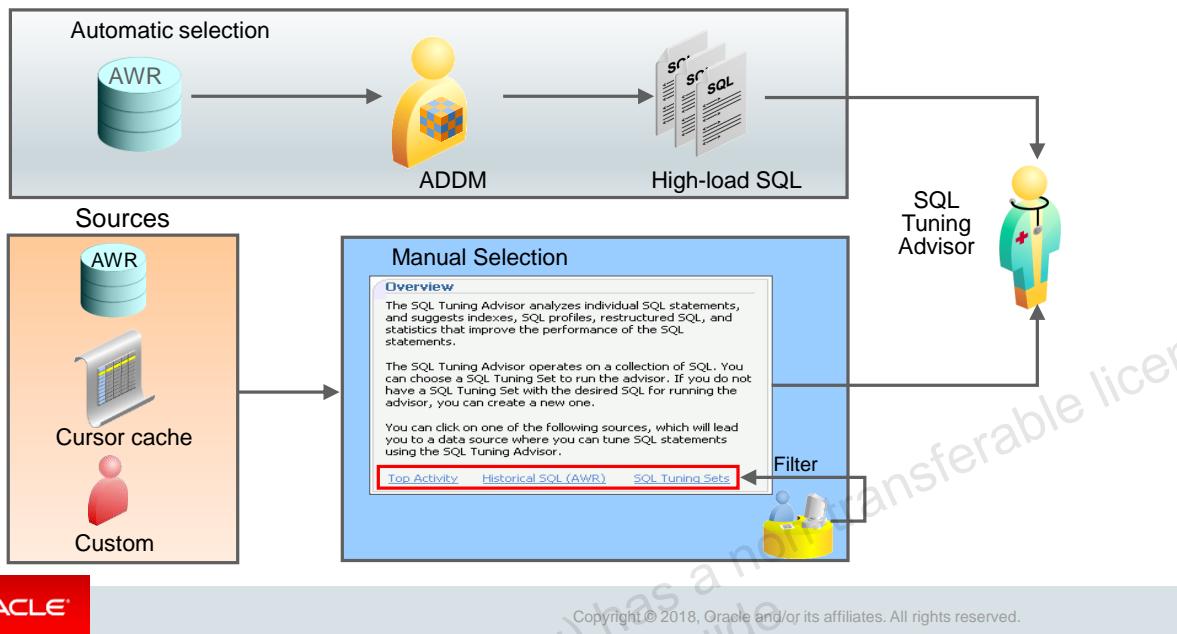
The goal of the SQL Structure Analysis is to help you identify poorly written SQL statements as well as to advise you on how to restructure them.

Certain syntax variations are known to have a negative impact on performance. In this mode, ATO evaluates statements against a set of rules, identifying less efficient coding techniques, and providing recommendations for an alternative statement where possible. The recommendation may be very similar, but not precisely equivalent to the original query. For example, the NOT EXISTS and NOT IN constructors are similar, but not exactly the same. Therefore, you have to decide whether the recommendation is valid. For this reason, ATO does not automatically rewrite the query, but gives advice instead.

The following categories of problems are detected by SQL Structure Analysis:

- Use of SQL constructors such as NOT IN instead of NOT EXISTS, or UNION instead of UNION ALL
- Use of predicates involving indexed columns with data-type mismatch that prevents use of the index
- Design mistakes (such as Cartesian products)

SQL Tuning Advisor: Usage Model



SQL Tuning Advisor takes one or more SQL statements as input. The input can come from different sources:

- High-load SQL statements identified by ADDM
- SQL statements that are currently in cursor cache
- SQL statements from Automatic Workload Repository (AWR): A user can select any set of SQL statements captured by AWR. This can be done by using snapshots or baselines.
- Custom workload: A user can create a custom workload consisting of statements that are of interest to the user. These may be statements that are not in cursor cache and are not high-load SQL statements to be captured by ADDM or AWR. For such statements, a user can create a custom workload and tune it by using the advisor.

SQL statements from cursor cache, AWR, and custom workload can be filtered and ranked before they are input to SQL Tuning Advisor.

For a multistatement input, an object called SQL Tuning Set (STS) is provided. An STS stores multiple SQL statements along with their execution information:

- **Execution context:** Parsing schema name and bind values
- **Execution statistics:** Average elapsed time and execution count

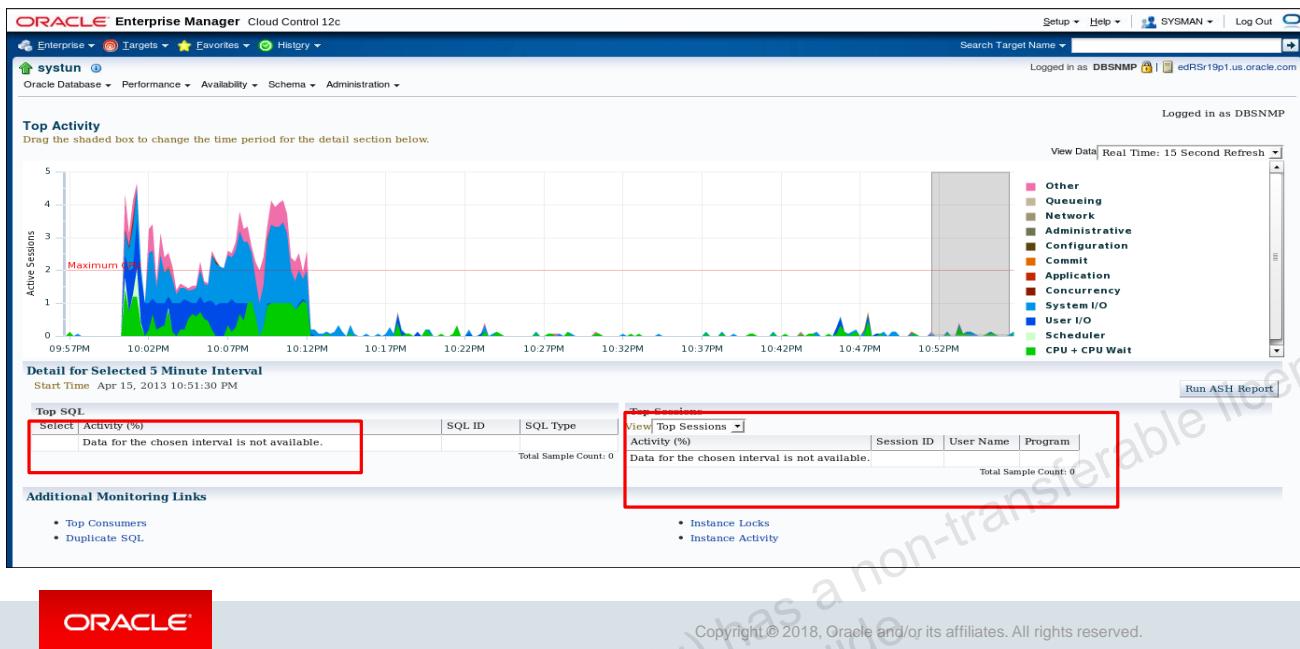
Note: Another STS can be a possible source for STS creation.

Cloud Control and SQL Tuning Advisor

The screenshot shows two pages of the Oracle Enterprise Manager Cloud Control 12c interface. The top page is the Performance Home page for the 'system' database, with the 'SQL' link highlighted. A red box highlights the 'SQL Tuning Advisor' link under the 'SQL' menu. The bottom page is the 'Schedule SQL Tuning Advisor' page. It shows fields for 'Name' (SQL_TUNING_1366066099939), 'SQL Tuning Set' (selected), and 'Scope' (Total Time Limit (minutes) set to 30, with 'Comprehensive' selected). A red box highlights the 'Run Analysis' button at the bottom of the page. The Oracle logo is visible at the bottom left, and the copyright notice 'Copyright © 2018, Oracle and/or its affiliates. All rights reserved.' is at the bottom right.

Access the Database Home page to access the SQL Tuning Advisor. From the Performance menu, click SQL, then SQL Tuning Advisor. This takes you to the Schedule SQL Tuning Advisor page. On this page, you find links to various other pages. You click the Top Activity link to open the Top Activity page.

Running SQL Tuning Advisor: Example



You can use Cloud Control to identify the high-load or top SQL statements. There are several locations in Cloud Control from where SQL Tuning Advisor can be launched with the identified SQL statements or with an STS:

- **Tuning ADDM-identified SQL statements:** The ADDM Finding Details page shows high-load SQL statements identified by ADDM. Each of these high-load SQL statements is known to consume a significant proportion of one or more system resources. You can use this page to launch SQL Tuning Advisor on a selected high-load SQL statement.
- **Tuning top SQL statements:** Another SQL source is the list of top SQL statements, as shown in the slide. You can identify the list of top SQL statements by looking at their cumulative execution statistics based on a selected time window. The user can select one or more top SQL statements identified by their SQL IDs, and then click Schedule SQL Tuning Advisor.
- **Tuning a SQL Tuning Set:** It is also possible to look at various STSs created by different users. An STS could have been created from a list of top SQL statements, by selecting SQL statements from a range of snapshots created by AWR, or by selecting customized SQL statements.

Schedule SQL Tuning Advisor Page

The diagram illustrates the workflow for scheduling a SQL Tuning Advisor task. It starts with the 'Schedule SQL Tuning Advisor' page, where a user enters parameters like a task name ('SQL_TUNING_JLS') and selects a scope ('Comprehensive'). After clicking 'Submit', the user is directed to the 'Processing' page, which displays the status of the task ('EXECUTING') and provides options to cancel or interrupt it.

Schedule SQL Tuning Advisor Page:

- Name: SQL_TUNING_JLS
- Description: (empty)
- Scope:**
 - Total Time Limit (minutes): 30
 - Scope of Analysis:
 - Limited: The analysis is done without SQL Profile recommendation.
 - Comprehensive: This analysis includes SQL Profile recommendations.
 - Time Limit per Statement (minutes): 5
- Schedule:**
 - Time Zone: UTC
 - Run Immediately or Later
 - Date: Jul 21, 2010
 - Time: 2:37:00 AM

Processing: SQL Tuning Advisor Task SQL_TUNING_JLS

- SQL ID: 5mxdwvuf9j3vp
- Time Limit (seconds): 1800
- Status: EXECUTING
- Started: Jul 21, 2010 2:33:40 PM
- Elapsed Time (seconds): 0

When SQL Tuning Advisor is launched, Enterprise Manager automatically creates a tuning task, provided that the user has the appropriate ADVISOR privilege to do so. Enterprise Manager shows the tuning task with automatic defaults on the Schedule SQL Tuning Advisor page. On this page, the user can change the automatic defaults pertaining to a tuning task.

One of the important options is to select the scope of the tuning task. If you select the Limited option, SQL Tuning Advisor produces recommendations based on statistics check, access path analysis, and SQL structure analysis. No SQL Profile recommendation is generated with Limited scope. If you select the Comprehensive option, SQL Tuning Advisor performs all of the Limited scope actions, and invokes the optimizer under SQL Profiling mode to build a SQL Profile, if applicable. With the Comprehensive option, you can also specify a time limit for the tuning task, which by default is 30 minutes. Another useful option is to run the tuning task immediately or schedule it to be run at a later time.

When the task is submitted, the Processing page appears. When the task is complete, the Recommendations page appears.

Implementing Recommendations

Database Instance: orcl.example.com > Advisor Central > SQL Tuning Task:SQL_TUNING_JLS > Logged in As SYS

Recommendations for SQL ID:5mxdwuuf9j3vp

Only one recommendation should be implemented.

SQL Text

```
SELECT /*+ ORDERED USE_NL(c) FULL(c) FULL(s)*/ COUNT(*) FROM SALES S, CUSTOMERS C WHERE C.CUST_ID = S.CUST_ID AND CUST_FIRST_NAME='Dina' ORDER BY TIME_ID
```

Select Recommendation

[Original Explain Plan \(Annotated\)](#)

[Implement](#) (highlighted with a red box)

Select Type	Findings	Recommendations	Rationale	Benefit (%)	Other Statistics	New Explain Plan	Compare Explain Plans
SQL Profile	A potentially better execution plan was found for this statement.	Consider accepting the recommended SQL profile. No SQL profile currently exists for this recommendation.		99.74			

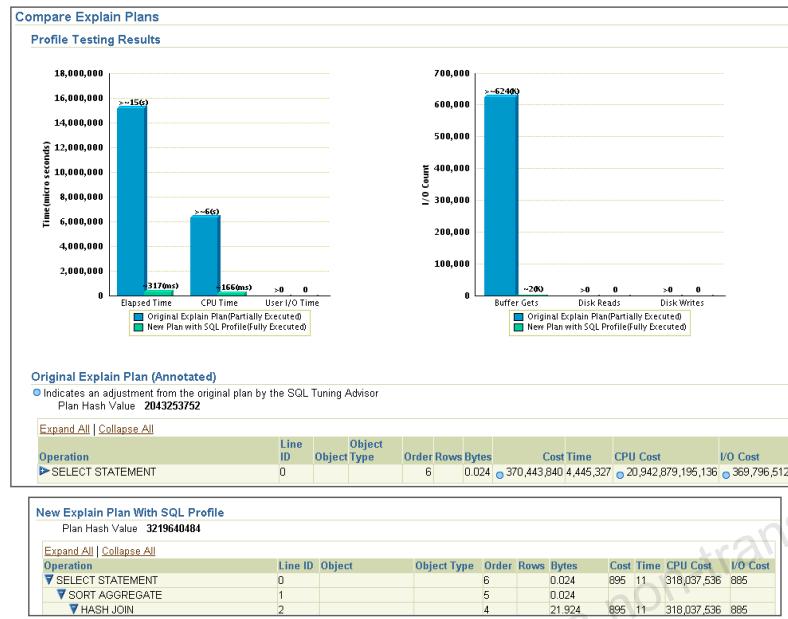
Compare Explain Plans

Profile Testing Results

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

On the Recommendations page, you can view the various recommendations. For each recommendation, as shown, a SQL Profile has been created; you can implement it if you want, after you view the new plan. Click the eyeglass icon to view the Compare Explain Plan page.

Compare Explain Plan Page



ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

On the Compare Explain Plan page you can view the projected benefits of implementing the recommendation, in this case a SQL profile. You can see the benefits graphically and in a table. Notice the Cost values for the SQL statement in the original and new explain plans. If the difference is not enough or the explain is not acceptable, you can ignore or delete the recommendation.

Quiz



SQL Tuning Advisor recommends:

- a. SQL Profiles
- b. Additional Indexes
- c. Deleting Indexes
- d. Rewriting SQL Statements
- e. All of the above



ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Answer: a, b, d

SQL Tuning Advisor in comprehensive mode recommends all except deleting indexes. SQL Tuning Advisor focuses on one SQL statement at a time. An entire workload must be considered to determine if deleting an index will help performance.

Quiz



A SQL Profile forces the best execution plan even when the data in the table changes.

- a. True
- b. False



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Answer: b

Summary

In this lesson, you should have learned how to:

- Describe statement profiling
- Use SQL Tuning Advisor



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

C

Using SQL Access Advisor

ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Objectives

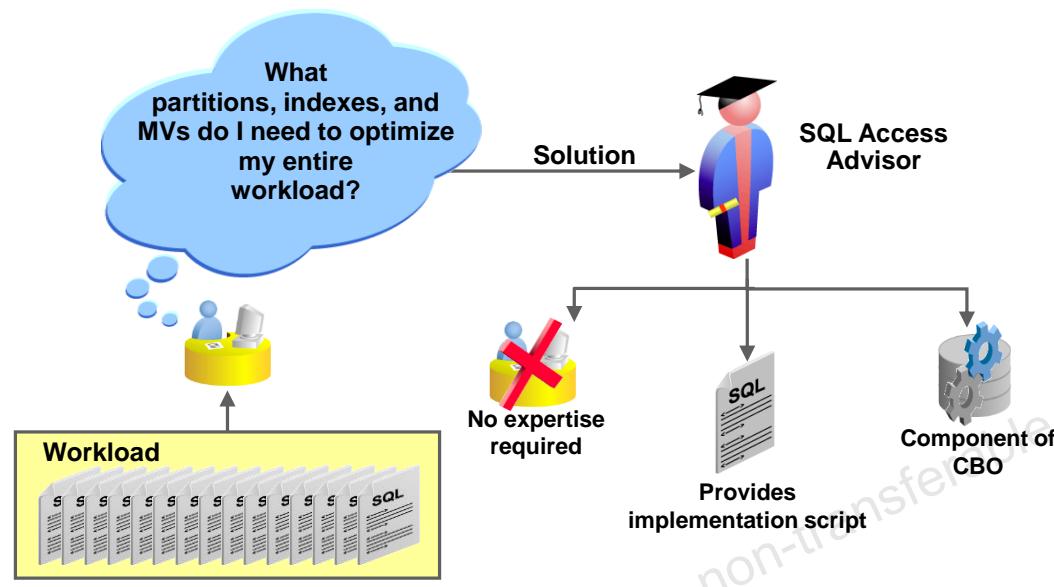
After completing this lesson, you should be able to use SQL Access Advisor.



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

SQL Access Advisor: Overview



Defining appropriate access structures to optimize SQL queries has always been a concern for developers. As a result, there have been many papers and scripts written as well as high-end tools developed to address the matter. In addition, with the development of partitioning and materialized view (MV) technology, deciding on access structures has become even more complex.

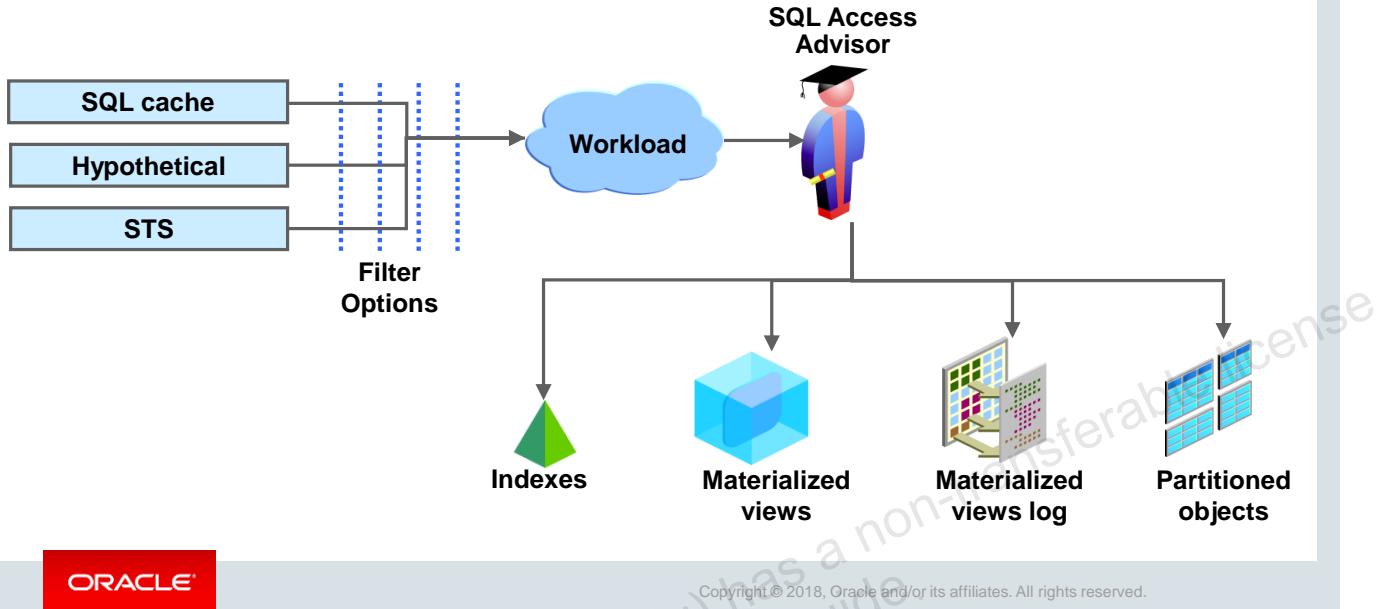
As part of the manageability improvements in Oracle Database 10g and 11g, SQL Access Advisor has been introduced to address this critical need.

SQL Access Advisor identifies and helps resolve performance problems relating to the execution of SQL statements by recommending which indexes, materialized views, materialized view logs, or partitions to create, drop, or retain. It can be run from Database Control or from the command line by using PL/SQL procedures.

