

Oracle® Database

Testing Guide

12c Release 1 (12.1)

E20852-14

June 2013

Oracle Database Testing Guide, 12c Release 1 (12.1)

E20852-14

Copyright © 2008, 2013, Oracle and/or its affiliates. All rights reserved.

Primary Author: Immanuel Chan

Contributing Authors: Jim Garrison, Mike Zampiceni

Contributor: The Oracle Database 12c documentation is dedicated to Mark Townsend, who was an inspiration to all who worked on this release.

Contributors: Ashish Agrawal, Waleed Ahmed, Helen Altmar, Lance Ashdown, Pete Belknap, Supiti Buranawatanachoke, Romain Colle, Karl Dias, Kurt Engeleiter, Leonidas Galanis, Veeranjanyulu Goli, Prabhaker Gongloor, Prakash Gupta, Shantanu Joshi, Prathiba Kalirengan, Karen McKeen, Mughees Minhas, Konstantinos Morfonios, Valarie Moore, Ravi Pattabhi, Yujun Wang, Keith Wong, Khaled Yagoub, Hailing Yu

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

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

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

Preface	xiii
Audience	xiii
Documentation Accessibility	xiii
Related Documents	xiii
Conventions	xiv
Changes in This Release for Oracle Database Testing Guide	xv
Changes in Oracle Database 12c Release 1 (12.1)	xv
1 Introduction to Oracle Database Testing	
SQL Performance Analyzer	1-1
Database Replay	1-2
Test Data Management	1-3
Part I SQL Performance Analyzer	
2 Introduction to SQL Performance Analyzer	
Capturing the SQL Workload	2-3
Setting Up the Test System	2-4
Creating a SQL Performance Analyzer Task	2-5
Measuring the Pre-Change SQL Performance	2-5
Making a System Change	2-7
Measuring the Post-Change SQL Performance	2-7
Comparing Performance Measurements	2-7
Fixing Regressed SQL Statements	2-8
3 Creating an Analysis Task	
Creating an Analysis Task Using Enterprise Manager	3-2
Using the Parameter Change Workflow	3-3
Using the Optimizer Statistics Workflow	3-6
Using the Exadata Simulation Workflow	3-9
Using the Guided Workflow	3-12
Creating an Analysis Task Using APIs	3-13
Running the Exadata Simulation Using APIs	3-14

Remapping Multitenant Container Database Identifiers in an Analysis Task	3-15
4 Creating a Pre-Change SQL Trial	
Creating a Pre-Change SQL Trial Using Enterprise Manager.....	4-2
Creating a Pre-Change SQL Trial Using APIs	4-4
5 Creating a Post-Change SQL Trial	
Creating a Post-Change SQL Trial Using Oracle Enterprise Manager.....	5-2
Creating a Post-Change SQL Trial Using APIs	5-3
6 Comparing SQL Trials	
Comparing SQL Trials Using Oracle Enterprise Manager	6-2
Analyzing SQL Performance Using Oracle Enterprise Manager.....	6-2
Reviewing the SQL Performance Analyzer Report Using Oracle Enterprise Manager	6-3
Reviewing the SQL Performance Analyzer Report: General Information	6-5
Reviewing the SQL Performance Analyzer Report: Global Statistics	6-5
Reviewing the SQL Performance Analyzer Report: Global Statistics Details.....	6-6
About SQL Performance Analyzer Active Reports	6-7
Tuning Regressed SQL Statements Using Oracle Enterprise Manager	6-8
Creating SQL Plan Baselines	6-8
Running SQL Tuning Advisor	6-9
Comparing SQL Trials Using APIs.....	6-10
Analyzing SQL Performance Using APIs.....	6-10
Reviewing the SQL Performance Analyzer Report in Command-Line	6-12
General Information	6-12
Result Summary	6-13
Result Details	6-15
Comparing SQL Tuning Sets Using APIs.....	6-17
Tuning Regressed SQL Statements Using APIs.....	6-22
Tuning Regressed SQL Statements From a Remote SQL Trial Using APIs	6-24
Creating SQL Plan Baselines Using APIs	6-26
Using SQL Performance Analyzer Views.....	6-26
7 Testing a Database Upgrade	
Upgrading from Oracle9i Database and Oracle Database 10g Release 1	7-1
Enabling SQL Trace on the Production System.....	7-3
Creating a Mapping Table	7-4
Building a SQL Tuning Set	7-5
Testing Database Upgrades from Oracle9i Database and Oracle Database 10g Release 1.....	7-6
Testing Database Upgrades from Releases 9.x and 10.1 Using Cloud Control	7-6
Testing Database Upgrades from Releases 9.x and 10.1 Using APIs	7-9
Upgrading from Oracle Database 10g Release 2 and Newer Releases	7-10
Testing Database Upgrades from Oracle Database 10g Release 2 and Newer Releases	7-11
Testing Database Upgrades from Releases 10.2 and Higher Using Cloud Control.....	7-12
Testing Database Upgrades from Releases 10.2 and Higher Using APIs.....	7-15
Tuning Regressed SQL Statements After Testing a Database Upgrade	7-17

Part II Database Replay

8 Introduction to Database Replay

Workload Capture	8-2
Workload Preprocessing	8-3
Workload Replay	8-3
Analysis and Reporting	8-3

9 Capturing a Database Workload

Prerequisites for Capturing a Database Workload	9-1
Setting Up the Capture Directory	9-2
Workload Capture Options	9-2
Restarting the Database	9-2
Using Filters with Workload Capture	9-3
Workload Capture Restrictions	9-4
Enabling and Disabling the Workload Capture Feature	9-5
Enterprise Manager Privileges and Roles	9-6
Database Replay Viewer Role	9-6
Database Replay Operator Role	9-6
Capturing a Database Workload Using Enterprise Manager	9-7
Capturing Workloads from Multiple Databases Concurrently	9-12
Monitoring Workload Capture Using Enterprise Manager	9-14
Monitoring an Active Workload Capture	9-14
Stopping an Active Workload Capture	9-15
Viewing a Completed Workload Capture	9-15
Capturing a Database Workload Using APIs	9-16
Defining Workload Capture Filters	9-17
Starting a Workload Capture	9-17
Stopping a Workload Capture	9-19
Exporting AWR Data for Workload Capture	9-19
Importing AWR Data for Workload Capture	9-19
Monitoring Workload Capture Using Views	9-20

10 Preprocessing a Database Workload

Preparing a Single Database Workload Using Enterprise Manager	10-1
Creating a Database Replay Task	10-2
Creating a Replay from a Replay Task	10-3
Preparing the Test Database	10-4
Preprocessing the Workload and Deploying the Replay Clients	10-6
Preprocessing a Database Workload Using APIs	10-9
Running the Workload Analyzer Command-Line Interface	10-9

11 Replaying a Database Workload

Setting Up the Test System	11-1
Restoring the Database	11-1

Resetting the System Time.....	11-2
Steps for Replaying a Database Workload.....	11-2
Setting Up the Replay Directory	11-2
Resolving References to External Systems	11-2
Connection Remapping.....	11-3
User Remapping.....	11-3
Specifying Replay Options	11-3
Preserving COMMIT Order	11-4
Controlling Session Connection Rate.....	11-4
Controlling Request Rate Within a Session.....	11-4
Using Filters with Workload Replay	11-4
Setting Up Replay Clients	11-5
Calibrating Replay Clients.....	11-5
Starting Replay Clients.....	11-6
Displaying Host Information	11-7
Replaying a Database Workload Using Enterprise Manager	11-8
Monitoring Workload Replay Using Enterprise Manager.....	11-15
Monitoring an Active Workload Replay	11-15
Viewing a Completed Workload Replay.....	11-15
Replaying a Database Workload Using APIs	11-17
Initializing Replay Data.....	11-17
Remapping Connections	11-18
Remapping Users	11-18
Setting Workload Replay Options	11-19
Defining Workload Replay Filters and Replay Filter Sets	11-20
Adding Workload Replay Filters.....	11-21
Deleting Workload Replay Filters	11-21
Creating a Replay Filter Set	11-21
Using a Replay Filter Set.....	11-22
Setting the Replay Timeout Action.....	11-22
Starting a Workload Replay.....	11-24
Pausing a Workload Replay	11-24
Resuming a Workload Replay.....	11-24
Cancelling a Workload Replay.....	11-25
Exporting AWR Data for Workload Replay.....	11-25
Importing AWR Data for Workload Replay	11-25
Monitoring Workload Replay Using APIs.....	11-26
Retrieving Information About Diverged Calls	11-26
Monitoring Workload Replay Using Views.....	11-27

12 Analyzing Captured and Replayed Workloads

Using Workload Capture Reports.....	12-1
Accessing Workload Capture Reports Using Enterprise Manager	12-2
Generating Workload Capture Reports Using APIs.....	12-2
Reviewing Workload Capture Reports.....	12-3
Using Workload Replay Reports.....	12-4
Accessing Workload Replay Reports Using Enterprise Manager	12-4

Generating Workload Replay Reports Using APIs.....	12-8
Reviewing Workload Replay Reports.....	12-9
Using Replay Compare Period Reports	12-9
Generating Replay Compare Period Reports Using APIs	12-10
Reviewing Replay Compare Period Reports	12-11
General Information	12-11
Replay Divergence	12-11
Main Performance Statistics	12-11
Top SQL/Call	12-12
Hardware Usage Comparison	12-12
ADDM Comparison	12-12
ASH Data Comparison.....	12-12
Using SQL Performance Analyzer Reports.....	12-14
Generating SQL Performance Analyzer Reports Using APIs	12-14

13 Using Workload Intelligence

Overview of Workload Intelligence	13-1
About Workload Intelligence	13-1
Use Case for Workload Intelligence	13-2
Requirements for Using Workload Intelligence	13-2
Analyzing Captured Workloads Using Workload Intelligence.....	13-3
Creating a Database User for Workload Intelligence	13-3
Creating a Workload Intelligence Job	13-3
Generating a Workload Model.....	13-4
Identifying Patterns in a Workload	13-5
Generating a Workload Intelligence Report	13-6
Example: Workload Intelligence Results.....	13-7

14 Using Consolidated Database Replay

Use Cases for Consolidated Database Replay	14-1
Database Consolidation Using Pluggable Databases	14-2
Stress Testing	14-2
Scale-Up Testing.....	14-2
Steps for Using Consolidated Database Replay	14-2
Capturing Database Workloads for Consolidated Database Replay	14-2
Supported Types of Workload Captures.....	14-3
Capture Subsets.....	14-3
Setting Up the Test System for Consolidated Database Replay	14-4
Preprocessing Database Workloads for Consolidated Database Replay.....	14-5
Replaying Database Workloads for Consolidated Database Replay.....	14-5
Defining Replay Schedules	14-5
Remapping Connections for Consolidated Database Replay	14-6
Remapping Users for Consolidated Database Replay	14-6
Preparing for Consolidated Database Replay	14-6
Replaying Individual Workloads	14-7
Reporting and Analysis for Consolidated Database Replay	14-7

Using Consolidated Database Replay with Enterprise Manager	14-7
Using Consolidated Database Replay with APIs	14-8
Generating Capture Subsets Using APIs	14-9
Setting the Replay Directory Using APIs	14-9
Defining Replay Schedules Using APIs	14-10
Creating Replay Schedules Using APIs	14-11
Adding Workload Captures to Replay Schedules Using APIs	14-11
Adding Schedule Orders to Replay Schedules Using APIs	14-12
Saving Replay Schedules Using APIs	14-14
Running Consolidated Database Replay Using APIs	14-14
Initializing Consolidated Database Replay Using APIs	14-15
Remapping Connection Using APIs	14-16
Remapping Users Using APIs	14-16
Preparing for Consolidated Database Replay Using APIs	14-17
Starting Consolidated Database Replay Using APIs	14-18
Example: Playing a Consolidated Workload with APIs	14-18

15 Using Workload Scale-Up

Overview of Workload Scale-Up	15-1
About Time Shifting	15-1
About Workload Folding	15-2
About Schema Remapping	15-2
Using Time Shifting	15-2
Using Workload Folding	15-5
Using Schema Remapping	15-7

Part III Test Data Management

16 Data Discovery and Modeling

Creating an Application Data Model	16-2
Managing Sensitive Column Types	16-5
Associating a Database to an Application Data Model	16-6
Importing and Exporting an Application Data Model	16-6
Importing an ADM	16-7
Exporting an ADM	16-7
Verifying or Upgrading a Source Database	16-8
Using Self Update to Download the Latest Data Masking and Test Data Management Templates	16-9

17 Data Subsetting

About Inline Masking and Subsetting	17-1
Creating a Data Subset Definition	17-2
Generating a Subset Script	17-6
Saving a Subset Script	17-8
Importing and Exporting Subset Templates and Dumps	17-9
Importing a Subset Definition	17-9