SQL Access Advisor takes an actual workload as input, or it can derive a hypothetical workload from the schema. It then recommends the access structures for a faster execution path. It provides the following advantages:

- Does not require you to have expert knowledge
- Bases decision making on rules that actually reside in the cost-based optimizer (CBO)
- Is synchronized with the optimizer and Oracle Database enhancements
- Is a single advisor covering all aspects of SQL access methods
- Provides simple, user-friendly graphical user interface (GUI) wizards
- Generates scripts for implementation of recommendations

SQL Access Advisor: Usage Model



SQL Access Advisor takes as input a workload that can be derived from multiple sources:

- SQL cache, to take the current content of V\$SQL
- Hypothetical, to generate a likely workload from your dimensional model (this option is useful when your system is being initially designed)
- SQL tuning sets (STS), from the workload repository

SQL Access Advisor also provides powerful workload filters that you can use to target the tuning. For example, a user can specify that the advisor should look at only the 30 most resource-intensive statements in the workload, based on optimizer cost. For the given workload, the advisor then does the following:

- Simultaneously considers index solutions, MV solutions, partition solutions, or combinations of all three
- Considers storage for creation and maintenance costs
- Does not generate drop recommendations for partial workloads
- Optimizes MVs for maximum query rewrite usage and fast refresh
- Recommends materialized view logs for fast refresh
- Recommends partitioning for tables, indexes, and materialized views
- Combines similar indexes into a single index
- Generates recommendations that support multiple workload queries

Recommendations

Recommendation	Comprehensive	Limited
Add new (partitioned) index on table or materialized view.	YES	YES
Drop an unused index.	YES	NO
Modify an existing index by changing the index type.	YES	NO
Modify an existing index by adding columns at the end.	YES	YES
Add a new (partitioned) materialized view.	YES	YES
Drop an unused materialized view (log).	YES	NO
Add a new materialized view log.	YES	YES
Modify an existing materialized view log to add new columns or clauses.	YES	YES
Partition an existing unpartitioned table or index.	YES	YES



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

SQL Access Advisor carefully considers the overall impact of recommendations and makes recommendations by using only the known workload and supplied information. Two workload analysis approaches are available:

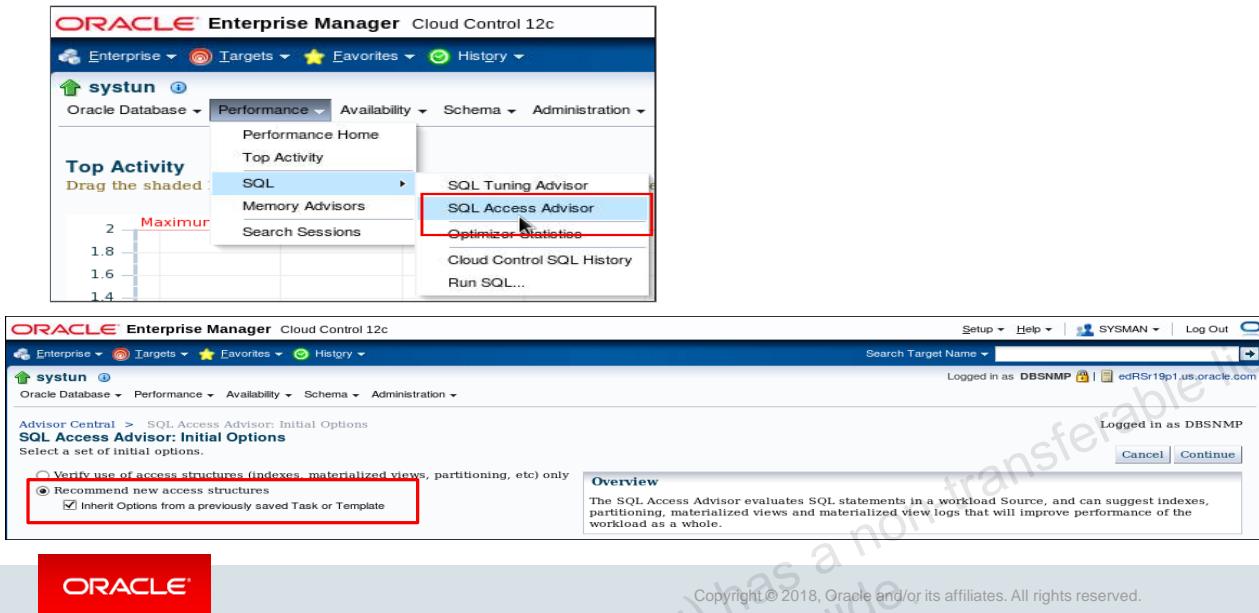
- **Comprehensive:** SQL Access Advisor addresses all aspects of tuning partitions, MVs, indexes, and MV logs. It assumes that the workload contains a complete and representative set of application SQL statements.
- **Limited:** Unlike the comprehensive workload approach, a limited workload approach assumes that the workload contains only problematic SQL statements. Thus, advice is sought for improving the performance of a portion of an application environment.

When comprehensive workload analysis is chosen, SQL Access Advisor forms a better set of global tuning adjustments, but the effect may be longer analysis time. As shown in the slide, the chosen workload approach determines the type of recommendations made by the advisor.

Note: Partition recommendations can work only on those tables that have at least 10,000 rows and on workloads that have some predicates and joins on columns of the NUMBER or DATE type.

Partitioning recommendations can be generated only on these types of columns. In addition, partitioning recommendations can be generated only for single-column interval and hash partitions. Interval partitioning recommendations can be output as range syntax, but interval is the default. Hash partitioning is done to leverage only partition-wise joins.

SQL Access Advisor Session: Initial Options

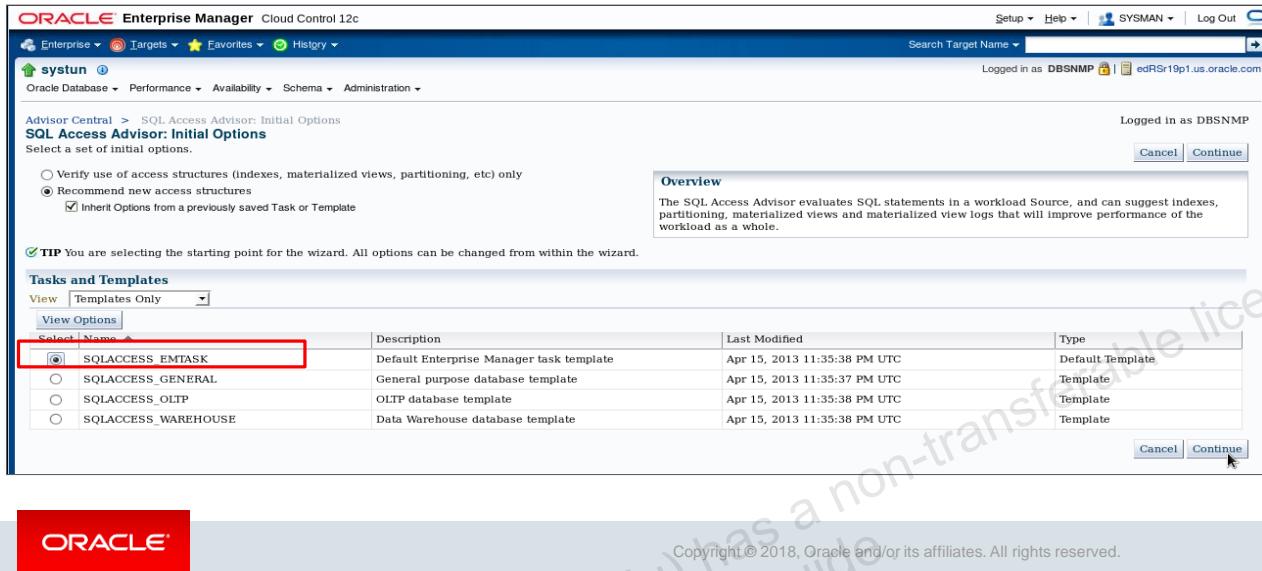


The next few slides describe a typical SQL Access Advisor session. You can access SQL Access Advisor by accessing the Database Home page in Cloud Control. From the Performance menu, select SQL, then SQL Access Advisor. SQL Access Advisor consists of several steps where you supply the SQL statements to tune and the types of access methods that you want to use.

On the SQL Access Advisor: Initial Options page, you can select a template or task from which to populate default options before starting the wizard.

Note: SQL Access Advisor may be interrupted while generating recommendations, thereby allowing the results to be reviewed. For general information about using SQL Access Advisor, see the “Overview of the SQL Access Advisor” section in the lesson titled “SQL Access Advisor” of the *Oracle Data Warehousing Guide*.

SQL Access Advisor Session: Initial Options



The screenshot shows the Oracle Enterprise Manager Cloud Control 12c interface. The top navigation bar includes 'Enterprise', 'Targets', 'Favorites', 'History', 'Setup', 'Help', 'SYSMAN', and 'Log Out'. The main menu has 'Oracle Database', 'Performance', 'Availability', 'Schema', and 'Administration' options. The current page is 'SQL Access Advisor: Initial Options'. On the left, there's a 'Tasks and Templates' section with a 'View' dropdown set to 'Templates Only'. Below it is a 'Select Name' dropdown. A table lists four templates:

	Description	Last Modified	Type
<input checked="" type="radio"/> SQLACCESS_EMTASK	Default Enterprise Manager task template	Apr 15, 2013 11:35:38 PM UTC	Default Template
<input type="radio"/> SQLACCESS_GENERAL	General purpose database template	Apr 15, 2013 11:35:37 PM UTC	Template
<input type="radio"/> SQLACCESS OLTP	OLTP database template	Apr 15, 2013 11:35:38 PM UTC	Template
<input type="radio"/> SQLACCESS_WAREHOUSE	Data Warehouse database template	Apr 15, 2013 11:35:38 PM UTC	Template

A tip message at the bottom left says: "TIP You are selecting the starting point for the wizard. All options can be changed from within the wizard." To the right, an 'Overview' box states: "The SQL Access Advisor evaluates SQL statements in a workload Source, and can suggest indexes, partitioning, materialized views and materialized view logs that will improve performance of the workload as a whole." At the bottom right of the page are 'Cancel' and 'Continue' buttons. The footer contains the Oracle logo and the text: "Copyright © 2018, Oracle and/or its affiliates. All rights reserved."

If you select the “Inherit Options from a previously saved Task or Template” option on the SQL Access Advisor: Initial Options page, you can select an existing task or an existing template to inherit the SQL Access Advisor options. By default, the `SQLACCESS_EMTASK` template is used.

You can view the various options defined by a task or a template by selecting the corresponding object and clicking View Options.

SQL Access Advisor: Workload Source

The screenshot shows the 'Workload Source' step of the SQL Access Advisor. At the top, there is a navigation bar with four tabs: 'Workload Source' (selected), 'Recommendation Options', 'Schedule', and 'Review'. Below the navigation bar, the title 'SQL Access Advisor: Workload Source' is displayed, along with the database name 'DBSNMP' and the user 'Logged In As DBSNMP'. On the right side, there are 'Cancel', 'Step 1 of 4', and 'Next' buttons. The main content area is titled 'Select the source of the workload that you want to use for the analysis. The best workload is one that fully represents all the SQL statements that access the underlying tables.' It contains three options:

- Current and Recent SQL Activity: SQL will be selected from the cache.
- Use an existing SQL Tuning Set: SQL Tuning Set [Search icon]
- Create a Hypothetical Workload from the Following Schemas and Tables: The advisor can create a hypothetical workload if the tables contain dimension or primary/foreign key constraints. Schemas and Tables [Add] [Search icon]

Below these options, there are two tips:

- TIP Enter a schema name to specify all the tables belonging to that schema.
- TIP For workloads containing a large number of SQL statements, Oracle recommends using filtering to reduce analysis time.

At the bottom right of the content area, the 'Next' button is highlighted with a red box. The footer of the page includes the ORACLE logo and the copyright notice 'Copyright © 2018, Oracle and/or its affiliates. All rights reserved.'

You can select your workload source from three different sources:

- **Current and Recent SQL Activity:** This source corresponds to SQL statements that are still cached in your system global area (SGA).
- **Existing SQL Tuning Set:** You also have the possibility of creating and using a SQL Tuning Set that holds your statements.
- **Hypothetical Workload:** This option provides a schema that allows the advisor to search for dimension tables and produce a workload. This option is very useful to initially design your schema.

Using the Filter Options section, you can further filter your workload source. Filter options are:

- Resource Consumption (number of statements ordered by Optimizer Cost, Buffer Gets, CPU Time, Disk Reads, Elapsed Time, Executions)
- Users
- Tables
- SQL Text
- Module IDs
- Actions

SQL Access Advisor: Recommendation Options

Workload Source Recommendation Options Schedule Review

SQL Access Advisor: Recommendation Options

Database system
Logged In As DBSNMP

Access Structures to Recommend

Indexes
 Materialized Views
 Partitioning

Scope

The advisor can run in one of two modes, Limited or Comprehensive. Limited Mode is meant to return quickly after processing the statements with the highest cost, potentially ignoring statements with a cost below a certain threshold. Comprehensive Mode will perform an exhaustive analysis.

Limited
Analysis will focus on highest cost statements
 Comprehensive
Analysis will be exhaustive

Advanced Options

Cancel | Back | Step 2 of 4 | **Next**

Cancel | Back | Step 2 of 4 | **Next**

ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

On the Recommendation Options page, you can select whether to limit SQL Access Advisor to recommendations based on a single access method. You can select the type of structures to be recommended by the advisor. If none of the three possible structures are chosen, the advisor evaluates existing structures instead of trying to recommend new ones.

You can use the Advisor Mode section to run the advisor in one of the two modes. These modes affect the quality of recommendations as well as the length of time required for processing. In Comprehensive Mode, the advisor searches a large pool of candidates, resulting in recommendations of the highest quality. In Limited Mode, the advisor performs quickly, limiting the candidate recommendations by working only on the highest-cost statements.

Note: You can click Advanced Options to show or hide options that allow you to set space restrictions, tuning options, and default storage locations.

SQL Access Advisor: Schedule and Review

The screenshot shows two sequential pages of the SQL Access Advisor wizard:

SQL Access Advisor: Schedule (Step 3 of 4)

- Task Name: SQLACCESS803916
- Task Description: SQL Access Advisor
- Journaling Level: Basic
- Task Expiration (days): 30
- Total Time Limit (minutes): DBMS_ADVISOR.ADVISOR_UNLIMITED
- Schedule Type: Standard
- Time Zone: PST8PDT
- Repeating: Do Not Repeat
- Start: Immediately
- Date: Feb 2, 2007
- Time: 8:55 AM

SQL Access Advisor: Review (Step 4 of 4)

- Task Name: SQLACCESS803916
- Task Description: SQL Access Advisor
- Scheduled Start Time: Run Immediately
- Options:
 - Show All Options
 - Modified Option Value Description
 - SQL Tuning Set SH.SQLSET_TEST_500 Import Workload from SQL Repository
 - Workload SQL Tuning Set The source of SQL statements to be used to create the workload
 - Source SQL Tuning Set

A red box highlights the "Next" button on the Schedule page and the "Submit" button on the Review page. A red arrow points from the "Next" button to the "Submit" button, indicating the flow between the two steps.

You can then schedule and submit your new analysis by specifying various parameters to the scheduler. The possible options are shown in the slide.

SQL Access Advisor: Results

Results

Select	Advisory Type	Name	Description	User	Status	Start Time	Duration (seconds)	Expires In (days)
<input checked="" type="radio"/>	SQL Access Advisor	SQLACCESSB03916	SQL Access Advisor	SH	COMPLETED	Feb 2, 2007 8:53:16 AM PST	139	30

Results for Task: SQLACCESSB03916

Task Name: SQLACCESSB03916
Status: COMPLETED
Advisor Mode: COMPREHENSIVE
Scheduler Job: ADV_SQLACCESSB03916

Started: Feb 2, 2007 8:53:16 AM PST
Ended: Feb 2, 2007 8:55:35 AM PST
Running Time (seconds): 139
Time Limit (seconds): UNLIMITED

Summary **Recommendations** [SQL Statements](#) [Details](#)

Overall Workload Performance

Potential for Improvement

Workload I/O Cost

Original Cost (240510) New Cost (12599)

Query Execution Time Improvement

No Performance Improvement Potential Performance Improvement

Recommendations

Recommendations: 4258
Space Requirements (MB): 1258
User Specified Space: Unlimited

[Hide Recommendation Action Counts](#)

Index : Create 2	Drop 0	Retain 0
Materialized View : Create 4	Drop 0	Retain 0
Materialized View Log : Create 3	Drop 0	Alter 0
Partitioned : Tables 1	Indexes 0	Materialized Views 2

SQL Statements

SQL Statements: 409
Statements remaining after filters were applied
[Hide Statement Counts](#)

Insert 0
Select 499
Update 0
Delete 0
Merge 0

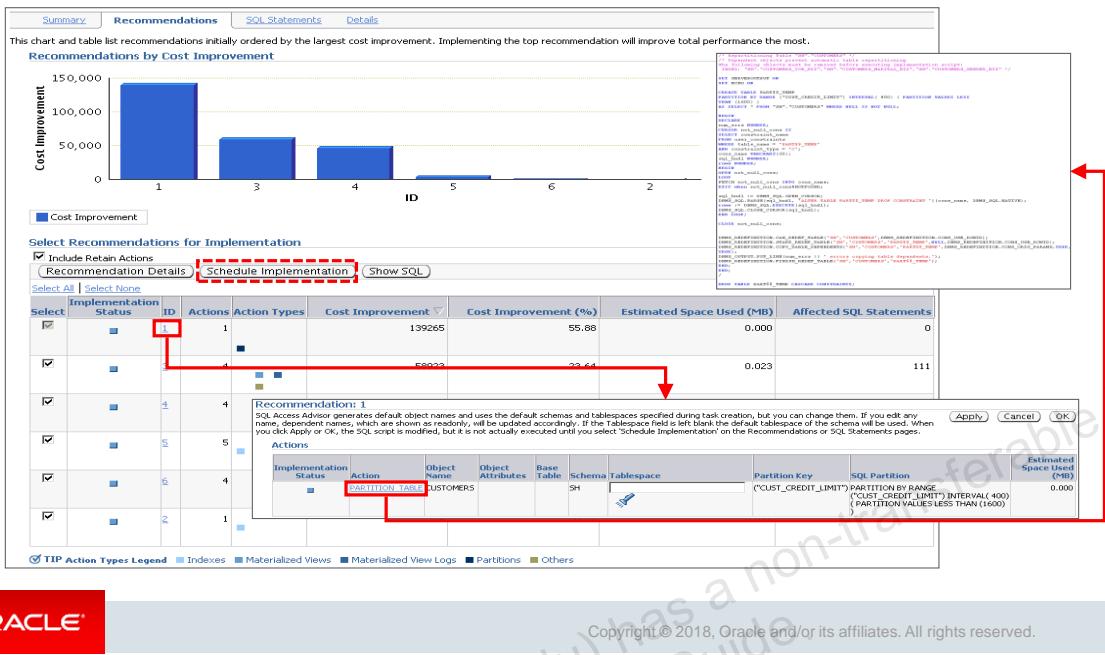
Skipped (Parsing or Privilege Errors): 499

ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

On the Advisor Central page, you can retrieve the task details for your analysis. By selecting the task name in the Results section, you can access the Results for Task Summary page, where you can see an overview of the SQL Access Advisor findings. The page shows you charts and statistics that provide overall workload performance and potential for improving query execution time for the recommendations. You can use the page to show statement counts and recommendation action counts.

SQL Access Advisor: Results and Implementation



To see other aspects of the results for the SQL Access Advisor task, click one of the three other tabs on the page: Recommendations, SQL Statements, or Details.

On the Recommendations page, you can drill down to each of the recommendations. For each of them, you see important information in the Select Recommendations for Implementation table. You can then select one or more recommendations and schedule their implementation.

If you click the ID for a particular recommendation, you are taken to the Recommendations page that displays all actions for the specified recommendation and, optionally, to modify the tablespace name of the statement. When you complete any changes, click OK to apply the changes. On the Recommendations page, you can view the full text of an action by clicking the link in the Action field for the specified action. You can view the SQL statements for all actions in the recommendation by clicking Show SQL.

The SQL Statements page (not shown here) gives you a chart and a corresponding table that lists SQL statements initially ordered by the largest cost improvement. The top SQL statement is improved the most by implementing its associated recommendation.

The Details page shows you the workload and task options that were used when the task was created. This page also gives you all journal entries that were logged during task execution.

You can also schedule implementation of the recommendations by clicking the Schedule Implementation button.

Quiz



Identify two available workload analysis methods.

- a. Comprehensive
- b. Complete
- c. Partial
- d. Limited



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Answer: a, d

Quiz



SQL Access Advisor identifies but cannot help resolve performance problems relating to the execution of SQL statements.

- a. True
- b. False



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Answer: b

Summary

In this lesson, you should have learned how to use SQL Access Advisor.



ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2019, Oracle and/or its affiliates.

GANG LIU (gangl@baylorhealth.edu) has a non-transferable license
to use this Student Guide.

D

Exploring the Oracle Database Architecture

ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- List the major architectural components of the Oracle Database server
- Explain memory structures
- Describe background processes
- Correlate logical and physical storage structures

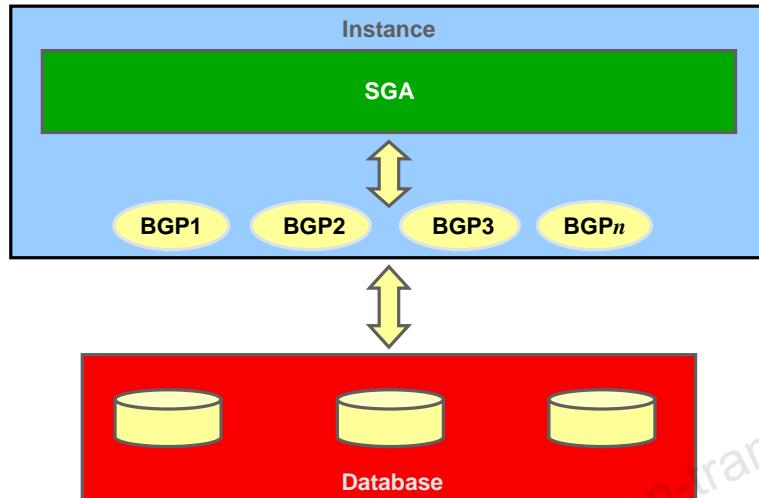


ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

This lesson provides an overview of the Oracle Database server architecture. You learn about its architectural components, memory structures, background processes, and logical and physical storage structures.

Oracle Database Server Architecture: Overview



An Oracle Database server consists of an Oracle database and one or more Oracle Database instances. An instance consists of memory structures and background processes. Every time an instance is started, a shared memory area called the system global area (SGA) is allocated and the background processes are started.

The SGA contains data and control information for one Oracle Database instance.

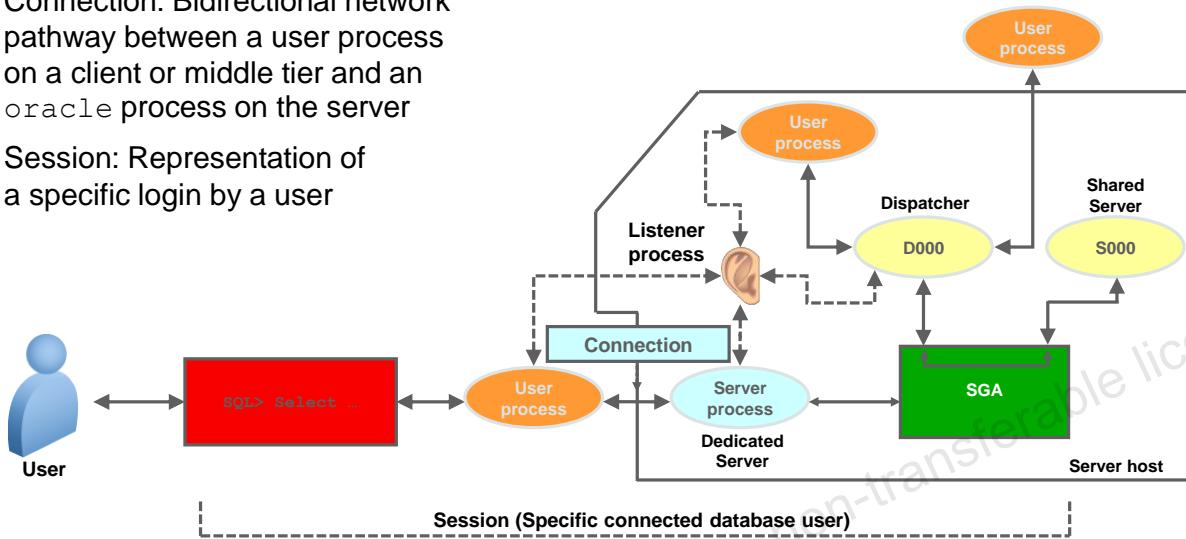
The background processes consolidate functions that would otherwise be handled by multiple Oracle Database server programs running for each user process. They may asynchronously perform input/output (I/O) and monitor other Oracle Database processes to provide increased parallelism for better performance and reliability.

The database consists of physical files and logical structures discussed later in this lesson. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting access to the logical storage structures.

Note: Oracle Real Application Clusters (Oracle RAC) comprises two or more Oracle Database instances running on multiple clustered computers that communicate with each other by means of an interconnect and access the same Oracle database.

Connecting to the Database Instance

- Connection: Bidirectional network pathway between a user process on a client or middle tier and an oracle process on the server
- Session: Representation of a specific login by a user



When users connect to an Oracle Database server, they are connected to an Oracle Database instance. The database instance services those users by allocating other memory areas in addition to the SGA and by starting other processes in addition to the Oracle Database background processes:

- User processes, which are sometimes called client or foreground processes, are created to run the software code of an application program. Most environments have separate machines for the client processes. A user process also manages communication with a corresponding server process through a program interface.
- The Oracle Database server creates server processes to handle requests from connected user processes. A server process communicates with the user process and interacts with the instance and the database to carry out requests from the associated user process.

An Oracle Database instance can be configured to vary the number of user processes for each server process. In a dedicated server configuration, a server process handles requests for a single user process.

A shared server configuration enables many user processes to share a small number of shared server processes, minimizing the number of server processes and maximizing the use of available system resources. One or more dispatcher processes are then used to queue user process requests in the SGA and dequeue shared server responses.

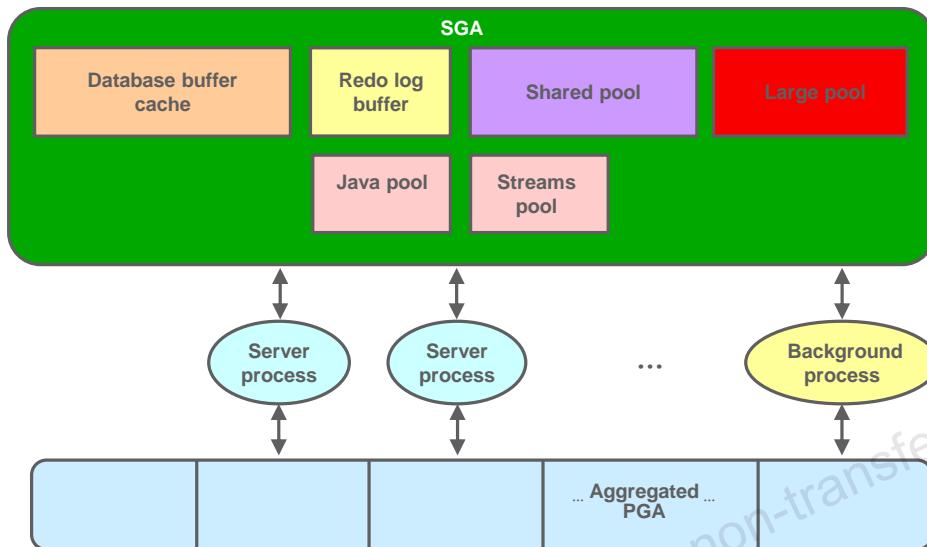
The Oracle Database server runs a listener that is responsible for handling network connections. The application connects to the listener that creates a dedicated server process or handles the connection to a dispatcher.

Connections and sessions are closely related to user processes, but are very different in meaning:

- A connection is a communication pathway between a user process and an Oracle Database instance. A communication pathway is established by using available interprocess communication mechanisms (on a computer that runs both the user process and Oracle Database) or network software (when different computers run the database application and the Oracle Database, and communicate through a network).
- A session represents the state of a current database user login to the database instance. For example, when a user starts Oracle SQL*Plus, the user must provide a valid database username and password, and then a session is established for that user. A session lasts from the time when a user connects until the user disconnects or exits the database application.

Note: Multiple sessions can be created and exist concurrently for a single Oracle Database user by using the same username. For example, a user with the username/password of HR/HR can connect to the same Oracle Database instance several times.

Oracle Database Memory Structures: Overview

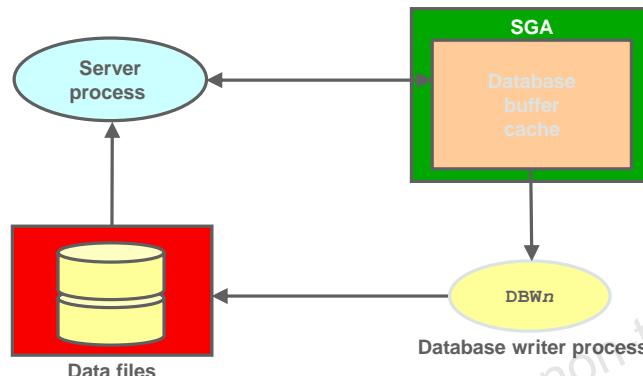


Oracle Database allocates memory structures for various purposes. For example, memory stores the program code that is run, data that is shared among users, and private data areas for each connected user. Two basic memory structures are associated with an instance:

- **System Global Area (SGA):** The SGA is shared by all server and background processes. The SGA includes the following data structures:
 - **Database buffer cache:** Caches blocks of data retrieved from the database files
 - **Redo log buffer:** Caches recovery information before writing it to the physical files
 - **Shared pool:** Caches various constructs that can be shared among sessions
 - **Large pool:** Optional area used for certain operations, such as Oracle backup and recovery operations, and I/O server processes
 - **Java pool:** Used for session-specific Java code and data in the Java Virtual Machine (JVM)
 - **Streams pool:** Used by Oracle Streams to store information about the capture and apply processes
- **Program Global Areas (PGA):** Memory regions that contain data and control information about a server or background process. A PGA is suballocated from the aggregated PGA area.

Database Buffer Cache

- Is part of the SGA
- Holds copies of data blocks that are read from data files
- Is shared by all concurrent processes



ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

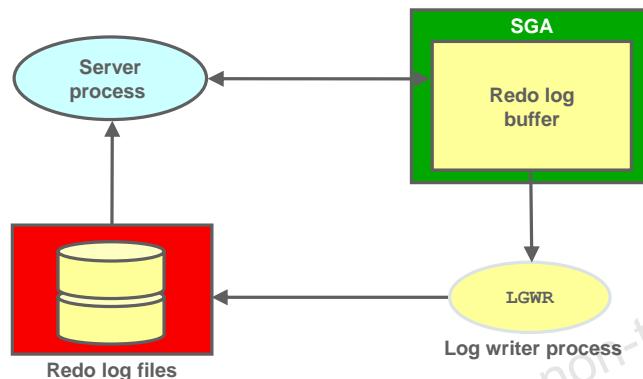
The database buffer cache is the portion of the SGA that holds copies of data blocks that are read from data files. All users concurrently connected to the instance share access to the database buffer cache.

The first time an Oracle Database server process requires a particular piece of data, it searches for the data in the database buffer cache. If the process finds the data already in the cache (a cache hit), it can read the data directly from memory. If the process cannot find the data in the cache (a cache miss), it must copy the data block from a data file on disk into a buffer in the cache before accessing the data. Accessing data through a cache hit is faster than accessing data through a cache miss.

The buffers in the cache are managed by a complex algorithm that uses a combination of least recently used (LRU) lists and touch count. The `DBWn` (database writer) processes are responsible for writing modified (dirty) buffers in the database buffer cache to disk when necessary.

Redo Log Buffer

- Is a circular buffer in the SGA (based on the number of CPUs)
- Contains redo entries that have the information to redo changes made by operations, such as DML and DDL



ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

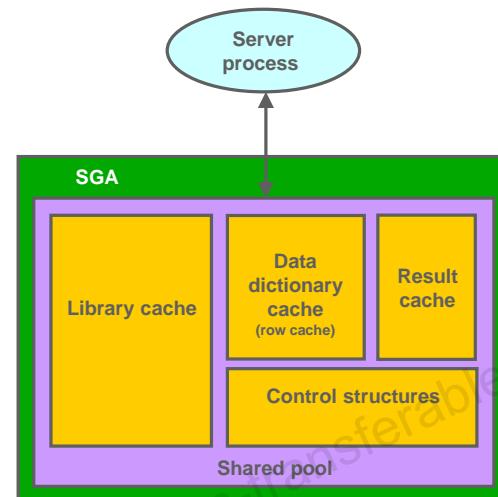
The redo log buffer is a circular buffer in the SGA that holds information about changes made to the database. This information is stored in redo entries. Redo entries contain the information necessary to reconstruct (or redo) changes that are made to the database by `INSERT`, `UPDATE`, `DELETE`, `CREATE`, `ALTER`, or `DROP` operations. Redo entries are used for database recovery, if necessary.

Redo entries are copied by Oracle Database server processes from the user's memory space to the redo log buffer in the SGA. The redo entries take up continuous, sequential space in the buffer. The `LGWR` (log writer) background process writes the redo log buffer to the active redo log file (or group of files) on disk. `LGWR` is a background process that is capable of asynchronous I/O.

Note: Depending on the number of CPUs on your system, there may be more than one redo log buffer. They are automatically allocated.

Shared Pool

- Is part of the SGA
- Contains:
 - Library cache
 - Shared parts of SQL and PL/SQL statements
 - Data dictionary cache
 - Result cache:
 - SQL queries
 - PL/SQL functions
 - Control structures
 - Locks



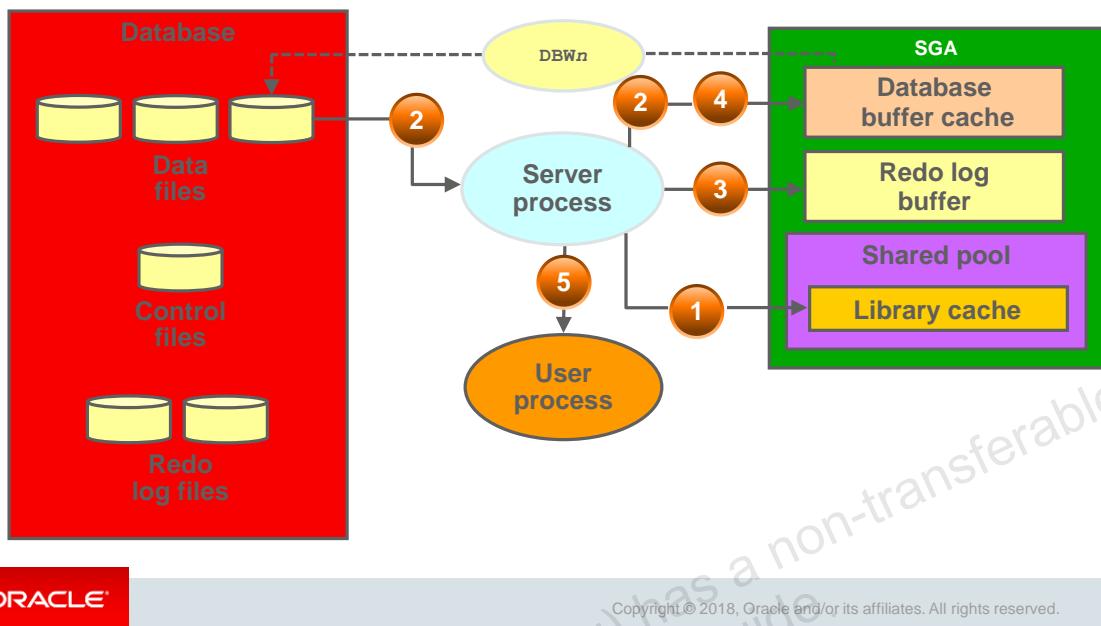
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The shared pool portion of the SGA contains the following main parts:

- The library cache includes the shareable parts of SQL statements, PL/SQL procedures, and packages. It also contains control structures, such as locks.
- The data dictionary is a collection of database tables containing reference information about the database. The data dictionary is accessed so often by Oracle Database that two special locations in memory are designated to hold dictionary data. One area is called the data dictionary cache, also known as the row cache, and the other area is called the library cache. All Oracle Database server processes share these two caches for access to data dictionary information.
- The result cache is composed of the SQL query result cache and the PL/SQL function result cache. This cache is used to store results of SQL queries or PL/SQL functions to speed up their future executions.
- Control structures are essentially lock structures.

Note: In general, any item in the shared pool remains until it is flushed according to a modified LRU algorithm.

Processing a DML Statement: Example



ORACLE

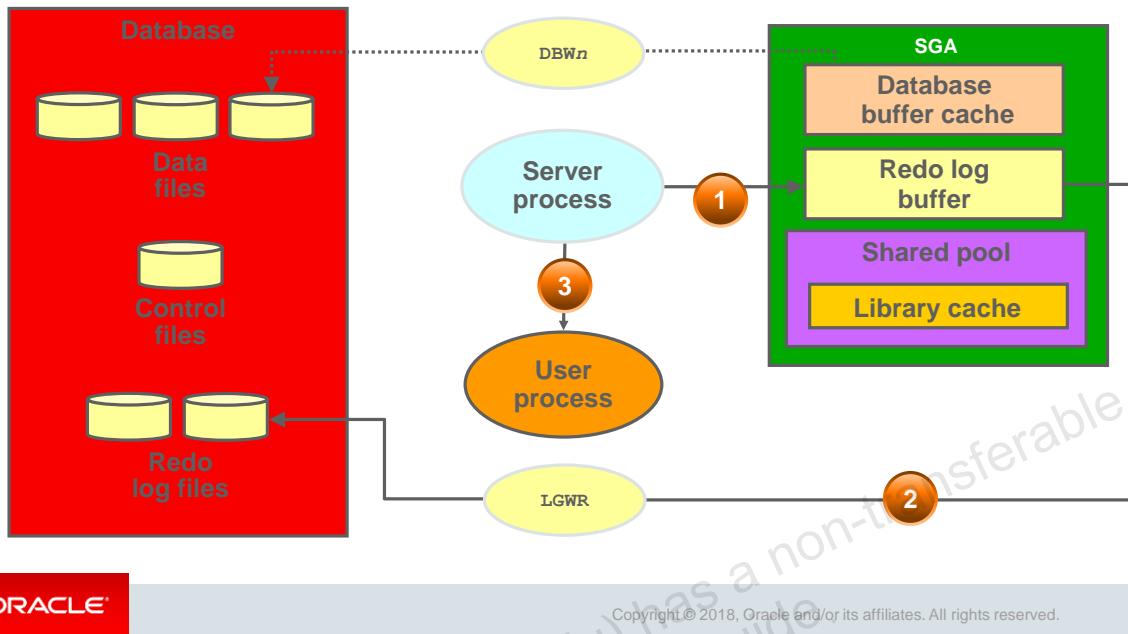
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The steps involved in executing a data manipulation language (DML) statement are:

1. The server process receives the statement and checks the library cache for any shared SQL area that contains a similar SQL statement. If a shared SQL area is found, the server process checks the user's access privileges for the requested data, and the existing shared SQL area is used to process the statement. If not, a new shared SQL area is allocated for the statement, so that it can be parsed and processed.
2. If the data and undo segment 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.
3. 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 undo buffers are modified. This is called write-ahead logging.
4. The undo segment buffers contain values of the data before it is modified. The undo 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.
5. The user gets feedback from the DML operation (such as how many rows were affected by the operation).

Note: Any changed blocks in the buffer cache are marked as dirty buffers; that is, the buffers are not the same as the corresponding blocks on the disk. These buffers are not immediately written to disk by the DBW_n processes.

COMMIT Processing: Example



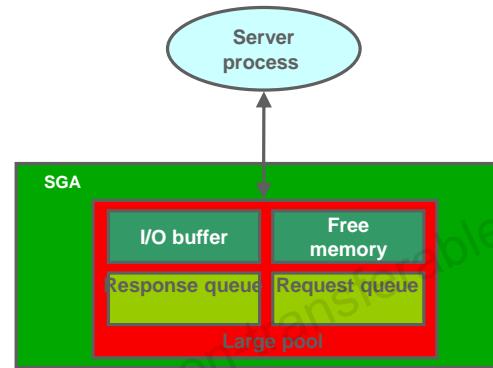
When `COMMIT` is issued, the following steps are performed:

1. The server process places a commit record, along with the system change number (SCN), in the redo log buffer. The SCN is monotonically incremented and is unique within the database. It is used by Oracle Database as an internal time stamp to synchronize data and to provide read consistency when data is retrieved from the data files. Using the SCN enables Oracle Database to perform consistency checks without depending on the date and time of the operating system.
2. The LGWR background process performs a contiguous write of all the redo log buffer entries up to and including the commit record to the redo log files. After this point, Oracle Database can guarantee that the changes are not lost even if there is an instance failure.
3. If modified blocks are still in the SGA, and if no other session is modifying them, then the database removes lock-related transaction information from the blocks. This process is known as commit cleanout.
4. The server process provides feedback to the user process about the completion of the transaction.

Note: If not done already, `DBWn` eventually writes the actual changes back to disk based on its own internal timing mechanism.

Large Pool

- Provides large memory allocations for:
 - Session memory for the shared server and Oracle XA interface
 - Parallel execution buffers
 - I/O server processes
 - Oracle Database backup and restore operations
- Is an optional pool, better suited when using the following:
 - Parallel execution
 - Recovery Manager
 - Shared server



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

You can configure an optional memory area called the large pool to provide large memory allocations for the following components:

- Session memory for the shared server, the Oracle XA interface (used where transactions interact with more than one database), or parallel execution buffers
- I/O server processes
- Oracle Database backup and restore operations

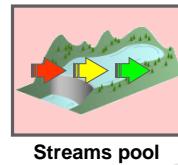
By allocating the memory components from the large pool, Oracle Database can use the shared pool primarily for caching the shared part of SQL and PL/SQL constructs. The shared pool was originally designed to store SQL and PL/SQL constructs. Using the large pool helps avoid fragmentation issues associated with large and small allocations that share the same memory area. Unlike the shared pool, the large pool does not have an LRU list.

You should consider configuring a large pool if your instance uses any of the following:

- **Parallel execution:** Parallel query uses shared pool memory to cache parallel execution message buffers.
- **Recovery Manager:** Recovery Manager uses the shared pool to cache I/O buffers during backup and restore operations.
- **Shared server:** In a shared server architecture, the session memory for each client process is included in the shared pool.

Java Pool and Streams Pool

- Java pool memory is used in server memory for all session-specific Java code and data in the JVM.
- Streams pool memory is used exclusively by Oracle Streams to:
 - Store buffered queue messages
 - Provide memory for Oracle Streams processes



ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

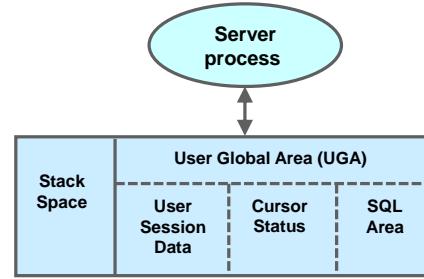
Java pool memory is used for all session-specific Java code and data in the JVM. Java pool memory is used in different ways, depending on the mode in which Oracle Database runs.

Oracle Streams enables the propagation and management of data, transactions, and events in a data stream either within a database or from one database to another. The Streams pool is used exclusively by Oracle Streams. The Streams pool stores buffered queue messages, and it provides memory for Oracle Streams capture and apply processes.

Note: A detailed discussion of Java programming and Oracle Streams is beyond the scope of this course.

Program Global Area

- PGA is a memory area that contains:
 - Session information
 - Cursor information
 - SQL execution work areas:
 - Sort area
 - Hash join area
 - Bitmap Merge area
 - Bitmap Create area
- The size of the work area influences SQL performance.
- Work areas can be managed automatically or manually.



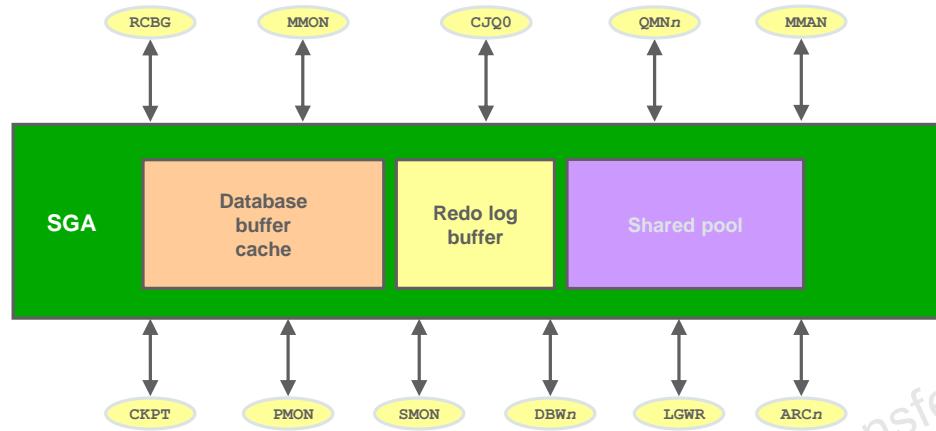
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The PGA can be compared to a temporary countertop workspace used by a file clerk (the server process) to perform a function on behalf of a customer (client process). The clerk clears a section of the countertop, uses the workspace to store details about the customer's request, and then gives up the space when the work is done.