Exporting a Subset Definition	17-11
Creating a Subset Version of a Target Database	17-11
Inline Masking and Subsetting Scenarios	17-12

18 Masking Sensitive Data

Overview of Oracle Data Masking	18-1
Data Masking Concepts	18-2
Security and Regulatory Compliance	18-2
Roles of Data Masking Users.....	18-2
Related Oracle Security Offerings	18-3
Agent Compatibility for Data Masking	18-3
Supported Data Types.....	18-3
Format Libraries and Masking Definitions	18-4
Recommended Data Masking Workflow	18-5
Data Masking Task Sequence	18-6
Defining Masking Formats	18-7
Creating New Masking Formats.....	18-7
Providing User-defined and Post-processing Functions	18-8
Using Masking Format Templates	18-9
Using Oracle-supplied Predefined Masking Formats	18-9
Patterns of Format Definitions.....	18-9
Category Definitions.....	18-10
Installing the DM_FMTLIB Package.....	18-11
Providing a Masking Format to Define a Column	18-12
Deterministic Masking Using the Substitute Format.....	18-15
Masking with an Application Data Model and Workloads	18-15
Adding Dependent Columns	18-19
Masking Dependent Columns for Packaged Applications.....	18-19
Selecting Data Masking Advanced Options	18-20
Data Masking Options	18-20
Random Number Generation.....	18-21
Pre- and Post-mask Scripts	18-22
Cloning the Production Database.....	18-23
Importing a Data Masking Template	18-23
Masking a Test System to Evaluate Performance	18-24
Using Only Masking for Evaluation.....	18-24
Using Cloning and Masking for Evaluation	18-25
Upgrade Considerations	18-25
Using the Shuffle Format	18-26
Using Group Shuffle	18-27
Using Conditional Masking	18-27
Using Data Masking with LONG Columns	18-28

Index

Preface

This preface contains the following topics:

- [Audience](#)
- [Documentation Accessibility](#)
- [Related Documents](#)
- [Conventions](#)

Audience

This document provides information about how to assure the integrity of database changes and manage test data using Oracle Real Application Testing, Oracle Data Masking, and data subsetting. This document is intended for database administrators, application designers, and programmers who are responsible for performing real-world testing of Oracle Database.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

Related Documents

For more information about some of the topics discussed in this document, see the following documents in the Oracle Database Release 12.1 documentation set:

- *Oracle Database 2 Day DBA*
- *Oracle Database 2 Day + Performance Tuning Guide*
- *Oracle Database Administrator's Guide*
- *Oracle Database Concepts*
- *Oracle Database Performance Tuning Guide*
- *Oracle Database SQL Tuning Guide*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Changes in This Release for Oracle Database Testing Guide

This preface lists changes in *Oracle Database Testing Guide*.

Changes in Oracle Database 12c Release 1 (12.1)

The following are changes in *Oracle Database Testing Guide* for Oracle Database 12c Release 1 (12.1).

New Features

The following features are new in this release:

- SQL Performance Analyzer support for the multitenant architecture
You can use a SQL tuning set that was transported from a non-CDB to a multitenant container database (CDB) as the input source to SQL Performance Analyzer by remapping its CDB identifiers.
See ["Remapping Multitenant Container Database Identifiers in an Analysis Task"](#) on page 3-15.
- Workload Intelligence
Workload Intelligence comprises a suite of Java programs that enable you to analyze the data stored in a captured workload.
See [Chapter 13, "Using Workload Intelligence"](#).
- Consolidated Database Replay
Consolidated Database Replay enables you to consolidate multiple workloads captured from one or multiple systems and replay them concurrently on a single test system, such as a single Oracle Exadata Machine.
See [Chapter 14, "Using Consolidated Database Replay"](#).
- Workload Scale-up
Database Replay enables you to perform scale-up testing and stress testing in various use cases and scenarios.
See [Chapter 15, "Using Workload Scale-Up"](#).
- Test data management
The test data management features enable you to manage test data by using Oracle Data Masking and data subsetting.

See [Part III, "Test Data Management"](#).

Other Changes

The following are additional changes in the release:

- New book

Oracle Database Real Application Testing User's Guide is now part of *Oracle Database Testing Guide*.

- New part

[Part III, "Test Data Management"](#) is added to *Oracle Database Testing Guide* to describe the test data management features of Oracle Database.

Introduction to Oracle Database Testing

The Oracle Real Application Testing option and test data management features of Oracle Database help you to securely assure the integrity of database changes and to manage test data.

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

When performing real-world testing, there is the risk of exposing sensitive data to non-production users in a test environment. The test data management features of Oracle Database helps to minimize this risk by enabling you to perform data masking and data subsetting on the test data.

This chapter contains the following sections:

- [SQL Performance Analyzer](#)
- [Database Replay](#)
- [Test Data Management](#)

Note: The use of SQL Performance Analyzer and Database Replay requires the Oracle Real Application Testing licensing option. For more information, see *Oracle Database Licensing Information*.

SQL Performance Analyzer

System changes—such as a upgrading a database or adding an index—may cause changes to execution plans of SQL statements, resulting in a significant impact on SQL performance. In some cases, the system changes may cause SQL statements to regress, resulting in performance degradation. In other cases, the system changes may improve SQL performance. Being able to accurately forecast the potential impact of system changes on SQL performance enables you to tune the system beforehand, in cases where the SQL statements regress, or to validate and measure the performance gain in cases where the performance of the SQL statements improves.

SQL Performance Analyzer automates the process of assessing the overall effect of a change on the full SQL workload by identifying performance divergence for each SQL statement. A report that shows the net impact on the workload performance due to the

change is provided. For regressed SQL statements, SQL Performance Analyzer also provides appropriate execution plan details along with tuning recommendations. As a result, you can remedy any negative outcome before the end users are affected. Furthermore, you can validate—with significant time and cost savings—that the system change to the production environment will result in net improvement.

You can use the SQL Performance Analyzer to analyze the impact on SQL performance of any type of system changes, including:

- Database upgrade
- Database consolidation testing for pluggable databases (PDBs) and manual schema consolidation
- Configuration changes to the operating system or hardware
- Schema changes
- Changes to database initialization parameters
- Refreshing optimizer statistics
- Validating SQL tuning actions

See Also:

- [Part I, "SQL Performance Analyzer"](#) for information about using SQL Performance Analyzer

Database Replay

Before system changes are made, such as hardware and software upgrades, extensive testing is usually performed in a test environment to validate the changes. However, despite the testing, the new system often experiences unexpected behavior when it enters production because the testing was not performed using a realistic workload. The inability to simulate a realistic workload during testing is one of the biggest challenges when validating system changes.

Database Replay enables realistic testing of system changes by essentially re-creating the production workload environment on a test system. Using Database Replay, you can capture a workload on the production system and replay it on a test system with the exact timing, concurrency, and transaction characteristics of the original workload. This enables you to fully assess the impact of the change, including undesired results, new contention points, or plan regressions. Extensive analysis and reporting is provided to help identify any potential problems, such as new errors encountered and performance divergence.

Database Replay captures the workload of external database clients at the database level and has negligible performance overhead. Capturing the production workload eliminates the need to develop simulation workloads or scripts, resulting in significant cost reduction and time savings. By using Database Replay, realistic testing of complex applications that previously took months using load simulation tools can now be completed in days. This enables you to rapidly test changes and adopt new technologies with a higher degree of confidence and at lower risk.

You can use Database Replay to test any significant system changes, including:

- Database and operating system upgrades
- Database consolidation testing for PDBs and manual schema consolidation
- Authoring and experimenting with various scenarios using workload scale-up

- Configuration changes, such as conversion of a database from a single instance to an Oracle Real Application Clusters (Oracle RAC) environment
- Storage, network, and interconnect changes
- Operating system and hardware migrations

See Also:

- [Part II, "Database Replay"](#) for information about using Database Replay

Test Data Management

When production data is copied into a testing environment, there is the risk of breaching sensitive information to non-production users, such as application developers or external consultants. In order to perform real-world testing, these non-production users need to access some of the original data, but not all the data, especially when the information is deemed confidential.

Oracle Database offers test data management features that help reduce this risk by enabling you to:

- Store the list of applications, tables, and relationships between table columns using Application Data Modeling
- Replicate information that pertains only to a particular site using data subsetting
- Replace sensitive data from your production system with fictitious data that can be used during testing using Oracle Data Masking

See Also:

- [Part III, "Test Data Management"](#) for information about managing test data

Part I

SQL Performance Analyzer

SQL Performance Analyzer enables you to assess the impact of system changes on the response time of SQL statements.

Part I covers SQL Performance Analyzer and contains the following chapters:

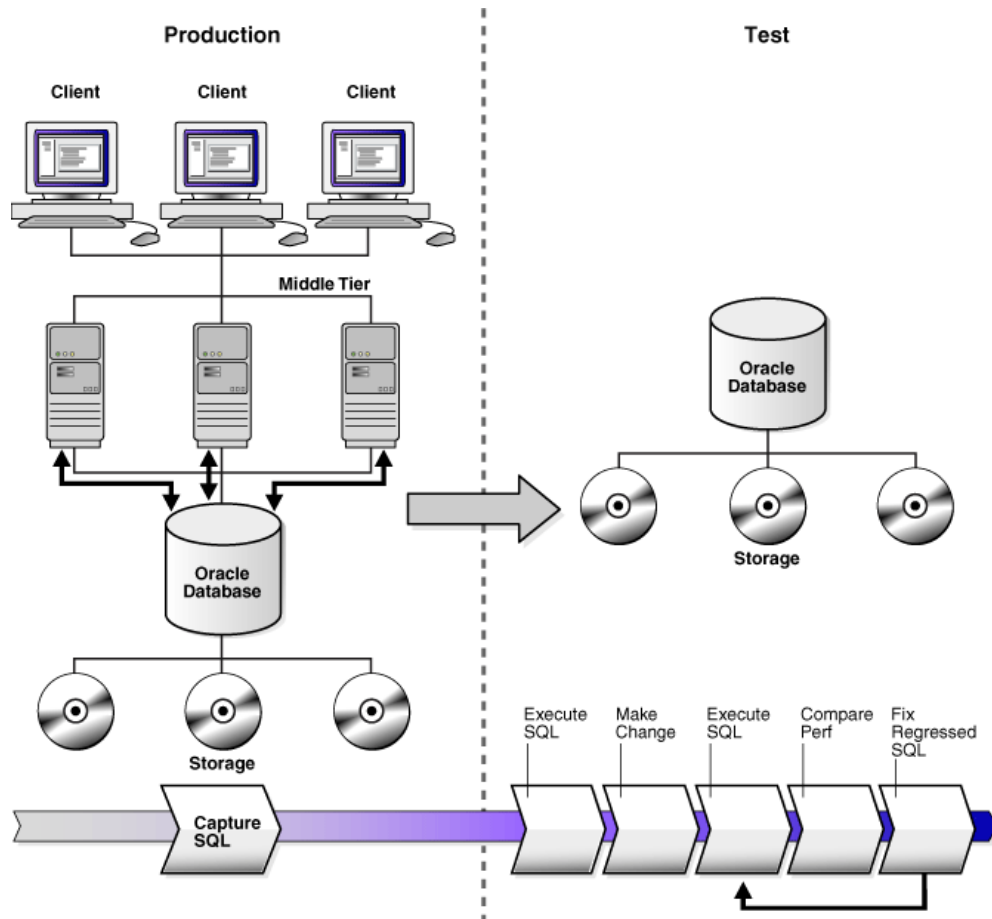
- [Chapter 2, "Introduction to SQL Performance Analyzer"](#)
- [Chapter 3, "Creating an Analysis Task"](#)
- [Chapter 4, "Creating a Pre-Change SQL Trial"](#)
- [Chapter 5, "Creating a Post-Change SQL Trial"](#)
- [Chapter 6, "Comparing SQL Trials"](#)
- [Chapter 7, "Testing a Database Upgrade"](#)

Introduction to SQL Performance Analyzer

You can run SQL Performance Analyzer on a production system or a test system that closely resembles the production system. Testing a system change on a production system will impact the system's throughput because SQL Performance Analyzer must execute the SQL statements that you are testing. Any global changes made on the system to test the performance effect may also affect other users of the system. If the system change does not impact many sessions or SQL statements, then running SQL Performance Analyzer on the production system may be acceptable. However, for systemwide changes—such as a database upgrade—using a production system is not recommended. Instead, consider running SQL Performance Analyzer on a separate test system so that you can test the effects of the system change without affecting the production system. Using a test system also ensures that other workloads running on the production system will not affect the analysis performed by SQL Performance Analyzer. Running SQL Performance Analyzer on a test system is the recommended approach and the methodology described here. If you choose to run the SQL Performance Analyzer on the production system, then substitute the production system for the test system where applicable.