Generally, the PGA memory is divided into the following areas:

- Session memory is the memory allocated to hold a session's variables (login information) and other information related to the session. For a shared server, the session memory is shared and not private.
- Cursors are handles to the private memory structures of specific SQL statements.
- SQL work areas are allocated to support memory-intensive operators, such as those listed in the slide. Generally, bigger work areas can significantly improve the performance of a particular operator at the cost of higher memory consumption.

Background Process



ORACLE®

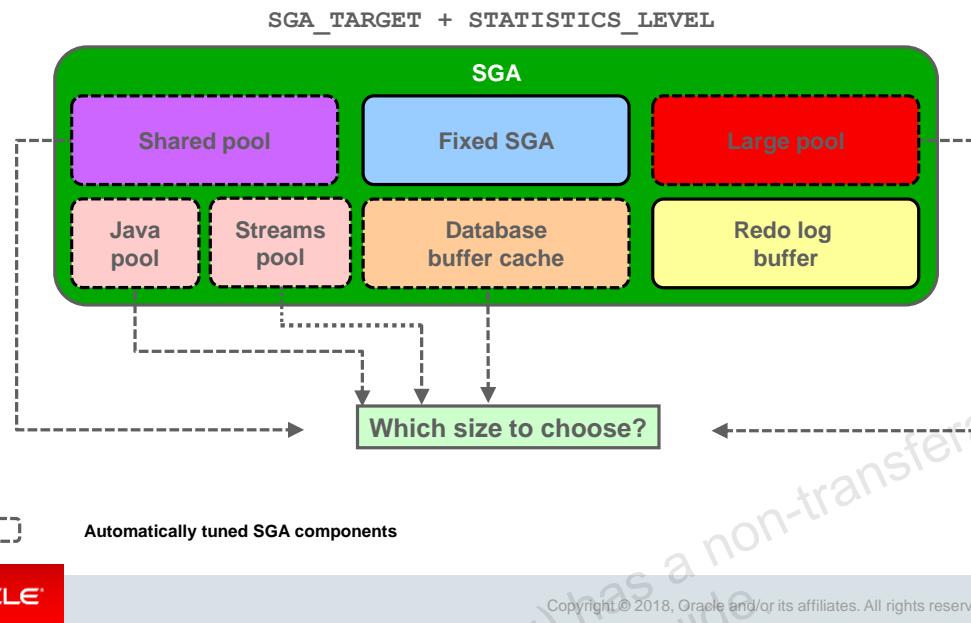
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The background processes commonly seen in non-RAC, non-ASM environments can include the following:

- **Database writer process (DBW n):** Asynchronously writes modified (dirty) buffers in the database buffer cache to disk
- **Log writer process (LGWR):** Writes the recovery information, called redo information, in the log buffer to a redo log file on disk
- **Checkpoint process (CKPT):** Records checkpoint information in control files and each data file header
- **System monitor process (SMON):** Performs recovery at instance startup and cleans up unused temporary segments
- **Process monitor process (PMON):** Performs process recovery when a user process fails
- **Result cache background process (RCBG):** Is used to maintain the result cache in the shared pool

- **Job queue process (CJQ0):** Runs user jobs used in batch processing through the Scheduler
- **Archiver processes (ARC n):** Copy redo log files to a designated storage device after a log switch has occurred
- **Queue monitor processes (QMN n):** Monitor the Oracle Streams message queues
- **Manageability monitoring process (MMON):** Performs manageability-related background tasks
- **Memory manager background process (MMAN):** Is used to manage SGA and PGA memory components automatically

Automatic Shared Memory Management

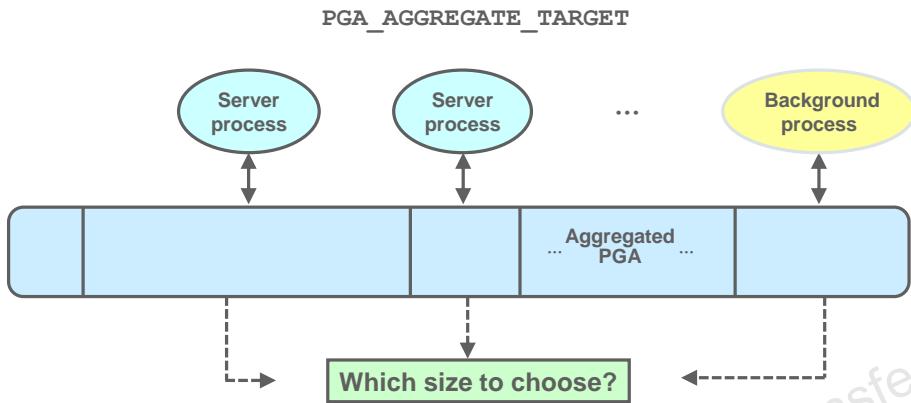


You can use the Automatic Shared Memory Management (ASMM) feature to enable the database to automatically determine the size of each of the memory components listed in the slide within the limits of the total SGA size.

The system uses an SGA size parameter (`SGA_TARGET`) that includes all memory in the SGA, including all automatically sized components, manually sized components, and any internal allocations during startup. ASMM simplifies the configuration of the SGA by enabling you to specify a total memory amount to be used for all SGA components. Oracle Database then periodically redistributes memory between the automatically-tuned components, according to workload requirements.

Note: You must set `STATISTICS_LEVEL` to `TYPICAL` or `ALL` to use ASMM.

Automated SQL Execution Memory Management



ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

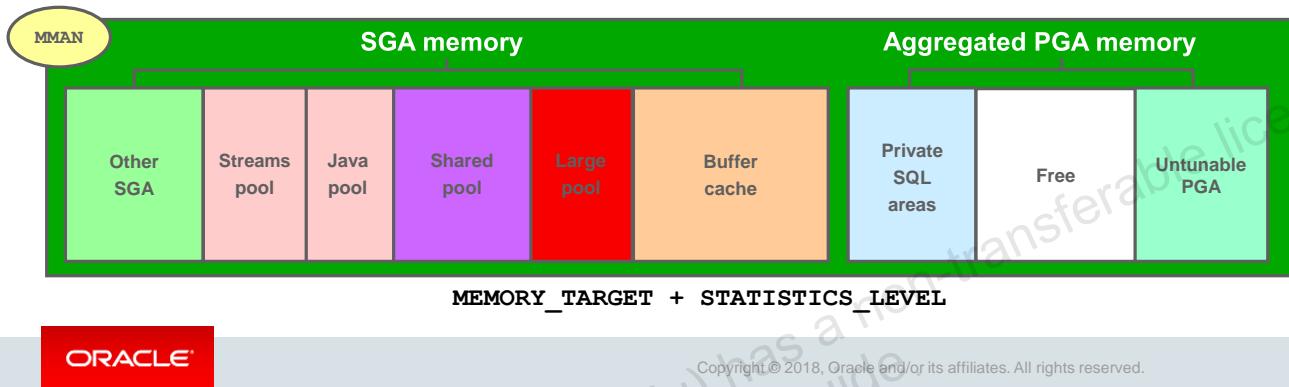
This feature provides an automatic mode for allocating memory to working areas in the PGA. You can use the `PGA_AGGREGATE_TARGET` parameter to specify the total amount of memory that should be allocated to the PGA areas of the sessions of the instance. In automatic mode, working areas that are used by memory-intensive operators (sorts and hash joins) can be adjusted automatically and dynamically.

This feature offers several performance and scalability benefits for decision support system (DSS) workloads and mixed workloads with complex queries. The overall system performance is maximized, and the available memory is allocated more efficiently among queries to optimize both throughput and response time. In particular, the savings that are gained from improved use of memory translate to better throughput at high loads.

Note: In earlier releases of the Oracle Database server, you had to manually specify the maximum work area size for each type of SQL operator, such as sort or hash join. This method proved to be very difficult because the workload changes constantly. Although the current release of Oracle Database supports this manual PGA memory management method that might be useful for specific sessions, it is recommended that you leave automatic PGA memory management enabled.

Automatic Memory Management

- Sizing of each memory component is vital for SQL execution performance.
- It is difficult to manually size each component.
- Automatic Memory Management automates memory allocation of each SGA component and aggregated PGAs.



As seen already, the size of the various memory areas of the instance directly impacts the speed of SQL processing. Depending on the database workload, it is difficult to size those components manually.

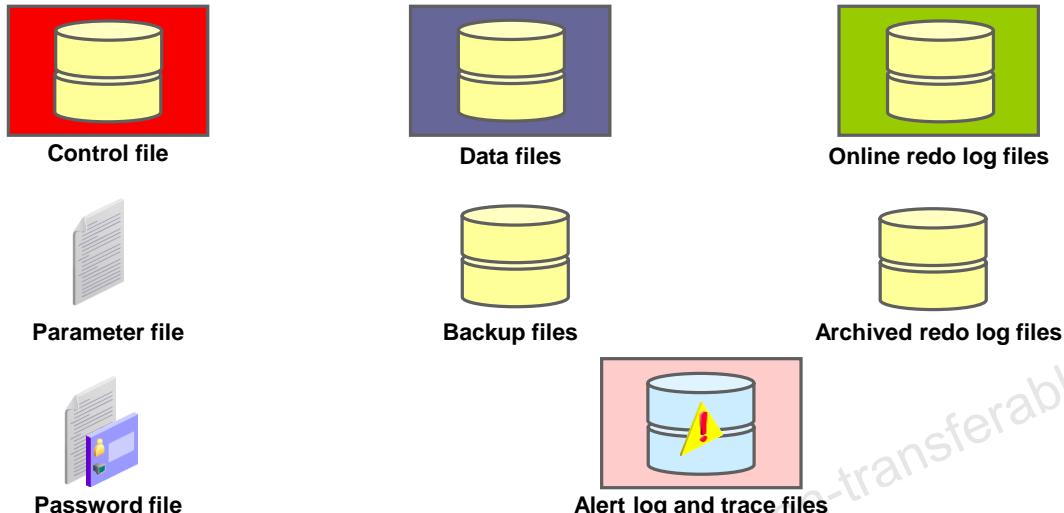
With Automatic Memory Management (AMM), the system automatically adapts the size of each memory's components to your workload memory needs.

When you set your `MEMORY_TARGET` initialization parameter for the database instance, the `MMAN` background process automatically tunes to the target memory size, redistributing memory as needed between the internal components of the SGA and between the SGA and the aggregated PGAs.

The ASSM feature uses the SGA memory broker that is implemented by two background processes: Manageability Monitor (`MMON`) and Memory Manager (`MMAN`). Statistics and memory advisory data are periodically captured in memory by `MMON`. `MMAN` coordinates the sizing of the memory components according to `MMON` decisions.

Note: Currently, this mechanism is implemented only on Linux, Solaris, HP-UX, AIX, and Windows.

Database Storage Architecture



ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The files that constitute an Oracle database are organized into the following:

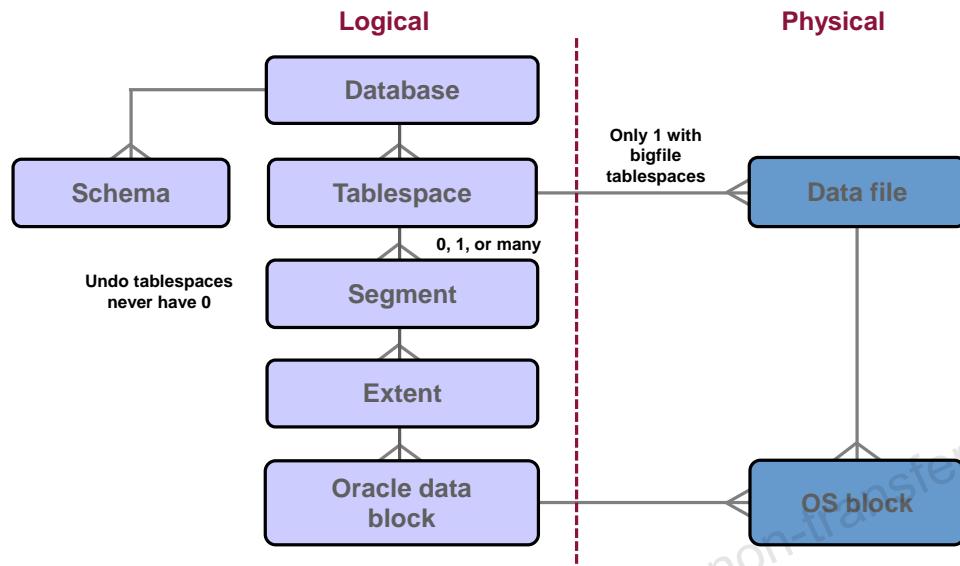
- **Control files:** Contain data about the database itself (that is, physical database structure information). These files are critical to the database. Without them, you cannot open data files to access the data in the database.
- **Data files:** Contain the user or application data of the database, as well as metadata and the data dictionary.
- **Online redo log files:** Allow for instance recovery of the database. If the database server crashes and does not lose any data files, the instance can recover the database with the information in these files.

The following additional files are important for the successful running of the database:

- **Parameter file:** Is used to define how the instance is configured when it starts up
- **Password file:** Allows sysdba, sysoper, and sysasm to connect remotely to the database and perform administrative tasks
- **Backup files:** Are used for database recovery. You typically restore a backup file when a media failure or user error has damaged or deleted the original file.
- **Archived redo log files:** Contain an ongoing history of the data changes (redo) that are generated by the instance. By using these files and a backup of the database, you can recover a lost data file; that is, archive logs enable the recovery of restored data files.

- **Trace files:** Each server and background process can write to an associated trace file. When an internal error is detected by a process, the process dumps information about the error to its trace file. Some of the information written to a trace file is intended for the developer, whereas other information is for Oracle Support Services.
- **Alert log file:** These files are special trace entries. The alert log of a database is a chronological log of messages and errors. Each instance has one alert log file. It is recommended that you periodically review this file.

Logical and Physical Database Structures



ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The database has logical structures and physical structures.

Tablespaces

A database is divided into logical storage units called tablespaces, which group related logical structures together. For example, tablespaces commonly group all the objects of an application to simplify some administrative operations. You may have a tablespace for different applications.

Databases, Tablespaces, and Data Files

The relationship among databases, tablespaces, and data files is illustrated in the slide. Each database is logically divided into one or more tablespaces. One or more data files are explicitly created for each tablespace to physically store the data of all logical structures in a tablespace. If it is a TEMPORARY tablespace instead of a tablespace containing data, the tablespace has a temporary file.

Schema

A schema is a collection of database objects that are owned by a database user. Schema objects are the logical structures that directly refer to the data contained in a database. Schema objects include structures such as tables, views, sequences, stored procedures, synonyms, indexes, clusters, and database links. In general, schema objects include everything that your application creates in the database.

Data Blocks

At the finest level of granularity, the data contained in an Oracle database is stored in data blocks. One data block corresponds to a specific number of bytes of physical database space on the disk. A data block size is specified for each tablespace when it is created. A database uses and allocates free database space in Oracle data blocks.

Extents

The next level of logical database space is an extent. An extent is a specific number of contiguous data blocks (obtained in a single allocation) that are used to store a specific type of information.

Segments

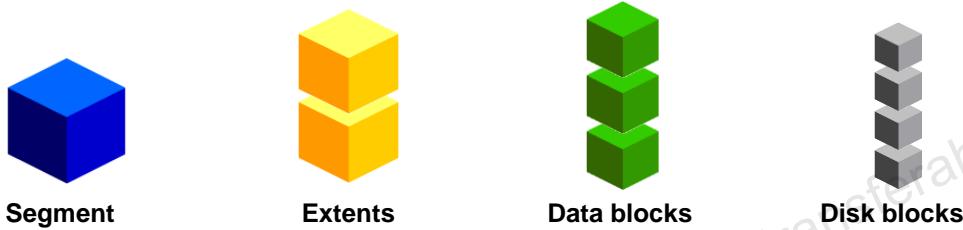
The level of logical database storage above an extent is called a segment. A segment is a set of extents that are allocated for a certain logical structure. Different types of segments include:

- **Data segments:** Each nonclustered, non-index-organized table has a data segment, with the exception of external tables and global temporary tables that have no segments, and partitioned tables in which each table has one or more segments. All the data contained in the table is stored in the extents of its data segment. For a partitioned table, each partition has a data segment. Each cluster has a data segment. The data of every table in the cluster is stored in the data segment of the cluster.
- **Index segments:** Each index has an index segment that stores all of its data. For a partitioned index, each partition has an index segment.
- **Undo segments:** One UNDO tablespace is created for each database instance. This tablespace contains numerous undo segments to temporarily store undo information. The information in an undo segment is used to generate read-consistent database information and, during database recovery, to roll back uncommitted transactions for users.
- **Temporary segments:** Temporary segments are created by the Oracle Database when a SQL statement needs a temporary work area to complete execution. When the statement finishes execution, the temporary segment's extents are returned to the instance for future use. Specify either a default temporary tablespace for every user, or a default temporary tablespace that is used across the database.

Oracle Database dynamically allocates space. When the existing extents of a segment are full, additional extents are added. Because extents are allocated as needed, the extents of a segment may or may not be contiguous on the disk.

Segments, Extents, and Blocks

- Segments exist in a tablespace.
- Segments are collections of extents.
- Extents are collections of data blocks.
- Data blocks are mapped to disk blocks.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Database objects such as tables and indexes are stored as segments in tablespaces. Each segment contains one or more extents. An extent consists of contiguous data blocks, which means that each extent can exist in only one data file. Data blocks are the smallest units of I/O in the database.

When the database requests a set of data blocks from the operating system (OS), the OS maps this set to an actual file system or disk block on the storage device. Because of this, you do not need to know the physical address of any of the data in your database. This also means that a data file can be striped or mirrored on several disks.

The size of the data block can be set at the time of database creation. The default size of 8 KB is adequate for most databases. If your database supports a data warehouse application that has large tables and indexes, a larger block size may be beneficial.

If your database supports a transactional application in which reads and writes are random, specifying a smaller block size may be beneficial. The maximum block size depends on your OS. The minimum Oracle block size is 2 KB; it should rarely (if ever) be used.

You can have tablespaces with a nonstandard block size. For details, see the *Oracle Database Administrator's Guide*.

SYSTEM and SYSAUX Tablespaces

- The SYSTEM and SYSAUX tablespaces are mandatory tablespaces that are created at the time of database creation. They must be online.
- The SYSTEM tablespace is used for core functionality (for example, data dictionary tables).
- The auxiliary SYSAUX tablespace is used for additional database components (such as the Enterprise Manager Repository).



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Each Oracle database must contain a SYSTEM tablespace and a SYSAUX tablespace, which are automatically created when the database is created. The system default is to create a smallfile tablespace. You can also create bigfile tablespaces, which enable the Oracle database to manage ultralarge files (up to 8 exabytes in size).

A tablespace can be online (accessible) or offline (not accessible). The SYSTEM tablespace is always online when the database is open. It stores tables that support the core functionality of the database, such as the data dictionary tables.

The SYSAUX tablespace is an auxiliary tablespace to the SYSTEM tablespace. The SYSAUX tablespace stores many database components, and it must be online for the correct functioning of all database components.

Note: The SYSAUX tablespace may be taken offline for performing tablespace recovery, whereas this is not possible in the case of the SYSTEM tablespace. Neither of them may be made read-only.

Quiz



The first time an Oracle Database server process requires a particular piece of data, it searches for the data in the:

- a. Database buffer cache
- b. PGA
- c. Redo log buffer
- d. Shared pool



ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Answer: a

Quiz



Which of the following is not a database logical structure?

- a. Tablespace
- b. Data file
- c. Schema
- d. Segment



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Answer: b

Quiz



The `SYSAUX` tablespace is used for core functionality, and the `SYSTEM` tablespace is used for additional database components, such as the Enterprise Manager Repository.

- a. True
- b. False



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Answer: b

Summary

In this lesson, you should have learned how to:

- List the major architectural components of the Oracle Database server
- Explain memory structures
- Describe background processes
- Correlate logical and physical storage structures



ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2019, Oracle and/or its affiliates.

GANG LIU (gangl@baylorhealth.edu) has a non-transferable license
to use this Student Guide.

E

Real-Time Database Operation Monitoring

ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe database operations
- Implement Real-Time Database Operation Monitoring



ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

This lesson provides an overview of the Oracle Database server architecture. You learn about its architectural components, memory structures, background processes, and logical and physical storage structures.

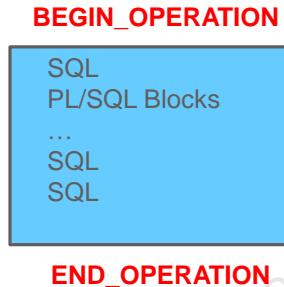
Real-Time Database Operation Monitoring: Overview

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



Copyright© 2018, 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 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, PL/SQL procedure, 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 to which batch job 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. Real-Time Database Operations Monitoring allows these jobs to be monitored while they are running.

Real-Time Database Operation Monitoring identifies an operation with a tag, determines what to monitor, and can generate reports. Real-Time Database Operation Monitoring monitors the progress of the operation, packages raw data for offline analysis and creates active reports that do not require access to production systems.

Use Cases

- Monitor batch jobs:
 - Set expected times; be alerted 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© 2018, 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 it can be killed, fixed, and restarted 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, then compare the two sets and identify the changes, for example, SQL execution plan, resource consumption, and degree of parallelism.

Current Tools

- Real-Time SQL Monitor:
 - Is a subset of Database Operation Monitoring
 - Monitors long-running single SQL statements and PL/SQL functions
 - Displays statistics for recently completed statements
- Active Session History:
 - 1-second sample
 - Useful for long running queries
- SQL Trace:
 - Large overhead
 - Incident must be reproducible.
- DBMS_MONITOR and trcsess utility
- V\$SESSION_LONGOPS and DBMS_APPLICATION_INFO



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

- **Real-Time SQL Monitor:** A feature added in 11.1, this is a subset of Database Operation Monitoring. Its main focus is to help DBAs know where a SQL statement is in its execution plan and where the time is being spent. It allows users to monitor the performance of SQL statements while they are executing as well as to see the breakdown of time and resources used for recently completed statements. With SQL Monitor, users can find the expensive (high response-time) SQLs (or PL/SQL functions) on their OLTP system, identify the expensive SQL statements in ETL, DDL, batch, reporting jobs, and check parallel-executed statements. It highlights poorly performed SQL: users can visualize hundreds of operations, find out why a particular operation is so expensive, and what is going on inside a SQL execution.
- **Active session history (ASH):** Contains samples of active sessions taken every second. It is on by default, so ASH can be used even if problems are not reproducible. Because samples are collected every second, ASH will contain more information for long-running queries. Unless the queries are very long, ASH is not very useful for tuning response time.
- **SQL Trace:** Produces a lot of detailed information. The information is hard to interpret. The problem must be reproducible to enable SQL Trace to capture the details.
- **DBMS_MONITOR package:** Enables the trace for a given database session, module, action, or service and the `trcsess` command-line utility that consolidates tracing information
- **V\$SESSION_LONGOPS view and DBMS_APPLICATION_INFO.SET_SESSION_LONGOPS procedures:** Useful for long-running operations, mainly to indicate how much work has been processed so far

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 or PL/SQL
- Composite
 - Multiple SQL statements and/or PL/SQL procedures and functions between two points in time
 - Single session
 - Multiple concurrent sessions



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

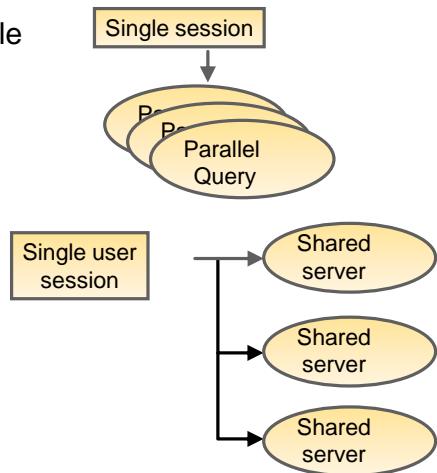
A DB operation is an operation that the database server runs to accomplish tasks. It can be either simple or composite:

- **Simple database operations:** Consist of one single SQL statement or one single PL/SQL function/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 cases.
 - In one case, exactly one session exists for the duration of the database operation.
 - In the other case, 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

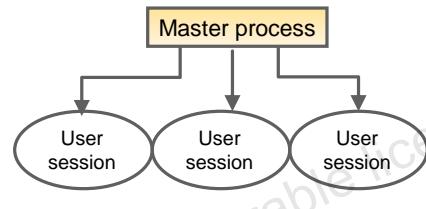
Single session

- Parallel query with single coordinator
- Single-user session connected to a server session sequentially



Multiple sessions

An operation using multiple user sessions concurrently



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

A composite operation in the scope of a single session is composed of one of the following:

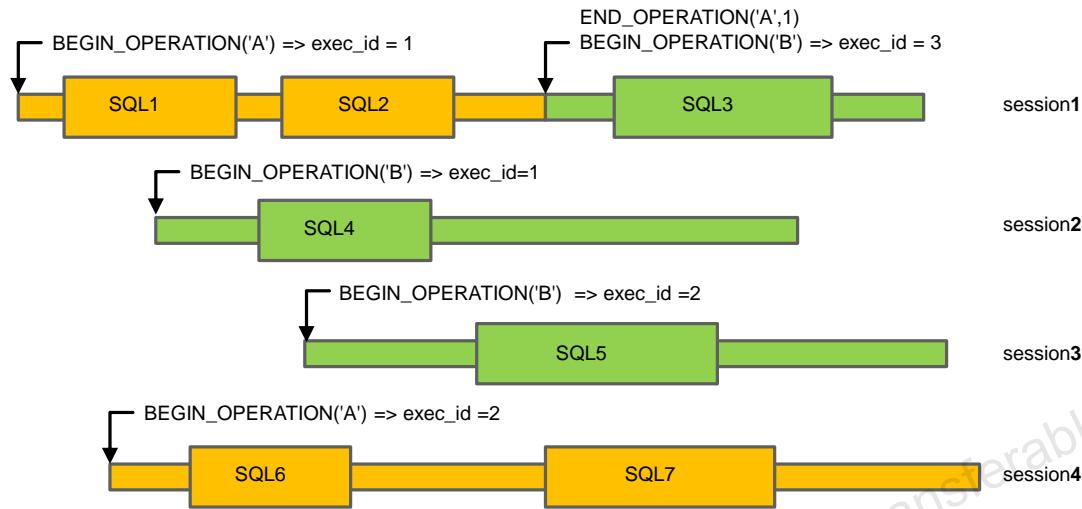
- SQL statements coordinated by the single session, such as parallel operations
- The active session, such as a user session in an application pool or shared server

Diagrams illustrating single-session and single-user session examples of a database operation are shown on the left side of the slide.

Note: These examples do not exclude a single session with statements that are neither non-parallel nor using shared servers.

A composite operation in the scope of multiple sessions is composed of SQL statements running concurrently such as during an ETL operation. This is shown in the diagram on the right side of the slide.

Database Operation Concepts



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In the graphic in the slide, there are four sessions. The first (upper) session shows that a session can belong to at most 1 database operation at a time. It belongs to database operation A at first, then belongs to database operation B. The END_OPERATION and then BEGIN_OPERATION commands cause the session to change database operations.

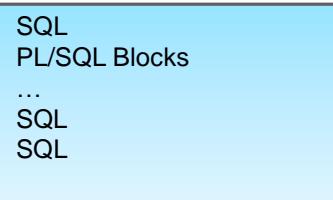
Sessions 1, 2, and 3 show that the name and execution ID are taken together to uniquely identify database operation B.

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

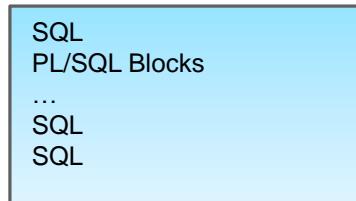
- Naming

DBOP (Name)



- Bracketing

DBMS_SQL_MONITOR.BEGIN_OPERATION



ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

To monitor a database operation, the database operation must be given a name, a begin point, and an end point. There are two ways to name a database operation. The first is to insert a start point by using the `DBMS_SQL_MONITOR.BEGIN_OPERATION` function and an end point by using `DBMS_SQL_MONITOR.END_OPERATION` procedure calls into the application. The second way is to set a tag. This is for JAVA and OCI applications. The tag can be set by using the OCI calls `OCIAttrSet` and `OCIApplCtxSet`, the Java call `setClientInfo`, or from the OS environment variable `ORA_DBOP`. The tag in Java or OCI piggybacks the next database call so the begin point is easily distinguished.

The explicit calls provide a clear beginning and end for 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, the tag should be set to `NULL` and sent to the database.

Enabling Monitoring of Database Operations

- At system level:
 - Set STATISTICS_LEVEL to TYPICAL.
 - Set CONTROL_MANAGEMENT_PACK_ACCESS to DIAGNOSTIC+TUNING.



- At database operation level:
 - Use the FORCED_TRACKING attribute in the DBMS_SQL_MONITOR.BEGIN_OPERATION function.
- At statement level:
 - Use the MONITOR hint.



ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The SQL monitoring feature is enabled by default when the STATISTICS_LEVEL initialization parameter is set to either TYPICAL (the default value) or ALL. Additionally, the CONTROL_MANAGEMENT_PACK_ACCESS parameter must be set to DIAGNOSTIC+TUNING (the default value) because SQL monitoring is a feature of the Oracle Database Tuning Pack. By default, SQL monitoring is started 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 database operation to be tracked when it starts. The default value of this parameter does not necessarily force any SQL statement or database operation to be tracked: The database operation is tracked only when it is sufficiently expensive. Use the FORCED_TRACKING attribute and set it to Y when starting the database operation with the DBMS_SQL_MONITOR.BEGIN_OPERATION function:

```
DBMS_MONITOR.BEGIN_OPERATION (
    dbop_name IN VARCHAR2,
    dbop_eid IN NUMBER :=NULL,
    forced_tracking IN VARCHAR2 := NO_FORCE_TRACKING,
    attribute_list IN VARCHAR2 :=NULL)
RETURN NUMBER;
```

Identifying, Starting, and Completing a Database Operation

1. Identify the database operation.
 - Operation name
 - 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)
```



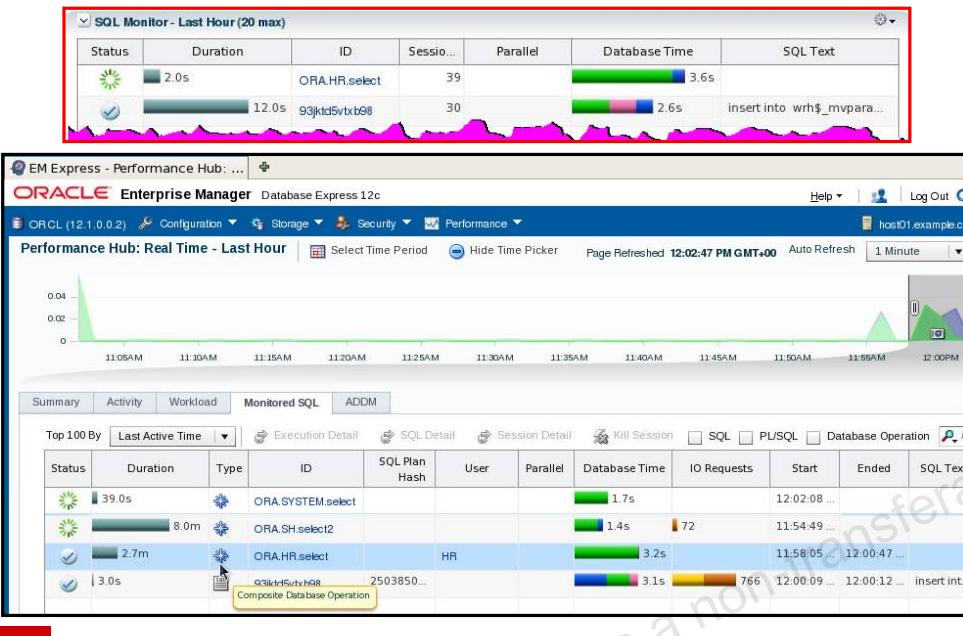
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

When monitoring, each database operation can be identified by the operation name and execution ID. Every DB operation has a name and an execution ID. The DB operation name and execution ID 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 will have 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 call to BEGIN_OPERATION returns the execution ID. If an execution ID is supplied to the call, it is returned. If the execution ID is not supplied, a unique execution ID is returned. In situations where a master process spawns several sessions that are associated with the same DB operation, an execution ID would be generated for the entire operation and each session would set 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 is for an operation that performs a materialized view refresh.

The different executions of the same database operation (with the same name) can have different SQL statements or PL/SQL functions. This could be caused by different code paths being taken through conditions in PL/SQL. There is also nothing to prevent one operation name being assigned to multiple sets of code. This could cause confusion.

Monitoring the Progress of a Database Operation



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Enterprise Manager Database Express and Enterprise Manager Cloud Control allow you to view database operations that are active and recently finished.

Using Enterprise Manager Database Express, on the Database Home page, the list of monitored SQL and database operations appears on the bottom-right side of the page as shown in the first screenshot in the slide.

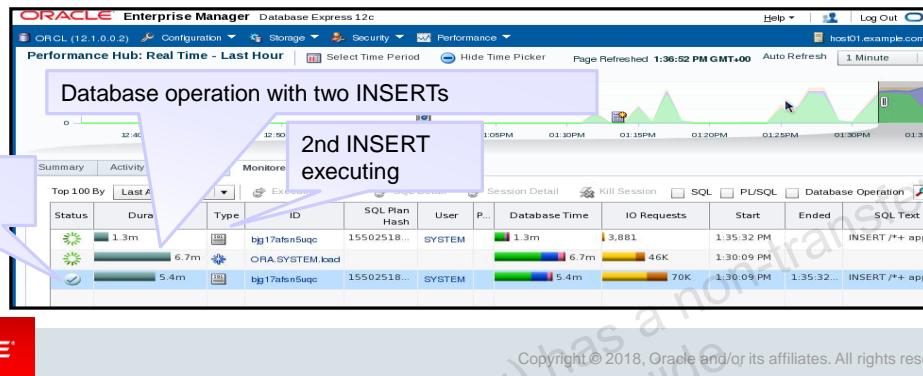
In the Performance menu, click the Performance Hub option to get a list of monitored SQL and database operations, as shown in the second screenshot in the slide. For details, click the database operation name. On the Details page, the execution ID is shown. By selecting the Database Operations filter above the list (SQL, PL/SQL and Database Operations), only database operations can be shown. Further filtering can be done on the operation name by using the search box on the right above the list.

Using Enterprise Manager Cloud Control, log in to the target database and click the Performance menu, then the SQL Monitoring option.

Monitoring Load Database Operations

```
SQL> VAR dbop_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.

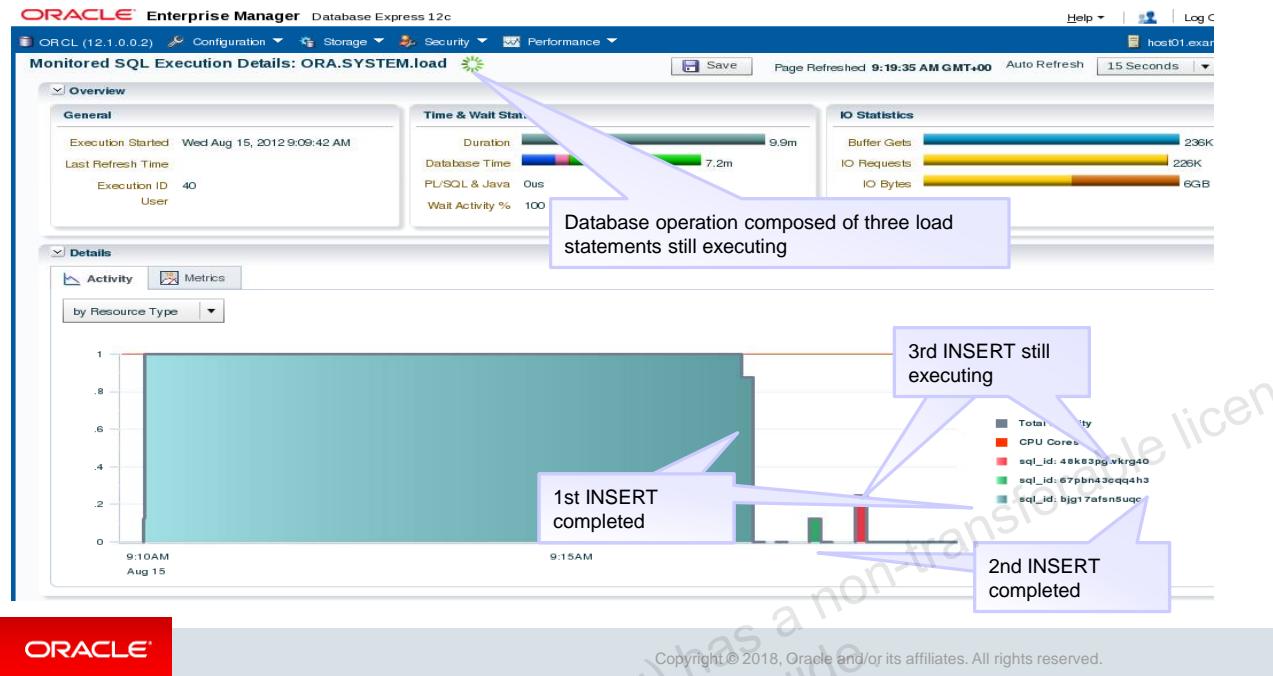


Monitoring load operations can be performed. From the example shown in the slide, the database operation consists of two bulk load statements.

Using Enterprise Manager Database Express, from the Performance Hub page, you see that the database load operation is still executing whereas the first bulk load statement is already completed. Another bulk load statement is still executing, being part of the same database load operation that is being monitored.

Using Enterprise Manager Cloud Control, logged in to a target database, from the Performance menu and SQL Monitoring option, you would see the same database load operation executing.

Monitoring Load Database Operation Details



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.

Reporting Database Operations by Using Views

V\$SQL_MONITOR view shows a list of executing and completed database operations through new columns.

```
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 © 2018, 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 executions in the operation.

The primary key is a combination of the columns DBOP_NAME, DBOP_EXEC_ID, and SQL_ID.

Reporting Database Operations by Using Views

- EM Cloud Control and EM Database Express use multiple views:
 - V\$SQL_MONITOR, V\$ACTIVE_SESSION_HISTORY,
and new columns: DBOP_NAME, DBOP_EXEC_ID
- AWR automatically captures SQL monitoring reports and stores active session history:
 - DBA_HIST_REPORTS, DBA_HIST_REPORTS_DETAILS
 - DBA_HIST_ACTIVE_SESS_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 © 2018, Oracle and/or its affiliates. All rights reserved.

EM Cloud Control or EM 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 through 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

DBMS_SQL_MONITOR package:

- REPORT_SQL_MONITOR_LIST_XML and REPORT_SQL_MONITOR_XML functions report a list and details of database operations in XML format.
- REPORT_SQL_MONITOR_LIST and REPORT_SQL_MONITOR functions report a list and details of database operations in a CLOB.

```
SQL> SELECT DBMS_SQL_MONITOR.REPORT_SQL_MONITOR_LIST_XML()
  2  FROM  dual;
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Reporting by Using DBMS_SQL_MONITOR.REPORT_SQL_MONITOR_xxx Functions

```
SQL> select DBMS_SQL_MONITOR.REPORT_SQL_MONITOR_LIST_XML() from
dual;
```

```
DBMS_SQL_MONITOR.REPORT_SQL_MONITOR_LIST_XML()
-----
-
<report db_version="12.1.0.0.2" cpu_cores="1" hyperthread="N"
timezone_offset="0
" elapsed_time="0.00" cpu_time="0.00">
<report_id><![CDATA[/orarep/sqlmonitor/list?monitor_type=3]]></r
eport_id>
<sql_monitor_list_report version="1" sysdate="08/14/2012
10:33:04">
    <dbop detail="F" dbop_name="ORA.SH.select2"
dbop_exec_start="08/14/2012 10:26:11" dbop_exec_id="12">
        <status>EXECUTING</status>
        <first_refresh_time>08/14/2012
10:26:11</first_refresh_time>
```

```
<refresh_count>0</refresh_count>
    <inst_id>1</inst_id>
<session_id>43</session_id>
    <session_serial>229</session_serial>
    <user_id>0</user_id>
    <con_id>0</con_id>
        <module>SQL*Plus</module>
    <service>SYS$USERS</service>
    <program>sqlplus@host01.example.com (TNS V1-
V3)</program>
        <plsql_entry_object_id>10531</plsql_entry_object_id>

<plsql_entry_subprogram_id>6</plsql_entry_subprogram_id>
    <plsql_object_id>10375</plsql_object_id>
    <plsql_subprogram_id>63</plsql_subprogram_id>
    <is_cross_instance>N</is_cross_instance>
        <stats type="monitor">
            <stat name="duration">413</stat>
            <stat name="elapsed_time">3072887</stat>
            <stat name="cpu_time">1143825</stat>
            <stat name="user_io_wait_time">368882</stat>
            <stat name="application_wait_time">475</stat>
            <stat name="other_wait_time">1559705</stat>
            <stat name="buffer_gets">23</stat>
            <stat name="read_reqs">37</stat>
            <stat name="read_bytes">13033472</stat>
            <stat name="write_reqs">52</stat>
            <stat name="write_bytes">42041344</stat>
        </stats>
    </dbop>
```

Quiz



Monitoring database operations is allowed for sets of SELECT statements only.

- a. True
- b. False



ORACLE

Copyright© 2018, 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



ORACLE

Copyright© 2018, 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



ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2019, Oracle and/or its affiliates.

GANG LIU (gangl@baylorhealth.edu) has a non-transferable license
to use this Student Guide.



Gathering and Managing Optimizer Statistics

ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Gather optimizer statistics
- Set statistic preferences
- Use dynamic statistics
- Use automatic re-optimization
- Use SQL plan directives
- Discuss online statistics gathering for bulk loads
- Discuss concurrent statistics gathering
- Manage optimizer statistics



ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

- Gathering Statistics
- Dynamic Statistics
- Automatic Re-optimization
- SQL Plan Directives
- Online Statistics Gathering for Bulk Loads
- Concurrent Statistics Gathering
- Gathering System Statistics
- Managing Optimizer Statistics



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Gathering Statistics: Overview

- Optimizer statistics collection is the gathering of optimizer statistics for database objects.
- You can gather statistics by using the following:
 - Automatic optimizer statistics gathering
 - Manual statistics gathering
 - DBMS_STATS package
 - Adaptive statistics
 - Dynamic statistics
 - Automatic re-optimization
 - SQL plan directives
 - Online statistics gathering for bulk loads

Selectivity:	
Equality	1%
Inequality	5%
Other predicates	5%
Table row length	20
# of index leaf blocks	25
# of distinct values	100
Table cardinality	100
Remote table cardinality	2000



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In Oracle Database, optimizer statistics collection is the gathering of optimizer statistics for database objects. The database can collect these statistics automatically, or you can collect them manually by using the system-supplied DBMS_STATS package. These are discussed in more detail in subsequent slides. It is recommended that you use automatic statistics gathering for objects.

The contents of tables and associated indexes change frequently, which can lead the optimizer to choose suboptimal execution plan for queries. Thus, statistics must be kept current to avoid any potential performance issues because of suboptimal plans.

To minimize DBA involvement, Oracle Database automatically gathers optimizer statistics at various times. Some automatic options are configurable, such as enabling AutoTask to run DBMS_STATS.

You can manage optimizer statistics either through Oracle Enterprise Manager Cloud Control (Cloud Control) or by using PL/SQL on the command line.

Note: When the system encounters a table with missing statistics, it dynamically gathers the necessary statistics needed by the optimizer. However, for certain types of tables (including remote tables and external tables), it does not perform dynamic statistics. In those cases, and also when dynamic statistics has been disabled, the optimizer uses default values for its statistics.