Analyzing the SQL performance effect of system changes using SQL Performance Analyzer involves the following steps, as illustrated in [Figure 2-1](#):

Figure 2–1 SQL Performance Analyzer Workflow



1. Capture the SQL workload that you intend to analyze and store it in a SQL tuning set, as described in ["Capturing the SQL Workload"](#) on page 2-3.
2. If you plan to use a test system separate from your production system, then perform the following steps:
 - a. Set up the test system to match the production environment as closely as possible.
 - b. Transport the SQL tuning set to the test system.

For more information, see ["Setting Up the Test System"](#) on page 2-4.

3. On the test system, create a SQL Performance Analyzer task, as described in ["Creating a SQL Performance Analyzer Task"](#) on page 2-5.
4. Build the pre-change SQL trial by test executing or generating execution plans for the SQL statements stored in the SQL tuning set, as described in ["Measuring the Pre-Change SQL Performance"](#) on page 2-5
5. Perform the system change, as described in ["Making a System Change"](#) on page 2-7
6. Build the post-change SQL trial by re-executing the SQL statements in the SQL tuning set on the post-change test system, as described in ["Measuring the Post-Change SQL Performance"](#) on page 2-7

7. Compare and analyze the pre-change and post-change versions of performance data, and generate a report to identify the SQL statements that have improved, remained unchanged, or regressed after the system change, as described in ["Comparing Performance Measurements"](#) on page 2-7
8. Tune any regressed SQL statements that are identified, as described in ["Fixing Regressed SQL Statements"](#) on page 2-8.
9. Ensure that the performance of the tuned SQL statements is acceptable by repeating steps 6 through 8 until your performance goals are met.

For each comparison, you can use any previous SQL trial as the pre-change SQL trial and the current SQL trial as the post-change SQL trial. For example, you may want to compare the first SQL trial to the current SQL trial to assess the total change, or you can compare the most recent SQL trial to the current SQL trial to assess just the most recent change.

Note: Oracle Enterprise Manager provides automated workflows for steps 3 through 9 to simplify this process.

Note: Data visibility and privilege requirements may differ when using SQL Performance Analyzer with pluggable databases (PDBs). For information about how manageability features—including SQL Performance Analyzer—work in a multitenant container database (CDB), see *Oracle Database Administrator's Guide*.

Capturing the SQL Workload

Before running SQL Performance Analyzer, capture a set of SQL statements on the production system that represents the SQL workload which you intend to analyze.

The captured SQL statements should include the following information:

- SQL text
- Execution environment
 - SQL binds, which are bind values needed to execute a SQL statement and generate accurate execution statistics
 - Parsing schema under which a SQL statement can be compiled
 - Compilation environment, including initialization parameters under which a SQL statement is executed
- Number of times a SQL statement was executed

Capturing a SQL workload has a negligible performance impact on your production system and should not affect throughput. A SQL workload that contains more SQL statements will better represent the state of the application or database. This will enable SQL Performance Analyzer to more accurately forecast the potential impact of system changes on the SQL workload. Therefore, you should capture as many SQL statements as possible. Ideally, you should capture all SQL statements that are either called by the application or are running on the database.

You can store captured SQL statements in a SQL tuning set and use it as an input source for SQL Performance Analyzer. A SQL tuning set is a database object that includes one or more SQL statements, along with their execution statistics and execution context. SQL statements can be loaded into a SQL tuning set from different

sources, including the cursor cache, Automatic Workload Repository (AWR), SQL trace files, and existing SQL tuning sets. Capturing a SQL workload using a SQL tuning set enables you to:

- Store the SQL text and any necessary auxiliary information in a single, persistent database object
- Populate, update, delete, and select captured SQL statements in the SQL tuning set
- Load and merge content from various data sources, such as the Automatic Workload Repository (AWR) or the cursor cache
- Export the SQL tuning set from the system where the SQL workload is captured and import it into another system
- Reuse the SQL workload as an input source for other advisors, such as the SQL Tuning Advisor and the SQL Access Advisor

See Also:

- *Oracle Database 2 Day + Performance Tuning Guide* for information about creating SQL tuning sets using Oracle Enterprise Manager
- *Oracle Database SQL Tuning Guide* for information about creating SQL tuning sets using APIs

Setting Up the Test System

After you have captured the SQL workload into a SQL tuning set on the production system, you can conduct SQL Performance Analyzer analysis on the same database where the workload was captured or on a different database. Because the analysis is resource-intensive, it is recommended that you capture the workload on a production database and transport it to a separate test database where the analysis can be performed. To do so, export the SQL tuning set from the production system and import it into a separate system where the system change will be tested.

There are many ways to create a test database. For example, you can use the `DUPLICATE` command of Recovery Manager (RMAN), Oracle Data Pump, or transportable tablespaces. Oracle recommends using RMAN because it can create the test database from pre-existing backups or from the active production datafiles. The production and test databases can reside on the same host or on different hosts.

You should configure the test database environment to match the database environment of the production system as closely as possible. In this way, SQL Performance Analyzer can more accurately forecast the effect of the system change on SQL performance.

After the test system is properly configured, export the SQL tuning set from the production system to a staging table, then import it from the staging table into the test system.

See Also:

- *Oracle Database Backup and Recovery User's Guide* for information about duplicating a database with RMAN
- *Oracle Database 2 Day + Performance Tuning Guide* for information about transporting SQL tuning sets using Oracle Enterprise Manager
- *Oracle Database SQL Tuning Guide* for information about transporting SQL tuning sets using APIs

Creating a SQL Performance Analyzer Task

After the SQL workload is captured and transported to the test system, and the initial database environment is properly configured, you can run SQL Performance Analyzer to analyze the effects of a system change on SQL performance.

To run SQL Performance Analyzer, you must first create a SQL Performance Analyzer task. A task is a container that encapsulates all of the data about a complete SQL Performance Analyzer analysis. A SQL Performance Analyzer analysis comprises of at least two SQL trials and a comparison. A SQL trial encapsulates the execution performance of a SQL tuning set under specific environmental conditions. When creating a SQL Performance Analyzer task, you will need to select a SQL tuning set as its input source. When building SQL trials using the test execute or explain plan methods, the SQL tuning set will be used as the source for SQL statements. The SQL Performance Analyzer analysis will show the impact of the environmental differences between the two trials.