Automatic Optimizer Statistics Gathering

- Statistics are gathered only automatically on all database objects that have no statistics or have stale statistics (> 10% of rows modified) by running an Oracle AutoTask task during a predefined maintenance window.
- Automated statistics collection:
 - Eliminates the need for manual statistics collection
 - Significantly reduces the chances of poor execution plans
- The automatic statistics gathering job uses the `DBMS_STATS.GATHER_DATABASE_STATS_JOB_PROC` procedure, which uses the same default parameter values as the other `DBMS_STATS.GATHER_*_STATS` procedures.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Oracle Database automatically gathers statistics on all database objects and maintains those statistics in a regularly scheduled maintenance job. The automated maintenance tasks infrastructure (known as AutoTask) schedules tasks to run automatically in Oracle Scheduler windows known as maintenance windows. By default, one window is scheduled for each day of the week. Automatic optimizer statistics collection runs as part of AutoTask. By default, the collection runs in all predefined maintenance windows. Automated statistics collection eliminates all of the manual tasks that are associated with managing the optimizer statistics, and significantly reduces the chances of getting poor execution plans because of missing or stale statistics.

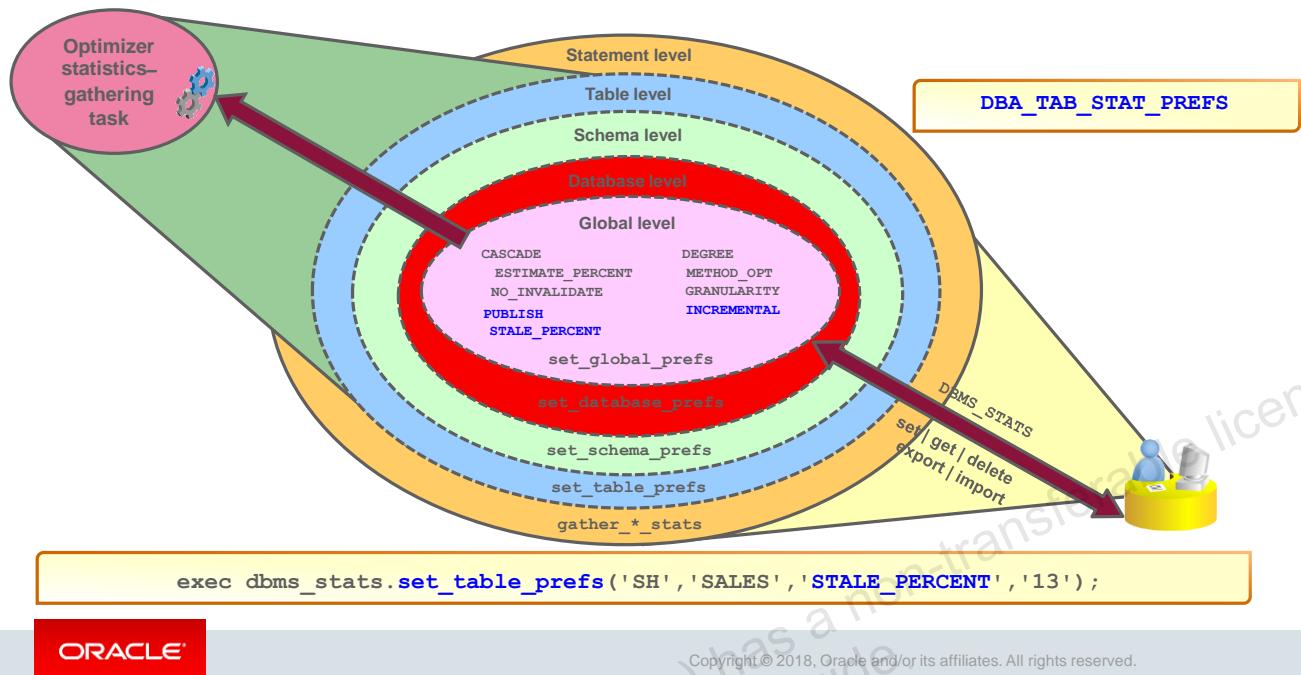
To collect the optimizer statistics, the database calls an internal procedure that operates similarly to the `GATHER_DATABASE_STATS` procedure with the `GATHERAUTO` option. Automatic statistics collection honors all preferences set in the database. The principal difference between manual and automatic collection is that the latter prioritizes database objects that need statistics. Before the maintenance window closes, automatic collection first processes objects that most require updated statistics. DML monitoring is enabled by default and is used by the Automatic Statistics Gathering feature to determine which objects have stale statistics as well as their degree of staleness. This information is used to identify candidates for statistics update.

Optimizer statistics are automatically gathered by using the automatic optimizer statistics–gathering task. This task does the following:

- Gathers statistics on all objects in the database that have missing or stale statistics in the predefined maintenance window
- Determines the appropriate sample size for each object
- Creates histograms as required

The task is created automatically at database creation time. System statistics are *not* gathered by the automatic optimizer statistics–gathering task.

Optimizer Statistic Preferences: Overview



The Automated Statistics Gathering feature was introduced in Oracle Database 10g Release 1 to reduce the burden of maintaining optimizer statistics. However, there were cases where you had to disable it and run your own scripts instead. One reason was the lack of object-level control. Whenever you found a small subset of objects for which the default gather statistics options did not work well, you had to lock the statistics and analyze them separately by using your own options. For example, the feature that automatically tries to determine adequate sample size (`ESTIMATE_PERCENT=AUTO_SAMPLE_SIZE`) does not work well against columns that contain data with very high frequency skews. The only way to get around this issue is to manually specify the sample size in your own script.

The Statistic Preferences feature in Oracle Database introduces flexibility so that you can rely more on the automated statistics-gathering feature to maintain the optimizer statistics when some objects require settings that differ from the database default.

This feature allows you to associate the statistics-gathering options that override the default behavior of the `GATHER_*_STATS` procedures and the automated Optimizer Statistics Gathering task at the object or schema level. You can use the `DBMS_STATS` package to manage the statistics-gathering options shown in the slide.

You can set, get, delete, export, and import those preferences at the table, schema, database, and global levels. Global preferences are used for tables that do not have preferences, whereas database preferences are used to set preferences on all tables. The preference values that are specified in various ways take precedence from the outer circles to the inner ones (as shown in the slide).

- **CASCADE** gathers statistics on the indexes as well. Index statistics gathering is not parallelized.
- **ESTIMATE_PERCENT** is the estimated percentage of rows used to compute statistics (null means all rows): The valid range is [0.000001,100]. Use the constant `DBMS_STATS.AUTO_SAMPLE_SIZE` to have the system determine the appropriate sample size for good statistics. This is the recommended default.
- **NO_INVALIDATE** controls the invalidation of dependent cursors of the tables for which statistics are being gathered. It does not invalidate the dependent cursors if set to `TRUE`. The procedure invalidates the dependent cursors immediately if set to `FALSE`. Use `DBMS_STATS.AUTO_INVALIDATE` to have the system decide when to invalidate dependent cursors. This is the default.
- **PUBLISH** is used to decide whether to publish the statistics to the dictionary or to store them in a pending area before publishing them.
- **STALE_PERCENT** is used to determine the threshold level at which an object is considered to have stale statistics. The value is a percentage of rows that were modified since the last statistics gathering. The example changes the 10 percent default to 13 percent only for `SH.SALES`.
- **DEGREE** determines the degree of parallelism used to compute statistics. The default for `degree` is null, which means use the table default value specified by the `DEGREE` clause in the `CREATE TABLE` or `ALTER TABLE` statement. Use the constant `DBMS_STATS.DEFAULT_DEGREE` to specify the default value based on the initialization parameters. The `AUTO_DEGREE` value automatically determines the degree of parallelism as either 1 (serial execution) or `DEFAULT_DEGREE` (the system default value based on the number of CPUs and initialization parameters), depending on the size of the object.
- **METHOD_OPT** is a SQL string used to collect histogram statistics. The default value is `FOR ALL COLUMNS SIZE AUTO`.
- **GRANULARITY** is the granularity of statistics to collect for partitioned tables.
- **INCREMENTAL** is used to gather global statistics on partitioned tables in an incremental way.

It is important to note that you can change default values for the parameters by using the `DBMS_STATS.SET_GLOBAL_PREFS` procedure.

Note: You can describe all effective statistics preference settings for all relevant tables by using the `DBA_TAB_STAT_PREFS` view.

Manual Statistics Gathering

You can use Enterprise Manager and the `DBMS_STATS` package to:

- Generate and manage statistics for use by the optimizer:
 - Gather/Modify
 - View/Name
 - Export/Import
 - Delete/Lock
- Gather statistics on:
 - Indexes, tables, columns, partitions
 - Object, schema, or database
- Gather statistics either serially or in parallel
- Gather/set system statistics



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Both Enterprise Manager and the `DBMS_STATS` package enable you to manually generate and manage statistics for the optimizer. You can use the `DBMS_STATS` package to gather, modify, view, export, import, lock, and delete statistics. You can also use this package to identify or name gathered statistics. You can gather statistics on indexes, tables, columns, and partitions at various levels of granularity: object, schema, and database level.

`DBMS_STATS` gathers only statistics needed for optimization; it does not gather other statistics. For example, the table statistics that are gathered by `DBMS_STATS` include the number of rows, number of blocks currently containing data, and average row length, but not the number of chained rows, average free space, or number of unused data blocks.

Note: Do not use the `COMPUTE` and `ESTIMATE` clauses of the `ANALYZE` statement to collect optimizer statistics. These clauses are supported solely for backward compatibility and may be removed in a future release. The `DBMS_STATS` package collects a broader, more accurate set of statistics, and gathers statistics more efficiently. You may continue to use the `ANALYZE` statement for other purposes that are not related to the optimizer statistics collection, such as the following:

- To use the `VALIDATE` or `LIST CHAINED ROWS` clauses
- To collect information on free list blocks

When to Gather Statistics Manually

- Rely mostly on automatic statistics collection:
 - Change the frequency of automatic statistics collection to meet your needs.
 - Remember that `STATISTICS_LEVEL` should be set to `TYPICAL` or `ALL` for automatic statistics collection to work properly.
- Gather statistics manually for:
 - Objects that are volatile
 - Objects modified in batch operations (gather statistics as part of the batch operation)
 - External tables, system statistics, and fixed objects
 - New objects (gather statistics right after object creation)



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The automatic statistics gathering mechanism gathers statistics on schema objects in the database for which statistics are absent or stale. It is important to determine when and how often to gather new statistics. The default gathering interval is nightly, but you can change this interval to suit your business needs. You can do so by changing the characteristics of your maintenance windows. Some cases may require manual statistics gathering. For example, the statistics on tables that are significantly modified during the day may become stale. There are typically two types of such objects:

- Volatile tables that are modified significantly during the course of the day
- Objects that are the target of large bulk loads that add 10 percent or more to the object's total size between statistics-gathering intervals

For external tables, statistics are collected manually only by using `GATHER_TABLE_STATS`. Because sampling on external tables is not supported, the `ESTIMATE_PERCENT` option should be explicitly set to null. Because data manipulation is not allowed against external tables, it is sufficient to analyze external tables when the corresponding file changes. Other areas in which statistics need to be manually gathered are system statistics and fixed objects, such as dynamic performance tables. These statistics are not automatically gathered.

Manual Statistics Collection: Factors

- Monitor objects for DML operations.
- Determine the correct sample sizes.
- Determine the degree of parallelism.
- Determine if histograms should be used.
- Determine the cascading effect on indexes.

Procedures to use in DBMS_STATS:

- GATHER_INDEX_STATS
- GATHER_TABLE_STATS
- GATHER_SCHEMA_STATS
- GATHER_DICTIONARY_STATS
- GATHER_DATABASE_STATS
- GATHER_SYSTEM_STATS



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

When you manually gather optimizer statistics, you must pay special attention to the following factors:

- Monitoring objects for mass DML operations and gathering statistics if necessary
- Determining the correct sample sizes
- Determining the degree of parallelism to speed up queries on large objects
- Determining if histograms should be created on columns with skewed data
- Determining whether changes on objects cascade to any dependent indexes

You can use the following procedures to gather statistics:

- GATHER_TABLE_STATS to collect table, column, and index statistics
- GATHER_SCHEMA_STATS to collect statistics for all objects in a schema
- GATHER_INDEX_STATS to collect index statistics
- GATHER_SCHEMA_STATS to collect statistics for all objects in a schema
- GATHER_DICTIONARY_STATS to collect statistics for all system schemas, including SYS and SYSTEM, and other optional schemas
- GATHER_DATABASE_STATS to collect statistics for all objects in a database

Gathering Object Statistics: Example

```
dbms_stats.gather_table_stats
('sh'                      -- schema
,'customers'                -- table
, null                     -- partition
, 20                       -- sample size(%)
, false                    -- block sample?
,'for all columns'         -- column spec
, 4                        -- degree of parallelism
,'default'                  -- granularity
, true );                  -- cascade to indexes
```

```
dbms_stats.set_param('CASCADE',
                      'DBMS_STATS.AUTO.Cascade');
dbms_stats.set_param('ESTIMATE_PERCENT', '5');
dbms_stats.set_param('DEGREE', 'NULL');
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The first example in the slide uses the DBMS_STATS package to gather statistics on the CUSTOMERS table of the SH schema. It uses some of the options discussed in the preceding slides.

You can use the SET_PARAM procedure in DBMS_STATS to set default values for parameters of all DBMS_STATS procedures. The second example in the slide shows this usage. You can also use the GET_PARAM function to get the current default value of a parameter.

Note: Granularity of statistics to collect is pertinent only if the table is partitioned. This parameter determines at which level statistics should be gathered. This can be at the partition, subpartition, or table level.

Object Statistics: Best Practices

- Ensure that all objects (tables and indexes) have statistics gathered.
- Use a sample size that is large enough if feasible.
- Gather optimizer statistics during periods of low activity.
- If partitions are in use, gather global statistics if possible.
- Use Oracle Database pending statistics to verify the effect of new statistics when tuning to minimize risk.
- Gather statistics after data has been loaded (>10% added), but before indexes are created.



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Review the following best practices:

- Ensure that all objects have statistics gathered. An easy way to achieve this is to use the `CASCADE` parameter. Note that index statistics are automatically gathered when the index is created or rebuilt.
- Use 100 percent sample size if possible. In Oracle Database 11g, Oracle gathers optimizer statistics with a new algorithm. It is very accurate, but requires minimal time. However, for very large systems, the gathering of statistics can be a very time-consuming and resource-intensive activity. In this environment, sample sizes need to be carefully controlled to ensure that gathering is completed within an acceptable timescale, within resource constraints, and within the maintenance window. If the 100 percent sample size is not feasible, try using at least an estimate of 30 percent. For more information, review MOS note 44961.1, “Statistics Gathering: Frequency and Strategy Guidelines.”
- Because gathering new optimizer statistics may invalidate cursors in the shared pool, it is prudent to restrict execution of all gathering operations to periods of low activity in the database, such as scheduled maintenance windows.

Lesson Agenda

- Gathering Statistics
- **Dynamic Statistics**
- Automatic Re-optimization
- SQL Plan Directives
- Online Statistics Gathering for Bulk Loads
- Concurrent Statistics Gathering
- Gathering System Statistics
- Managing Optimizer Statistics



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Dynamic Statistics: Overview

- Dynamic statistics replaces dynamic sampling.
- Dynamic statistics are used to compensate for missing, stale, or incomplete statistics.
- They can be used for table scans, index access, and joins.
- The optimizer computes a time budget for generating dynamic statistics based on query run time.
- Statistics are stored in memory and can be shared across queries.
- They are used to determine more accurate statistics when estimating:
 - Table cardinality
 - Predicate selectivity



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

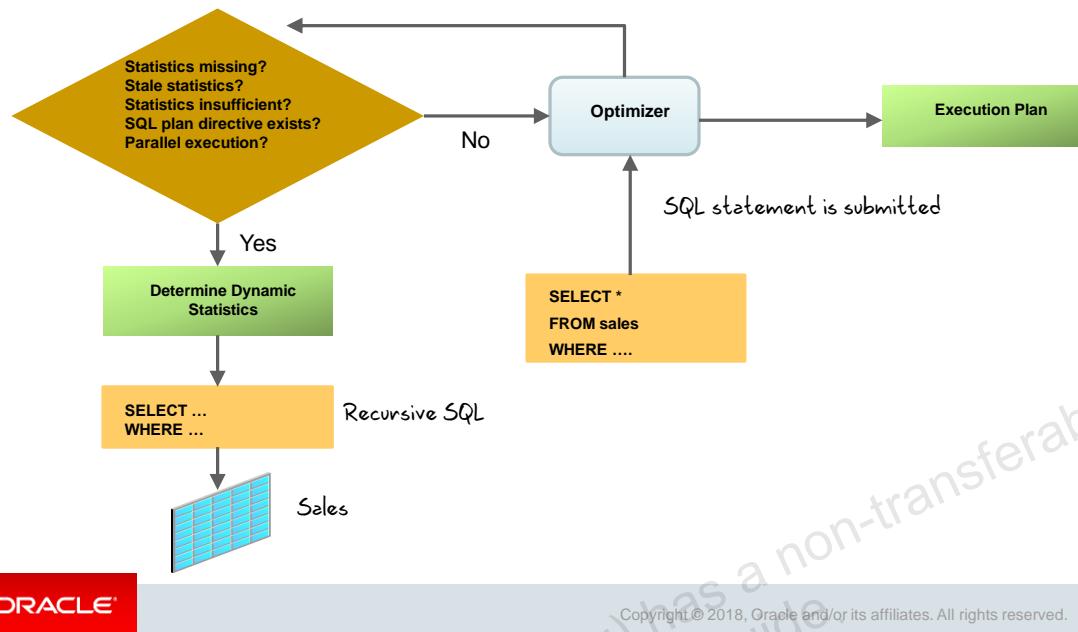
Dynamic statistics (previously called dynamic sampling) improves server performance by determining more accurate selectivity and cardinality estimates that allow the optimizer to produce better performing plans. For example, although it is recommended that you collect statistics on all of your tables for use by the optimizer, you may not gather statistics for your temporary tables and working tables that are used for intermediate data manipulation. In those cases, the optimizer provides a value through a simple algorithm that can lead to a suboptimal execution plan. The decision on whether to use dynamic statistics depends on the query, existing base statistics, and total execution time.

In earlier releases, dynamic statistics were used only when one or more of the tables in a query did not have optimizer statistics. Starting in this release, the optimizer automatically decides whether dynamic statistics are useful and which dynamic statistics level to use for all SQL statements.

Dynamic statistics are beneficial in the following situations:

- An execution plan is suboptimal because of complex predicates.
- The sampling time is a small fraction of total execution time for the query.
- The query is executed many times so that the sampling time is amortized.

Dynamic Statistics at Work



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Sampling is done at compile time.

Automatic dynamic statistics are enabled when any of the following conditions is true:

- The `OPTIMIZER_DYNAMIC_SAMPLING` initialization parameter uses its default value, which means that it is not explicitly set.
- The `OPTIMIZER_DYNAMIC_SAMPLING` initialization parameter is set to 11.
- You invoke the gathering of dynamic statistics through a SQL hint.

As shown in the figure, the optimizer automatically performs dynamic statistics in the following cases:

Missing Statistics

When tables in a query have no statistics, the optimizer gathers basic statistics on these tables before optimization. Statistics can be missing because new objects are created by the application without a follow-up call to `DBMS_STATS` to gather statistics, or because statistics were locked on an object before statistics were gathered.

In this case, the statistics are not as high-quality or as complete as the statistics gathered by using the `DBMS_STATS` package. This trade-off is made to limit the impact on the compile time of the statement.

Stale Statistics

Statistics gathered by DBMS_STATS can become out-of-date. Typically, statistics are stale when 10 percent or more of the rows in the table have changed since the last time that statistics were gathered.

Insufficient Statistics

Statistics can be insufficient when the optimizer estimates the selectivity of predicates (filter or join) or the GROUP BY clause without taking into account the correlation between columns, skew in the column data distribution, and so on.

In the figure, the database issues a recursive SQL statement to scan a small random sample of the table blocks. The database applies the relevant single-table predicates and joins to estimate predicate selectivities.

The database persists the results of dynamic statistics as shareable statistics. The database can share the results during the SQL compilation of one query with recompilations of the same query. The database can also reuse the results for queries that have the same patterns. If no rows were inserted, deleted, or updated in the table being sampled, dynamic statistics is repeatable.

At the beginning of optimization, when deciding whether a table is a candidate for dynamic statistics, the optimizer checks for the existence of persistent SQL plan directives on the table (see Figure 10-2). For each directive, the optimizer registers a statistics expression that the optimizer computes when it must determine the selectivity of a predicate involving the table.

When sampling is necessary, the database must determine the sample size. Starting in Oracle Database 12c Release 1 (12.1), if the OPTIMIZER_DYNAMIC_SAMPLING initialization parameter is not explicitly set to a value other than 11, then the optimizer automatically decides whether to use dynamic statistics and which level to use. The database issues a recursive SQL statement to scan a small random sample of the table blocks. The database applies the relevant single-table predicates and joins to estimate predicate selectivities.

The database persists the results of dynamic statistics as sharable statistics. The database can share the results during the SQL compilation of one query with recompilations of the same query. The database can also reuse the results for queries that have the same patterns. If no rows have been inserted, deleted, or updated in the table being sampled, then the use of dynamic statistics is repeatable.

You control dynamic statistics with the OPTIMIZER_DYNAMIC_SAMPLING initialization parameter. The DYNAMIC_SAMPLING and DYNAMIC_SAMPLING_EST_CDN hints can be used to further control dynamic statistics.

OPTIMIZER_DYNAMIC_SAMPLING

- Dynamic session or system parameter can be set to a value from 0 to 10.
 - 0 turns off dynamic statistics.
 - 1 samples all tables that do not have statistics.
 - 2 samples at least one table in the statement that has no statistics.
- Dynamic statistics is repeatable if no update activity occurred.
- You can disable the feature by setting the `OPTIMIZER_DYNAMIC_SAMPLING` initialization parameter to 0.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

You control dynamic statistics level with the `OPTIMIZER_DYNAMIC_SAMPLING` parameter, which can be set to a value from 0 to 10. A value of 0 means collection of dynamic statistics is not done. A value of 1 means dynamic statistics is performed on all unanalyzed tables if the following criteria are met:

- There is at least 1 nonpartitioned table in the query that does not have statistics.
- This table has no indexes.
- This table has more blocks than the number of blocks that would be used for dynamic statistics of this table.

The default value is 2 if `OPTIMIZER_FEATURES_ENABLE` is set to 10.0.0 or higher. Use dynamic statistics if at least one table in the statement has no statistics. The number of blocks sampled is two times the default number of dynamic statistics blocks (64).

Increasing the value of the parameter results in a more aggressive application of dynamic statistics, in terms of both the type of tables sampled (analyzed or unanalyzed) and the amount of I/O spent on sampling.

Note: Dynamic statistics is repeatable if no rows have been inserted, deleted, or updated in the table being sampled since the previous sample operation.

Lesson Agenda

- Gathering Statistics
- Dynamic Statistics
- **Automatic Re-optimization**
- SQL Plan Directives
- Online Statistics Gathering for Bulk Loads
- Concurrent Statistics Gathering
- Gathering System Statistics
- Managing Optimizer Statistics



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

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© 2018, Oracle and/or its affiliates. All rights reserved.

In contrast to adaptive plans, automatic re-optimization changes a plan on subsequent executions after the initial execution. At the end of the first execution of a SQL statement, the optimizer uses the information gathered during the initial execution to determine whether automatic re-optimization is worthwhile. If the execution information differs significantly from the optimizer's original estimates, then the optimizer looks for a replacement plan on the next execution. The optimizer uses the information gathered during the previous execution to help determine an alternative plan. The optimizer can re-optimize a query several times, each time learning more and further improving the plan. Oracle Database 12c supports multiple forms of re-optimization.

As of Oracle Database 12c, join statistics are also included. Statements are continually monitored to verify if statistics fluctuate over different executions. It 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.

Re-optimization: Statistics Feedback

- Statistics feedback was introduced in Oracle Database 11g, Release 2.
- Statistics feedback is useful for queries where the data volume being processed is stable over time.
- During query execution, optimizer estimates are compared to execution statistics: if they vary significantly, then a new plan will be chosen for subsequent executions.



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Statistics feedback (formerly called 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 repeated queries that have cardinality misestimates. The optimizer can estimate cardinalities incorrectly for many reasons, such as missing statistics, 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.

Statistics Feedback: Monitoring Query Executions

```
select /*+gather_plan_statistics*/ c.cust_first_name, c.cust_last_name,
sum(s.amount_sold)
from customers c, sales s
where c.cust_id=s.cust_id
and c.cust_city='Los Angeles'
and c.cust_state_province='CA'
and c.country_id=52790
and s.time_id='09-NOV-00'
group by c.cust_first_name, c.cust_last_name;
```

	Id	Operation	Name	Starts	E-Rows	A-Rows
	0	SELECT STATEMENT		1	1	0
*	1	HASH GROUP BY		1	1	0
*	2	HASH JOIN		1	1	0
	3	PARTITION RANGE SINGLE		1	40	0
*	4	TABLE ACCESS FULL	SALES	1	5	40
*	5	TABLE ACCESS FULL	CUSTOMERS	1	1	13

Initial cardinality estimate is more than 8X off.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Execution of the query given in the code box is monitored. The example shows that the initial plan is a hash join between sales and customers.

The query has a combination of an equality filter predicate and an inequality filter predicate, which causes a misestimation in the cardinality of the `customers` 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 `customers` is underestimated.

Statistics Feedback: Reparsing Statements

```
select /*+gather_plan_statistics*/ c.cust_first_name, c.cust_last_name,
sum(s.amount_sold)
from customers c, sales s
where c.cust_id=s.cust_id
and c.cust_city='Los Angeles'
and c.cust_state_province='CA'
and c.country_id=52790
and s.time_id='09-NOV-00'
group by c.cust_first_name, c.cust_last_name;
```

I	I Operation	I Name	I Starts	I E-Rows	I A-Rows	I
I 0	SELECT STATEMENT		1	1	0	
I 1	HASH GROUP BY		1	1	0	
I* 2	HASH JOIN		1	1	0	
I* 3	TABLE ACCESS FULL	CUSTOMERS	1	13	13	
I* 4	PARTITION RANGE SINGLE		1	40	40	
I* 5	TABLE ACCESS FULL	SALES	1	40	40	

Predicate Information (identified by operation id):

```

2 - access("C"."CUST_ID"="S"."CUST_ID")
3 - filter(("C"."CUST_CITY"='Los Angeles' AND "C"."CUST_STATE_PROVINCE"
5 - filter("S"."TIME_ID"='09-NOV-00'))
```

Note

- statistics feedback used for this statement

Estimates are now correct.

Statistics feedback appears in the notes section of the plan.



Copyright © 2018, 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 it displays a different plan. The new plan shows correct cardinality estimates and a new join order. Information learned is stored in the cursor only and is lost if cursor ages out.

Lesson Agenda

- Gathering Statistics
- Dynamic Statistics
- Automatic Re-optimization
- **SQL Plan Directives**
- Online Statistics Gathering for Bulk Loads
- Concurrent Statistics Gathering
- Gathering System Statistics
- Managing Optimizer Statistics



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

SQL Plan Directives

- A SQL plan directive is additional information and instructions that the optimizer can use to generate a better plan:
 - Collect missing statistics.
 - Create column group statistics.
 - Perform dynamic statistics.
- Directives can be used for multiple statements:
 - Directives are collected on query expressions.
- They are persisted to disk in the SYSAUX tablespace.
- Directives are automatically maintained:
 - Created as needed during compilation or execution:
 - Missing statistics, cardinality misestimates
 - Purged if not used after a year



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

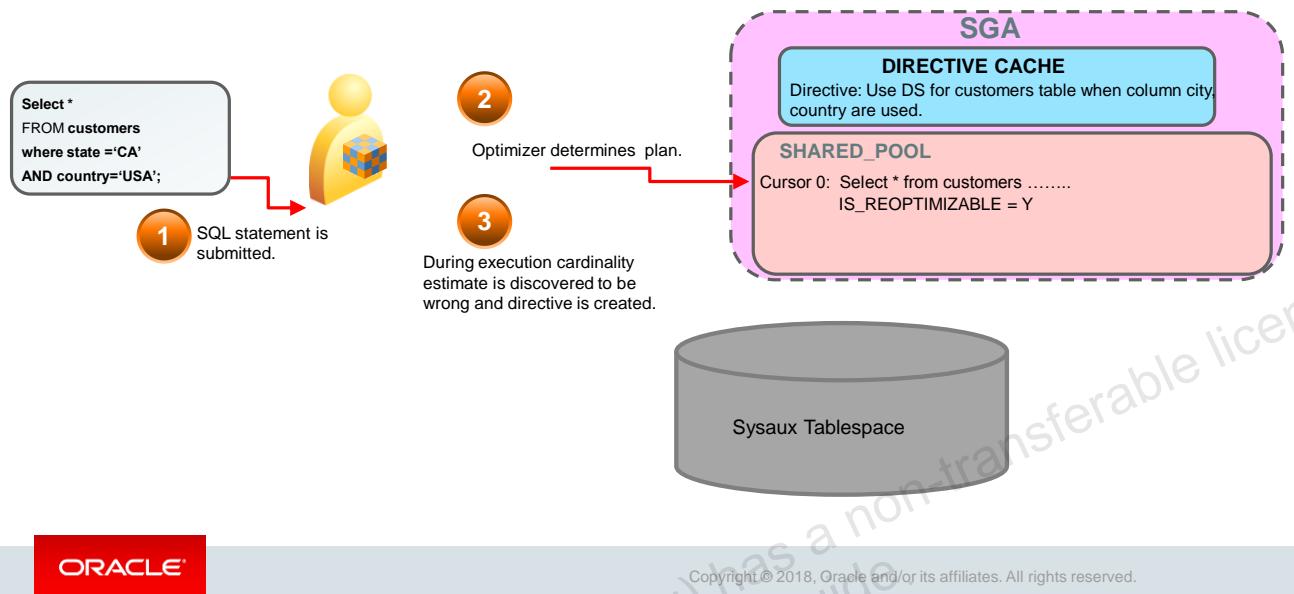
In previous releases, the database stored compilation and execution statistics in the shared SQL area, which is non-persistent. Starting in Oracle Database 12c, the database can use a SQL plan directive, which is additional information and instructions that the optimizer can use to generate a more optimal plan. For example, a SQL plan directive might instruct the optimizer to collect missing statistics, create column group statistics, or perform dynamic statistics. During SQL compilation or execution, the database analyzes the query expressions that are missing statistics or that misestimate optimizer cardinality to create SQL plan directives. When the optimizer generates an execution plan, the directives give the optimizer additional information about objects that are referenced in the plan.

SQL plan directives are not tied to a specific SQL statement or `SQL_ID`. The optimizer can use SQL plan directives for SQL statements that are nearly identical because SQL plan directives are defined on a query expression. For example, directives can help the optimizer with queries that use similar patterns, such as web-based queries that are the same except for a select list item. The database stores SQL plan directives persistently in the SYSAUX tablespace. When generating an execution plan, the optimizer can use SQL plan directives to obtain more information about the objects that are accessed in the plan.

Directives are automatically maintained, created as needed, and purged if not used after a year.

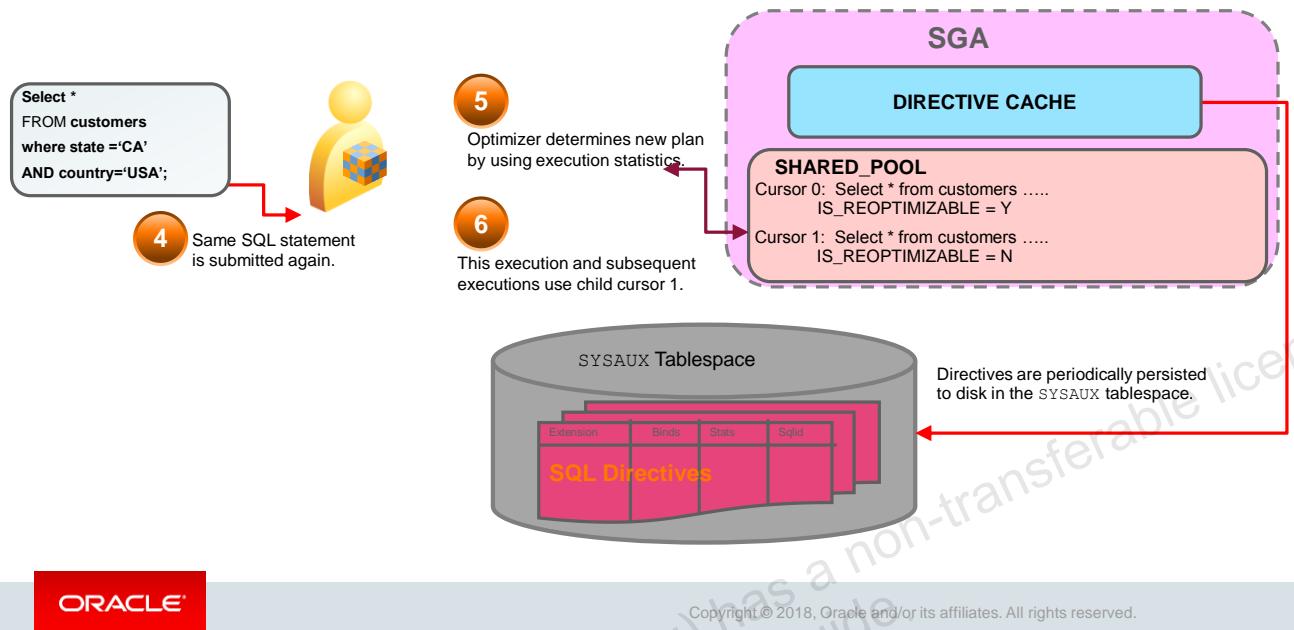
Directives can be monitored in `DBA_SQL_PLAN_DIR_OBJECTS`. SQL plan directives improve plan accuracy by persisting both compilation and execution statistics in the SYSAUX tablespace, allowing them to be used by multiple SQL statements.

SQL Plan Directives



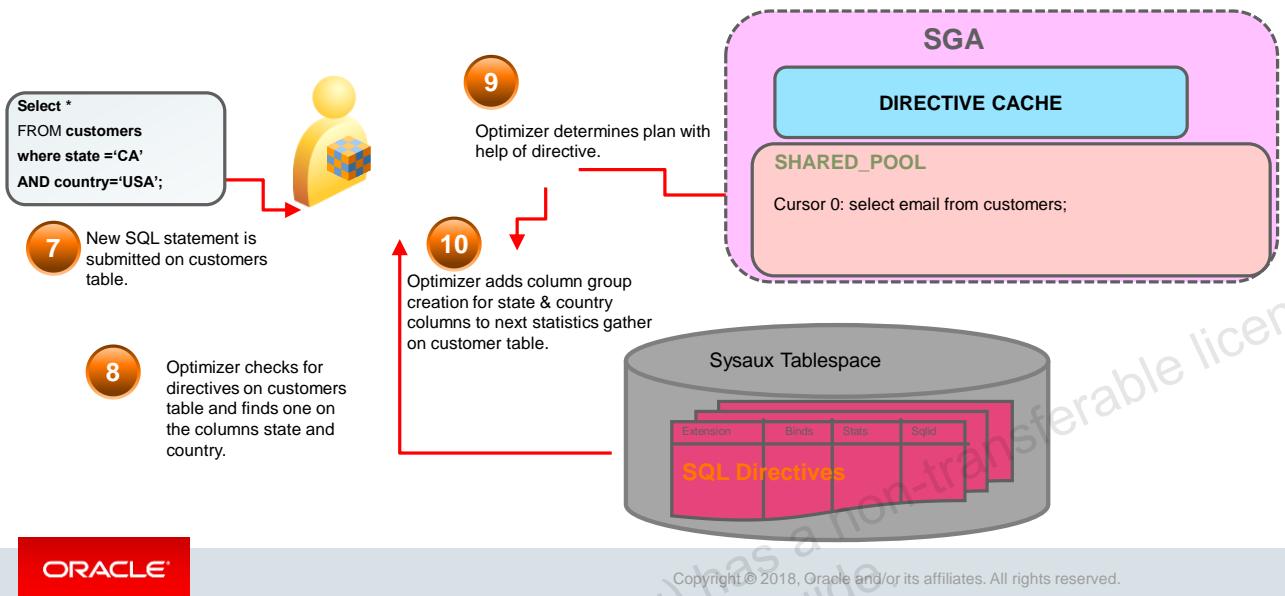
The slide shows how SQL directives are created. The SQL statement is submitted, and then the optimizer determines the plan. During execution, if the cardinality estimates are not correct, then the directives are created.

SQL Plan Directives



The database initially creates directives in the shared pool. The database periodically writes the directives to the SYSAUX tablespace. When the same SQL statement is resubmitted, the optimizer determines the new plans by using execution statistics. The execution uses child cursor 1.

SQL Plan Directives



The SQL statement on the `CUSTOMERS` table is submitted again. This time, the optimizer checks for directives on the `CUSTOMERS` table and finds one on the `state` and `country` columns. After the optimizer gets the directive, it determines the plan with the help of the directive. The optimizer adds columns group creation for the `state` and `country` columns to the next statistics gather on the `CUSTOMERS` table.

SQL Plan Directives: Example

```
B.1. Manually Flush Directives (auto flush done every 15 minutes)
=====
SQL> exec dbms_spd.flush_sql_plan_directive
PL/SQL procedure successfully completed.

B.2. [Display Directives]
=====

SQL> col state format a5
SQL> col subobject_name format a11
SQL> col object_name format a13
SQL> select object_name, SUBOBJECT_NAME, type, state, reason from dba_sql_plan_directives d,
and object_name in ('ORDER_ITEMS', 'PRODUCT_INFORMATION');

OBJECT_NAME    SUBOBJECT_N TYPE      STATE REASON
-----          -----      ---      ---   -----
ORDER_ITEMS    UNIT_PRICE  DYNAMIC_SAMPLING NEW    SINGLE TABLE CARDINALITY MISESTIMATE
ORDER_ITEMS    QUANTITY    DYNAMIC_SAMPLING NEW    SINGLE TABLE CARDINALITY MISESTIMATE
ORDER_ITEMS          DYNAMIC SAMPLING NEW    SINGLE TABLE CARDINALITY MISESTIMATE
```

B.3. Use Directives

```
=====
SQL> explain plan for
select /*Q10*/ product_name
from order_items o, product_information p
where o.unit_price = 15
and quantity > 1
and p.product_id = o.product_id;
Explained.

SQL> select * from table(dbms_xplan.display(format=>'basic +note'));
PLAN_TABLE_OUTPUT
Plan hash value: 2615131494
| Id | Operation           | Name        |
|---|:-----|:-----|
| 0 | SELECT STATEMENT   |            |
| 1 |  HASH JOIN          |            |
| 2 |   TABLE ACCESS FULL| ORDER_ITEMS |
| 3 |   TABLE ACCESS FULL| PRODUCT_INFORMATION |

Note
-----
- dynamic sampling used for this statement (level=2)
- 1 Sql Plan Directive used for this statement
```



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

SQL plan directives are created on query expressions rather than at a statement or object level to ensure they can be applied to multiple SQL statements. It is also possible to have multiple SQL plan directives used for a SQL statement.

This slide shows SQL directive examples.

You can monitor SQL plan directives by using the `DBA_SQL_PLAN_DIRECTIVES` and `DBA_SQL_PLAN_DIR_OBJECTS` views.

The Notes section of the plan display functionality shows you information about SQL plan directives such as the SQL plan directive used for this statement.

Lesson Agenda

- Gathering Statistics
- Dynamic Statistics
- Automatic Re-optimization
- SQL Plan Directives
- **Online Statistics Gathering for Bulk Loads**
- Concurrent Statistics Gathering
- Gathering System Statistics
- Managing Optimizer Statistics



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Online Statistics Gathering for Bulk Loads

- A bulk load is a `CREATE TABLE AS SELECT` or `INSERT INTO ... SELECT` operation.
- In Oracle Database 12c, the database automatically gathers table statistics during the following types of bulk loads:
 - `CREATE TABLE AS SELECT`
 - `INSERT INTO ... SELECT` into an empty table by using a direct path insert
 - `INSERT INTO ... SELECT` into a non-empty table, partition, or subpartition
- Statistics are available right away after a load.
- No additional table scan is required to gather statistics.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

With online statistics gathering, statistics are automatically created as part of a bulk load operation such as a `CREATE TABLE AS SELECT` operation or an `INSERT INTO ... SELECT` operation on an empty table.

Online statistics gathering eliminates the need to manually gather statistics after a bulk data load has occurred. It behaves in a similar manner to the statistics gathering done during a `CREATE INDEX` or `REBUILD INDEX` command. In a data warehouse, you frequently need to load a large amount of data into the database. For example, a sales record data warehouse might load sales data nightly, where online statistics gathering is useful.

Oracle Database 12c now gathers the statistics automatically, and the principal benefits are improved performance and manageability of bulk load operations by eliminating user intervention to gather statistics after the load. It also avoids additional full table scans, required for separate statistics gathering operations.

While gathering online statistics, the database does not gather index statistics or histograms. If these statistics are required, Oracle recommends running DBMS_STATS.GATHER_TABLE_STATS with the options parameter set to GATHER AUTO after the bulk load. In this case, GATHER_TABLE_STATS gathers only missing or stale statistics.

Thus, the database gathers only index statistics or histograms, and not table and basic column statistics collected during the bulk load. You can set the table preference options to GATHER AUTO on the tables that will have a bulk load. In this way, you need not explicitly set the options parameter when running GATHER_TABLE_STATS.

By default, the database gathers statistics during bulk loads. You can disable the feature at the statement level by using the NO_GATHER_OPTIMIZER_STATISTICS hint. You can also explicitly enable the feature at the statement level by using the GATHER_OPTIMIZER_STATISTICS hint.

Currently, statistics gathering does not happen for bulk load statements when any of the following conditions apply to the target table:

- It is in an Oracle-owned schema such as SYS.
- It is a nested table.
- It is an index-organized table (IOT).
- It is an external table.
- It is a global temporary table defined as ON COMMIT DELETE ROWS.
- It has virtual columns.
- It has a PUBLISH preference set to FALSE.
- It is partitioned, INCREMENTAL is set to true, and extended syntax is not used.

For example, if you issued