See Also:

- [Chapter 3, "Creating an Analysis Task"](#) for information about how to create a SQL Performance Analyzer task

Measuring the Pre-Change SQL Performance

Create a pre-change SQL trial before making the system change. You can use the following methods to generate the performance data needed for a SQL trial with SQL Performance Analyzer:

- Test execute

This method test executes SQL statements through SQL Performance Analyzer. This can be done on the database running SPA Performance Analyzer or on a remote database.

- Explain plan

This method generates execution plans only for SQL statements through SQL Performance Analyzer. This can be done on the database running SPA Performance Analyzer or on a remote database. Unlike the `EXPLAIN PLAN` statement, SQL trials using the explain plan method take bind values into account and generate the actual execution plan.

- Convert SQL tuning set

This method converts the execution statistics and plans stored in a SQL tuning set. This is only supported for APIs.

The test execute method runs each of the SQL statements contained in the workload to completion. During execution, SQL Performance Analyzer generates execution plans and computes execution statistics for each SQL statement in the workload. Each SQL statement in the SQL tuning set is executed separately from other SQL statements, without preserving their initial order of execution or concurrency. This is done at least twice for each SQL statement, for as many times as possible until the execution times out (up to a maximum of 10 times). The first execution is used to warm the buffer cache. All subsequent executions are then used to calculate the run-time execution statistics for the SQL statement based on their averages. The actual number of times that the SQL statement is executed depends on how long it takes to execute the SQL statement. Long-running SQL statement will only be executed a second time, and the execution statistics from this execution will be used. Other (faster-running) SQL statements are executed multiple times, and their execution statistics are averaged

over these executions (statistics from the first execution are not used in the calculation). By averaging statistics over multiple executions, SQL Performance Analyzer can calculate more accurate execution statistics for each SQL statement. To avoid a potential impact to the database, DDLs are not supported. By default, only the query portion of DMLs is executed. Using APIs, you can execute the full DML by using the `EXECUTE_FULLDML` task parameter. Parallel DMLs are not supported and the query portion is not executed unless the parallel hints are removed.

Depending on its size, executing a SQL workload can be time and resource intensive. With the explain plan method, you can choose to generate execution plans only, without collecting execution statistics. This technique shortens the time to run the trial and lessens the effect on system resources, but a comprehensive performance analysis is not possible because only the execution plans will be available during the analysis. However, unlike generating a plan with the `EXPLAIN PLAN` command, SQL Performance Analyzer provides bind values to the optimizer when generating execution plans, which provides a more reliable prediction of what the plan will be when the SQL statement is executed.

In both cases, you can execute the SQL workload remotely on a separate database using a database link. SQL Performance Analyzer will establish a connection to the remote database using the database link, execute the SQL statements on that database, collect the execution plans and run-time statistics for each SQL statement, and store the results in a SQL trial on the local database that can be used for later analysis. This method is useful in cases where you want to:

- Test a database upgrade
- Execute the SQL workload on a system running another version of Oracle Database
- Store the results from the SQL Performance Analyzer analysis on a separate test system
- Perform testing on multiple systems with different hardware configurations
- Use the newest features in SQL Performance Analyzer even if you are using an older version of Oracle Database on your production system

Once the SQL workload is executed, the resulting execution plans and run-time statistics are stored in a SQL trial.

You can also build a SQL trial using the execution statistics and plan stored in a SQL tuning set. While this method is only supported for APIs, it may be useful in cases where you have another method to run your workload (such as Database Replay or another application testing tool), and you do not need SQL Performance Analyzer to drive the workload on the test system. In such cases, if you capture a SQL tuning set during your test runs, you can build SQL trials from these SQL tuning sets using SQL Performance Analyzer to view a more comprehensive analysis report. Unlike a standard SQL Performance Analyzer report—which has only one execution plan in each trial and one set of execution statistics generated by executing the SQL statement with one set of binds—you can generate a report that compares SQL trials built from SQL tuning sets that show all execution plans from both trials with potentially many different sets of binds across multiple executions.

See Also:

- [Chapter 4, "Creating a Pre-Change SQL Trial"](#) for information about how to measure the pre-change performance
- [Chapter 7, "Testing a Database Upgrade"](#) for information about executing a SQL workload on a remote system to test a database upgrade

Making a System Change

Make the change whose effect on SQL performance you intend to measure. SQL Performance Analyzer can analyze the effect of many types of system changes. For example, you can test a database upgrade, new index creation, initialization parameter changes, or optimizer statistics refresh. If you are running SQL Performance Analyzer on the production system, then consider making a change using a private session to avoid affecting the rest of the system.

Measuring the Post-Change SQL Performance

After performing the system change, create a post-change SQL trial. It is highly recommended that you create the post-change SQL trial using the same method as the pre-change SQL trial. Once built, the post-change SQL trial represents a new set of performance data that can be used to compare to the pre-change version. The results are stored in a new, or post-change, SQL trial.

See Also:

- [Chapter 5, "Creating a Post-Change SQL Trial"](#) for information about how to measure the post-change performance

Comparing Performance Measurements

SQL Performance Analyzer compares the performance of SQL statements before and after the change and produces a report identifying any changes in execution plans or performance of the SQL statements.

SQL Performance Analyzer measures the impact of system changes both on the overall execution time of the SQL workload and on the response time of every individual SQL statement in the workload. By default, SQL Performance Analyzer uses elapsed time as a metric for comparison. Alternatively, you can choose the metric for comparison from a variety of available SQL run-time statistics, including:

- CPU time
- User I/O time
- Buffer gets
- Physical I/O
- Optimizer cost
- I/O interconnect bytes
- Any combination of these metrics in the form of an expression

If you chose to generate explain plans only in the SQL trials, then SQL Performance Analyzer will use the optimizer cost stored in the SQL execution plans.

Once the comparison is complete, the resulting data is generated into a SQL Performance Analyzer report that compares the pre-change and post-change SQL performance. The SQL Performance Analyzer report can be viewed as an HTML, text, or active report. Active reports provides in-depth reporting using an interactive user interface that enables you to perform detailed analysis even when disconnected from the database or Oracle Enterprise Manager.

See Also:

- [Chapter 6, "Comparing SQL Trials"](#) for information about comparing performance measurements and reporting

Fixing Regressed SQL Statements

If the performance analysis performed by SQL Performance Analyzer reveals regressed SQL statements, then you can make changes to remedy the problem. For example, you can fix regressed SQL by running SQL Tuning Advisor or using SQL plan baselines. You can then repeat the process of executing the SQL statements and comparing its performance to the first execution. Repeat these steps until you are satisfied with the outcome of the analysis.