```
DBMS_STATS.SET_TABLE_PREFS(null,'sales', 'incremental','true'),  
then the database does not gather statistics for INSERT INTO sales SELECT, but it can  
gather statistics for INSERT INTO sales PARTITION (sales_q4_2000) SELECT.
```

Lesson Agenda

- Gathering Statistics
- Dynamic Statistics
- Automatic Re-optimization
- SQL Plan Directives
- Online Statistics Gathering for Bulk Loads
- **Concurrent Statistics Gathering**
- Gathering System Statistics
- Managing Optimizer Statistics



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Concurrent Statistics Gathering

- You can use concurrent statistics gathering mode to simultaneously gather optimizer statistics for multiple tables in a schema and multiple partitions or subpartitions in a table.
- You can enable concurrent statistics gathering by setting the `CONCURRENT` preference with `DBMS_STATS.SET_GLOBAL_PREF`.
- You can use the following tools to create and manage multiple statistics gathering jobs concurrently:
 - Oracle Database Scheduler
 - Advanced Queuing
 - Oracle Database Resource Manager



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

In Oracle Database 11.2.0.2, concurrent statistics gathering was introduced. When the global statistics gathering preference `CONCURRENT` is set, Oracle employs the Oracle Job Scheduler and Advanced Queuing components to create and manage one statistics gathering job per object (tables and/or partitions) concurrently.

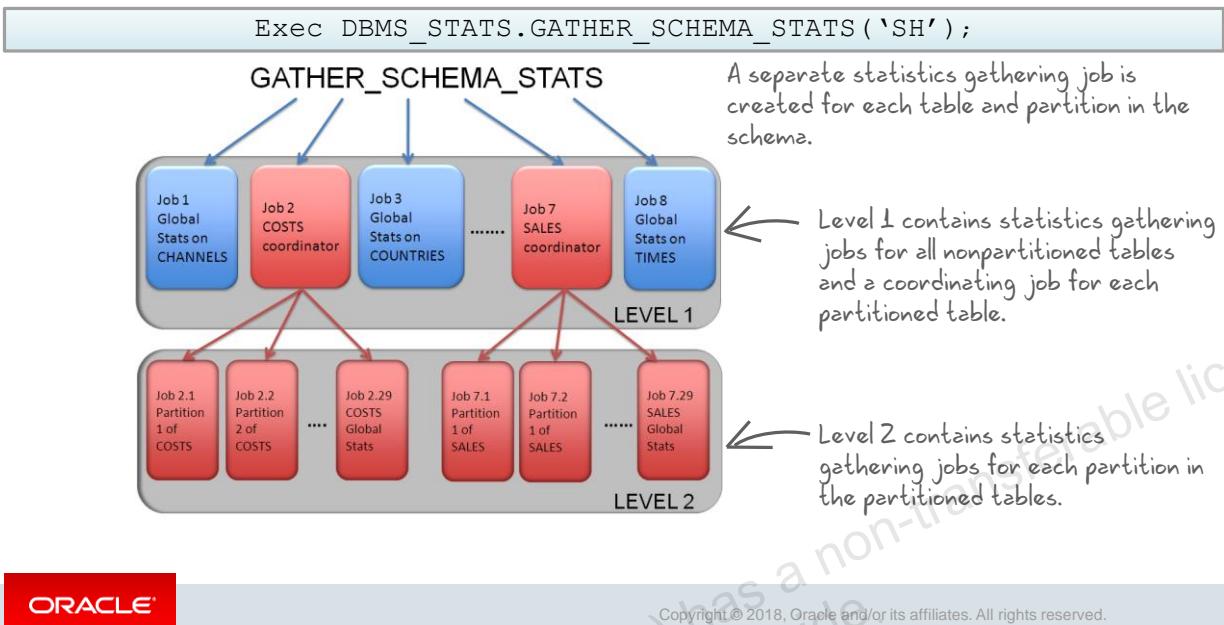
In Oracle Database 12c, concurrent statistics gathering has been enhanced to make better use of each scheduler job. If a table, partition, or subpartition is very small or empty, the database may automatically batch the object with other small objects into a single job to reduce the overhead of job maintenance.

Concurrent statistics gathering can now be employed by the nightly statistics gathering job by setting the preference `CONCURRENT` to `ALL` or `AUTOMATIC`. The new `ORA$AUTOTASK` consumer group has been added to the Resource Manager plan active used during the maintenance window, to ensure concurrent statistics gathering does not use too many system resources. Concurrent statistics gathering mode enables the database to simultaneously gather optimizer statistics for multiple tables in a schema, table partitions, or table subpartitions.

The Scheduler decides how many jobs to execute concurrently, and how many will be queued based on available system resources. The number of concurrent gather operations is controlled by the `job_queue_processes` parameter.

The `DBMS_STATS` package does not explicitly manage resources used by concurrent statistics gathering jobs that are part of a user-initiated statistics gathering call. Thus, the database may use system resources fully during concurrent statistics gathering. To address this situation, you can use the Resource Manager to cap resources consumed by concurrent statistics gathering jobs. The Resource Manager must be enabled to gather statistics concurrently.

Concurrent Statistics Gathering: Creating Jobs at Different Levels



ORACLE

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The database runs as many concurrent jobs as possible. The Job Scheduler decides how many jobs to execute concurrently and how many to queue. As running jobs complete, the Scheduler dequeues and runs more jobs until the database has gathered statistics on all tables, partitions, and subpartitions. The maximum number of jobs is bounded by the JOB_QUEUE_PROCESSES initialization parameter and available system resources.

In most cases, the DBMS_STATS procedures create a separate job for each table partition or subpartition. However, if the partition or subpartition is very small or empty, the database may automatically batch the object with other small objects into a single job to reduce the overhead of job maintenance.

The figure in the slide shows the creation of jobs at different levels in the SH schema, where the COSTS and SALES tables are partitioned and other tables are nonpartitioned. Job 2 acts as a coordinator job for the COSTS table and job 7 acts as a coordinator job for the SALES table. The respective job creates a job for each partition in the tables, as well as a separate job for the global statistics of the COSTS and SALES tables.

Lesson Agenda

- Gathering Statistics
- Dynamic Statistics
- Automatic Re-optimization
- SQL Plan Directives
- Online Statistics Gathering for Bulk Loads
- Concurrent Statistics Gathering
- **Gathering System Statistics**
- Managing Optimizer Statistics



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Gathering System Statistics: Automatic Collection – Example

First day

```
EXECUTE DBMS_STATS.GATHER_SYSTEM_STATS(
interval => 120,
stattab => 'mystats', statid => 'OLTP');
```

First night

```
EXECUTE DBMS_STATS.GATHER_SYSTEM_STATS(
interval => 120,
stattab => 'mystats', statid => 'OLAP');
```

Next days

```
EXECUTE DBMS_STATS.IMPORT_SYSTEM_STATS(
stattab => 'mystats', statid => 'OLTP');
```

Next nights

```
EXECUTE DBMS_STATS.IMPORT_SYSTEM_STATS(
stattab => 'mystats', statid => 'OLAP');
```



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

The example in the slide shows database applications processing OLTP transactions during the day and running reports at night.

First, system statistics must be collected during the day. In this example, gathering ends after 120 minutes and statistics are stored in the `mystats` table.

Then, system statistics are collected during the night. Gathering ends after 120 minutes and statistics are stored in the `mystats` table.

Generally, the syntax in the slide is used to gather system statistics. Before invoking the `GATHER_SYSTEM_STATS` procedure with the `INTERVAL` parameter specified, you must activate job processes by using a command such as `SQL> alter system set job_queue_processes = 1;`.

Note: The default value of `job_queue_processes` is 1000. You can also invoke the same procedure with different arguments to enable manual gathering instead of using jobs.

If appropriate, you can switch between the statistics gathered. Note that it is possible to automate this process by submitting a job to update the dictionary with appropriate statistics. During the day, a job may import the OLTP statistics for the daytime run, and during the night, another job imports the OLAP statistics for the nighttime run.

Gathering System Statistics: Manual Collection – Example

- Start manual system statistics collection in the data dictionary:

```
EXECUTE DBMS_STATS.GATHER_SYSTEM_STATS( -  
gathering_mode => 'START');
```

- Generate the workload.
- End the collection of system statistics:

```
EXECUTE DBMS_STATS.GATHER_SYSTEM_STATS( -  
gathering_mode => 'STOP');
```



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The example in the previous slide shows how to collect system statistics with jobs by using the internal parameter of the `DBMS_STATS.GATHER_SYSTEM_STATS` procedure. To collect system statistics manually, another parameter of this procedure can be used, as shown in the slide.

First, you must start system statistics collection, and then you can end the collection process at any time after you are certain that a representative workload has been generated on the instance.

The example collects system statistics and stores them directly in the data dictionary.

Lesson Agenda

- Gathering Statistics
- Dynamic Statistics
- Automatic Re-optimization
- SQL Plan Directives
- Online Statistics Gathering for Bulk Loads
- Concurrent Statistics Gathering
- Gathering System Statistics
- Managing Optimizer Statistics



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Managing Statistics: Overview (Export / Import / Lock / Restore / Publish)

- Purpose:
 - To revert to pre-analyzed statistics if gathering statistics causes critical statements to perform badly
 - To test the new statistics before publishing
- Importing previously exported statistics (9*i*)
- Locking and unlocking statistics on a specific table (10*g*)
- Restoring statistics archived before gathering (10*g*)
- Statistics can be pending before publishing (11*g R2*).



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

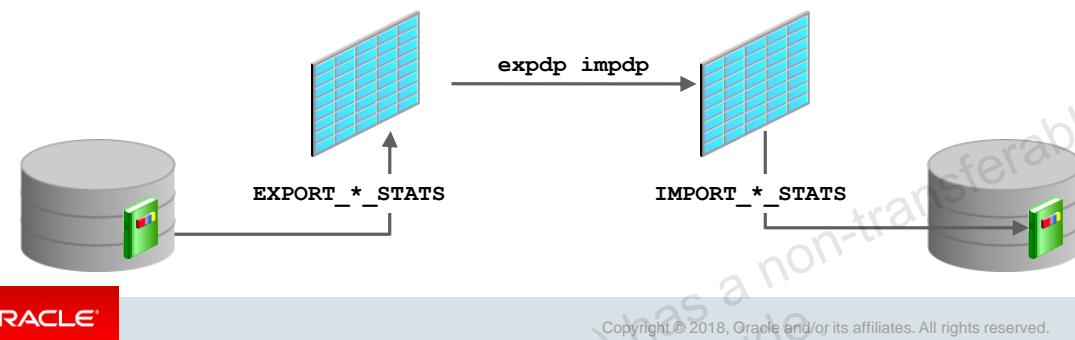
In subsequent slides, the following topics are covered to discuss how to manage optimizer statistics:

- Importing previously exported statistics (9*i*)
- Locking and unlocking statistics on a specific table (10*g*)
- Restoring statistics archived before gathering (10*g*)
- Statistics can be pending before publishing (11*g R2*).

Exporting and Importing Statistics

Use DBMS_STATS procedures:

- CREATE_STAT_TABLE creates the statistics table.
- EXPORT_*_STATS moves the statistics to the statistics table.
- Use Data Pump to move the statistics table.
- IMPORT_*_STATS moves the statistics to the data dictionary.



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

You can export and import statistics from the data dictionary to user-owned tables, enabling you to create multiple versions of statistics for the same schema. You can also copy statistics from one database to another database. You may want to do this to copy the statistics from a production database to a scaled-down test database.

Before exporting statistics, you first need to create a table for holding the statistics. The procedure DBMS_STATS.CREATE_STAT_TABLE creates the statistics table. After table creation, you can export statistics from the data dictionary into the statistics table by using the DBMS_STATS.EXPORT_*_STATS procedures. You can then import statistics by using the DBMS_STATS.IMPORT_*_STATS procedures.

The optimizer does not use statistics stored in a user-owned table. The only statistics used by the optimizer are the statistics stored in the data dictionary. To have the optimizer use the statistics in a user-owned table, you must import those statistics into the data dictionary by using the statistics-import procedures.

To move statistics from one database to another, you must first export the statistics on the first database, then copy the statistics table to the second database by using the Data Pump Export and Import utilities or other mechanisms, and then import the statistics into the second database.

Locking and Unlocking Statistics

- Prevents automatic gathering
- Is mainly used for volatile tables:
 - Lock without statistics implies dynamic statistics.

```
BEGIN
  DBMS_STATS.DELETE_TABLE_STATS('OE','ORDERS');
  DBMS_STATS.LOCK_TABLE_STATS('OE','ORDERS');
END;
```

- Lock with statistics for representative values.

```
BEGIN
  DBMS_STATS.GATHER_TABLE_STATS('OE','ORDERS');
  DBMS_STATS.LOCK_TABLE_STATS('OE','ORDERS');
END;
```

- Unlock the statistics on oe.orders.

```
BEGIN
  DBMS_STATS.UNLOCK_TABLE_STATS('OE','ORDERS');
END; /
```



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Starting with Oracle Database 10g, you can lock statistics on a specified table with the `LOCK_TABLE_STATS` procedure of the `DBMS_STATS` package. You can lock statistics to prevent them from changing. After statistics are locked, you cannot make modifications to the statistics until the statistics have been unlocked.

You can lock statistics on a table without statistics or set them to `NULL` by using the `DELETE_*_STATS` procedures to prevent automatic statistics collection so that you can use dynamic statistics on a volatile table with no statistics. You can also lock statistics on a volatile table at a point when it is fully populated so that the table statistics are more representative of the table population.

You can also lock statistics at the schema level by using the `LOCK_SCHEMA_STATS` procedure. You can query the `STATTYPE_LOCKED` column in the `{USER | ALL | DBA}_TAB_STATISTICS` view to determine whether the statistics on the table are locked.

You can use the `UNLOCK_SCHEMA_STAT` and `UNLOCK_TABLE_STATS` procedures to unlock the statistics on a specified table.

You can set the value of the `FORCE` parameter to `TRUE` to overwrite the statistics even if they are locked. The `FORCE` argument is found in the following `DBMS_STATS` procedures: `DELETE_*_STATS`, `IMPORT_*_STATS`, `RESTORE_*_STATS`, and `SET_*_STATS`.

Note: When you lock the statistics on a table, all the dependent statistics, including table statistics, column statistics, histograms, and dependent index statistics, are considered locked.

Restoring Statistics

- Past statistics may be restored with the DBMS_STATS.RESTORE_*_STATS procedures.

```
BEGIN
  DBMS_STATS.RESTORE_TABLE_STATS(
    OWNNAME=>'OE',
    TABNAME=>'INVENTORIES',
    AS_OF_TIMESTAMP=>'15-JUL-10 09.28.01.597526000 AM -05:00');
END;
```

- Statistics are automatically stored:
 - With a time stamp in DBA_TAB_STATS_HISTORY
 - When collected with DBMS_STATS procedures
- Statistics are purged:
 - When the STATISTICS_LEVEL is set to TYPICAL or ALL automatically
 - After 31 days or time defined by DBMS_STATS.ALTER_STATS_HISTORY_RETENTION



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Old versions of statistics are saved automatically whenever statistics in the dictionary are modified with the DBMS_STATS procedures. You can restore statistics by using the RESTORE procedures of the DBMS_STATS package. These procedures use a time stamp as an argument and restore statistics as of that time stamp. This is useful when newly-collected statistics lead to suboptimal execution plans and the administrator wants to revert to the previous set of statistics.

Guidelines for Restoring Optimizer Statistics

Restoring statistics is similar to importing and exporting statistics. In general, restore statistics instead of exporting them in the following situations:

- You want to recover older versions of the statistics. For example, you want to restore the optimizer behavior to an earlier date.
- You want the database to manage the retention and purging of statistics histories.

Export statistics rather than restoring them in the following situations:

- You want to experiment with multiple sets of statistics and change the values back and forth.
- You want to move the statistics from one database to another database; for example, moving statistics from a production system to a test system.
- You want to preserve a known set of statistics for a longer period than the desired retention date for restoring statistics.

Restrictions for Restoring Optimizer Statistics

When restoring previous versions of statistics, the following limitations apply:

- DBMS_STATS.RESTORE_*_STATS procedures cannot restore user-defined statistics.
- Old versions of statistics are not stored when the ANALYZE command has been used for collecting statistics.
- When you drop a table, workload information used by the auto-histogram gathering feature and saved statistics history used by the RESTORE_*_STATS procedures is lost. Without this data, these features do not function properly. To remove all rows from a table, and to restore these statistics with DBMS_STATS, use TRUNCATE instead of dropping and re-creating the same table.

There are dictionary views that can be used to determine the time stamp for restoration of statistics. The *_TAB_STATS_HISTORY views (ALL, DBA, or USER) contain a history of table statistics modifications. For the example in the slide, the time stamp was determined by:

```
select stats_update_time from dba_tab_stats_history  
where table_name = 'INVENTORIES'
```

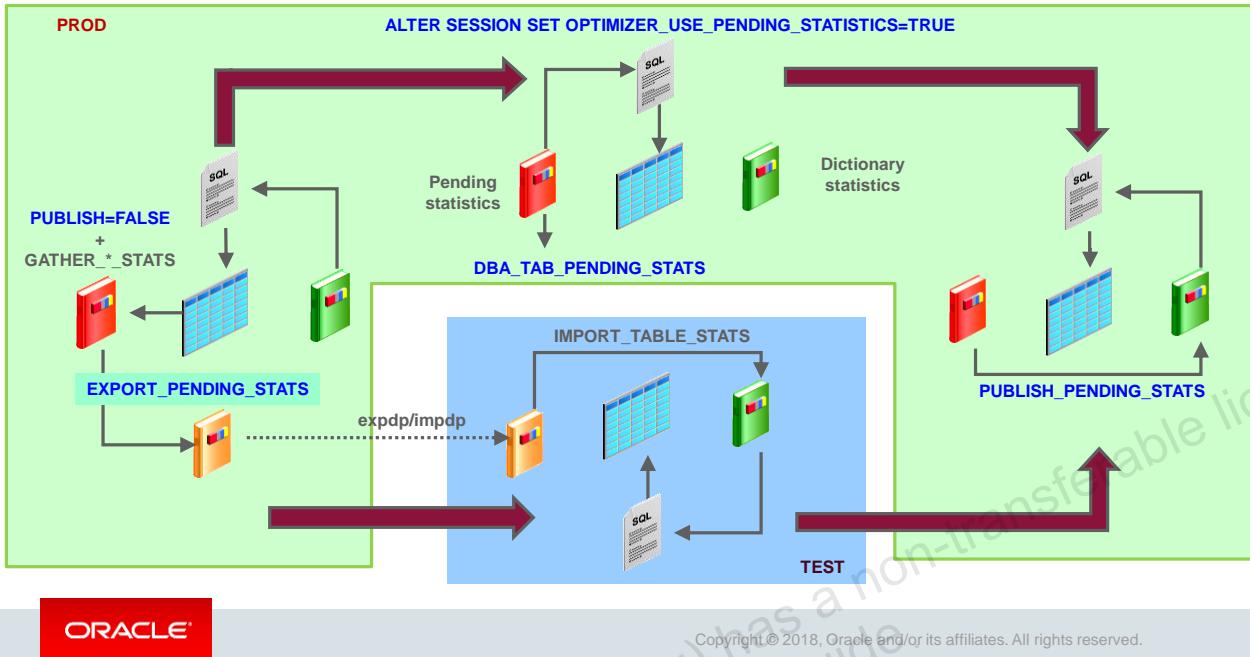
The database purges old statistics automatically at regular intervals based on the statistics history retention setting and the time of the recent analysis of the system.

You can use the following procedure to obtain information about the optimizer statistics history:

- GET_STATS_HISTORY_RETENTION
 - This function can retrieve the current statistics history retention value.
- GET_STATS_HISTORY_AVAILABILITY
 - This function retrieves the oldest time stamp when statistics history is available. Users cannot restore statistics to a time stamp older than the oldest time stamp.

You can configure retention by using the DBMS_STATS.ALTER_STATS_HISTORY_RETENTION procedure. The default value is 31 days, which means that you would be able to restore the optimizer statistics to any time in the last 31 days.

Deferred Statistics Publishing: Overview



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

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

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

When you specify the PUBLISH to FALSE gather option, gathered statistics are stored in the pending statistics tables instead of being current. These pending statistics are accessible from a number of views:

```
{ALL | DBA | USER} { TAB | COL | IND | TAB_HISTGRM } _PENDING_STATS.
```

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

To test the pending statistics, you have two options:

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

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

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

Deferred Statistics Publishing: Example

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

1

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

2

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

3

Execute your workload from the same session.

4

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

5



Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

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

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

Running Statistics Gathering Functions in Reporting Mode

- You can run the `DBMS_STATS` functions in reporting mode.
- In this mode, the optimizer does not actually gather statistics, but reports objects that would be processed if you were to use a specified statistics gathering function.
- `DBMS_STATS.REPORT_GATHER_*_STATS` functions can be used to set the report mode.

```
SET LINES 200 PAGES 0
SET LONG 100000
COLUMN REPORT FORMAT A200
VARIABLE my_report CLOB;
BEGIN
  :my_report :=DBMS_STATS.REPORT_GATHER_SCHEMA_STATS(
    ownname => 'OE',
    detail_level => 'TYPICAL',
    format => 'HTML' );
END;
/
```



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Reporting on Past Statistics Gathering Operations

- You can use DBMS_STATS functions to report on a specific statistics gathering operation or on operations that occurred during a specified time.
- REPORT_STATS_OPERATIONS generates a report of all statistics operations that occurred between two points in time.
- REPORT_SINGLE_STATS_OPERATION generates a report of the specified operation.

```
VARIABLE my_report CLOB;
BEGIN
:my_report := DBMS_STATS.REPORT_STATS_OPERATIONS ( since => SYSDATE-1 , until =>
SYSDATE , detail_level => 'TYPICAL' , format => 'HTML' );
END; /
```



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Managing SQL Plan Directives

- You can use DBMS_SPD procedures and functions to manage directives manually.
- You must have the Administer SQL Management Object privilege to execute the DBMS_SPD APIs.
- You can force the database to write the SQL plan directives to disk by using this syntax:

```
BEGIN  
  DBMS_SPD.FLUSH_SQL_PLAN_DIRECTIVE;  
END;
```

- You can delete the existing SQL plan directive for the sh schema:

```
BEGIN  
  DBMS_SPD.DROP_SQL_PLAN_DIRECTIVE ( directive_id =>  
    1484026771529551585 );  
END;
```



Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

The database automatically manages SQL plan directives. If the directives are not used in 53 weeks, then the database automatically purges them.

You can use DBMS_SPD procedures and functions to manage directives manually.

- FLUSH_SQL_PLAN_DIRECTIVE: Forces the database to write directives from memory to persistent storage in the SYSAUX tablespace
- DROP_SQL_PLAN_DIRECTIVE: Drops a SQL plan directive
- DBA_SQL_PLAN_DIRECTIVES: Displays information about the SQL plan directives in the system

Quiz



When there are no statistics for an object being used in a SQL statement, the optimizer uses:

- a. Rule-based optimization
- b. Dynamic statistics
- c. Fixed values
- d. Statistics gathered during the parse phase
- e. Random values



ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Answer: b, c

Quiz



There is a very volatile table in the database. The size of the table changes by more than 50 percent daily. What steps are part of the procedure to force dynamic statistics?

- a. Delete statistics.
- b. Lock statistics.
- c. Gather statistics when the table is at its largest.
- d. Set DYNAMIC_SAMPLING=9.
- e. Set DYNAMIC_SAMPLING=0.
- f. Allow the DYNAMIC_SAMPLING parameter to default.



ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Answer: a, b, f

Quiz



A query plan changes during execution because runtime conditions indicate that optimizer estimates are inaccurate.

- a. True
- b. False



ORACLE

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.

Answer: a

Summary

In this lesson, you should have learned how to:

- Gather optimizer statistics
- Set statistic preferences
- Use dynamic statistics
- Use automatic re-optimization
- Use SQL plan directives
- Discuss online statistics gathering for bulk loads
- Discuss concurrent statistics gathering
- Manage optimizer statistics



ORACLE®

Copyright© 2018, Oracle and/or its affiliates. All rights reserved.