See Also:

- [Chapter 6, "Comparing SQL Trials"](#) for information about fixing regressed SQL statements

Creating an Analysis Task

Once you have captured a SQL workload that you want to analyze into a SQL tuning set (STS), you can run SQL Performance Analyzer to analyze the effects of a system change on SQL performance. To run SQL Performance Analyzer, you must first create a SQL Performance Analyzer task. A task is a container that encapsulates all of the data about a complete SQL Performance Analyzer analysis. A SQL Performance Analyzer analysis comprises of at least two SQL trials and a comparison. A SQL trial captures the execution performance of a SQL tuning set under specific environmental conditions and can be generated automatically using SQL Performance Analyzer by one of the following methods:

- Test executing SQL statements
- Generating execution plans for SQL statements
- Referring to execution statistics and plans captured in a SQL tuning set

When creating a SQL Performance Analyzer task, you will need to select a SQL tuning set as its input source. The SQL tuning set will be used as the source for test executing or generating execution plans for SQL trials. Thus, performance differences between trials are caused by environmental differences. For more information, see ["Creating a SQL Performance Analyzer Task"](#) on page 2-5.

This chapter described how to create a SQL Performance Analyzer task and contains the following topics:

- [Creating an Analysis Task Using Enterprise Manager](#)
- [Creating an Analysis Task Using APIs](#)
- [Remapping Multitenant Container Database Identifiers in an Analysis Task](#)

Note: The primary interface for running SQL Performance Analyzer is Oracle Enterprise Manager. If for some reason Oracle Enterprise Manager is unavailable, you can run SQL Performance Analyzer using the `DBMS_SQLPA` PL/SQL package.

Tip: Before running SQL Performance Analyzer, capture the SQL workload to be used in the performance analysis into a SQL tuning set on the production system, then transport it to the test system where the performance analysis will be performed, as described in ["Capturing the SQL Workload"](#) on page 2-3.

Creating an Analysis Task Using Enterprise Manager

There are several workflows available in Oracle Enterprise Manager for creating a SQL Performance Analyzer task.

To create an analysis task using Enterprise Manager:

1. From the **Performance** menu, select **SQL**, then **SQL Performance Analyzer**.

If the Database Login page appears, then log in as a user with administrator privileges.

The SQL Performance Analyzer page appears.

SQL Performance Analyzer

Page Refreshed **May 1, 2012 5:29:47 PM PDT** [Refresh](#) [View Data](#) **Real Time: 15 Second Refresh**

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

SQL Performance Analyzer Workflows

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

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

SQL Performance Analyzer Tasks

Select	Name	Owner	Last Modified	Current Step Name	Type	Last Run Status	SQLs Processed	Steps Completed
	No SQL Performance Analyzer Tasks available.							

TIP For an explanation of the icons and symbols used in the following table, see the [Icon Key](#)

2. Under SQL Performance Analyzer Workflows, select the workflow for creating the desired type of analysis task:

- Upgrade from 9i or 10.1

Use the upgrade from 9i or 10.1 workflow to test a database upgrade from Oracle9i Database or Oracle Database 10g Release 1 to Oracle Database 10g Release 2 and newer releases, as described in ["Upgrading from Oracle9i Database and Oracle Database 10g Release 1"](#) on page 7-1.

- Upgrade from 10.2 or 11g

Use the upgrade from 10.2 or 11g workflow to test a database upgrade from Oracle Database 10g Release 2 or Oracle Database 11g to a later release, as described in ["Upgrading from Oracle Database 10g Release 2 and Newer Releases"](#) on page 7-10.

- Parameter Change

Use the parameter change workflow to determine how a database initialization parameter change will affect SQL performance, as described in ["Using the Parameter Change Workflow"](#) on page 3-3.

- Optimizer Statistics

Use the optimizer statistics workflow to analyze how changes to optimizer statistics will affect SQL performance, as described in ["Using the Optimizer Statistics Workflow"](#) on page 3-6.

- Exadata Simulation

Use the Exadata simulation workflow to simulate how using Oracle Exadata will affect SQL performance, as described in ["Using the Exadata Simulation Workflow"](#) on page 3-9.

- Guided workflow

Use the guided workflow to compare SQL performance for all other types of system changes, as described in ["Using the Guided Workflow"](#) on page 3-12.

Using the Parameter Change Workflow

The parameter change workflow enables you to test the performance effect on a SQL workload when you change the value of a single environment initialization parameter. For example, you can compare SQL performance by setting the `OPTIMIZER_FEATURES_ENABLE` initialization parameter to 10.2.0.4 and 12.1.0.1.

After you select a SQL tuning set and a comparison metric, SQL Performance Analyzer creates a task and performs a trial with the initialization parameter set to the original value. SQL Performance Analyzer then performs a second trial with the parameter set to the changed value by issuing an `ALTER SESSION` statement. The impact of the change is thus contained locally to the testing session. Any regression or change in performance is reported in a system-generated SQL Performance Analyzer report.

Note: To create an analysis task for other types of system changes, use the guided workflow instead, as described in ["Using the Guided Workflow"](#) on page 3-12.

To use the SQL Performance Analyzer parameter change workflow:

1. On the SQL Performance Analyzer page, under SQL Performance Analyzer Workflows, click **Parameter Change**.

The Parameter Change page appears.

Parameter Change

Task Information

* Task Name

* SQL Tuning Set

Description

Creation Method

Per-SQL Time Limit

TIP Time limit is on elapsed time of test execution of SQL.

Parameter Change

* Parameter Name

* Base Value

* Changed Value

Trial Comparison

Comparison Metric

Schedule

Time Zone

☒ Immediately

☐ Later

Date

(example: May 1, 2012)

Time ☐ AM ☒ PM

2. In the Task Name field, enter the name of the task.
3. In the SQL Tuning Set field, enter the name of the SQL tuning set that contains the SQL workload to be analyzed.

Alternatively, click the search icon to search for a SQL tuning set using the Search and Select: SQL Tuning Set window.

The selected SQL tuning set now appears in the SQL Tuning Set field.
4. In the Description field, optionally enter a description of the task.
5. In the Creation Method list, determine how the SQL trial is created and what contents are generated by performing one of the following actions:
 - **Select Execute SQLs.**
The SQL trial generates both execution plans and statistics for each SQL statement in the SQL tuning set by actually running the SQL statements.
 - **Select Generate Plans.**
The SQL trial invokes the optimizer to create execution plans only without actually running the SQL statements.
6. In the Per-SQL Time Limit list, determine the time limit for SQL execution during the trial by performing one of the following actions:
 - **Select 5 minutes.**
The execution will run each SQL statement in the SQL tuning set up to 5 minutes and gather performance data.
 - **Select Unlimited.**
The execution will run each SQL statement in the SQL tuning set to completion and gather performance data. Collecting execution statistics provides greater accuracy in the performance analysis but takes a longer time. Using this setting is not recommended because the task may be stalled by one SQL statement for a prolonged time period.
 - **Select Customize** and enter the specified number of seconds, minutes, or hours.
7. In the Parameter Change section, complete the following steps:
 - a. In the Parameter Name field, enter the name of the initialization parameter whose value you want to modify, or click the Search icon to select an initialization parameter using the Search and Select: Initialization Parameters window.
 - b. In the Base Value field, enter the current value of the initialization parameter.
 - c. In the Changed Value field, enter the new value of the initialization parameter.
8. In the Comparison Metric list, select the comparison metric to use for the analysis:
 - If you selected **Generate Plans** in Step 5, then select **Optimizer Cost**.
 - If you selected **Execute SQLs** in Step 5, then select one of the following options:
 - **Elapsed Time**
 - **CPU Time**
 - **User I/O Time**

- **Buffer Gets**
- **Physical I/O**
- **Optimizer Cost**
- **I/O Interconnect Bytes**

To perform the comparison analysis by using more than one comparison metric, perform separate comparison analyses by repeating this procedure using different metrics.

9. In the Schedule section:

- a. In the Time Zone list, select your time zone code.
- b. Select **Immediately** to start the task now, or **Later** to schedule the task to start at a time specified using the Date and Time fields.

10. Click **Submit**.

The SQL Performance Analyzer page appears.

In the SQL Performance Analyzer Tasks section, the status of this task is displayed. To refresh the status icon, click **Refresh**. After the task completes, the Status field changes to Completed.

SQL Performance Analyzer Tasks								
Delete		View Latest Report						
Select	Name	Owner	Last Modified	Current Step Name	Type	Last Run Status	SQLs Processed	Steps Completed
	SPA_PARAM_CHANGE	SYS	May 1, 2012 5:42:32 PM	EXEC_187	Compare	Completed	506 of 506	4 of 4
TIP For an explanation of the icons and symbols used in the following table, see the Icon Key								

11. In the SQL Performance Analyzer Tasks section, select the task and click the link in the Name column.

The SQL Performance Analyzer Task page appears.

SQL Performance Analyzer Task: SYS.SPA_PARAM_CHANGE

[View Latest Report](#)

Page Refreshed **May 1, 2012 5:14:01 PM PDT**

[Refresh](#)

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

[SQL Tuning Set](#)

[SQL Trials](#)

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

[Create SQL Trial](#)

SQL Trial Name	Description	Created	SQL Executed	Status
INITIAL_SQL_TRIAL	parameter db_file_multiblock_read_count set to 128	5/1/12 5:41 PM	Yes	COMPLETED
SECOND_SQL_TRIAL	parameter db_file_multiblock_read_count set to 64	5/1/12 5:42 PM	Yes	COMPLETED

[SQL Trial Comparisons](#)

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

[Run SQL Trial Comparison](#)

Trial 1 Name	Trial 2 Name	Compare Metric	Created	Status	Comparison Report	SQL Tune Report
INITIAL_SQL_TRIAL	SECOND_SQL_TRIAL	Elapsed Time	5/1/12 5:42 PM	COMPLETED		

This page contains the following sections:

- **SQL Tuning Set**

This section summarizes information about the SQL tuning set, including its name, owner, description, and the number of SQL statements it contains.

- **SQL Trials**

This section includes a table that lists the SQL trials used in the SQL Performance Analyzer task.

- **SQL Trial Comparisons**

This section contains a table that lists the results of the SQL trial comparisons

12. Click the icon in the Comparison Report column.

The SQL Performance Analyzer Task Result page appears.

13. Review the results of the performance analysis, as described in ["Reviewing the SQL Performance Analyzer Report Using Oracle Enterprise Manager"](#) on page 6-3.
14. In cases when regression are identified, click the icon in the SQL Tune Report column to view a SQL tuning report.

Using the Optimizer Statistics Workflow

The optimizer statistics workflow enables you to analyze the effects of optimizer statistics changes on the performance of a SQL workload.

SQL Performance Analyzer tests the effect of new optimizer statistics by enabling pending optimizer statistics in the testing session. The first SQL trial measures the baseline SQL tuning set performance; the second SQL trial uses the pending optimizer statistics. You can then run a comparison report for the two SQL trials.

To use the optimizer statistics workflow:

1. On the SQL Performance Analyzer page, under SQL Performance Analyzer Workflows, click **Optimizer Statistics**.

The Optimizer Statistics page appears.

Optimizer Statistics

Task Information

* Task Name

* SQL Tuning Set

Description

Creation Method

Per-SQL Time Limit

TIP Time limit is on elapsed time of test execution of SQL.

Trial Comparison

Comparison Metric

Schedule

Time Zone

☒ Immediately ☐ Later

Date

(example: May 2, 2012)

Time ☐ AM ☒ PM

2. In the Task Name field, enter the name of the task.

3. In the SQL Tuning Set field, enter the name of the SQL tuning set that contains the SQL workload to be analyzed.

Alternatively, click the search icon to search for a SQL tuning set using the Search and Select: SQL Tuning Set window.

The selected SQL tuning set now appears in the SQL Tuning Set field.

4. In the Description field, optionally enter a description of the task.
5. In the Creation Method list, determine how the SQL trial is created and what contents are generated by performing one of the following actions:

- Select **Execute SQLs**.

The SQL trial generates both execution plans and statistics for each SQL statement in the SQL tuning set by actually running the SQL statements.

- Select **Generate Plans**.

The SQL trial invokes the optimizer to create execution plans only without actually running the SQL statements.

6. In the Per-SQL Time Limit list, determine the time limit for SQL execution during the trial by performing one of the following actions:

- Select **5 minutes**.

The execution will run each SQL statement in the SQL tuning set up to 5 minutes and gather performance data.

- Select **Unlimited**.

The execution will run each SQL statement in the SQL tuning set to completion and gather performance data. Collecting execution statistics provides greater accuracy in the performance analysis but takes a longer time. Using this setting is not recommended because the task may be stalled by one SQL statement for a prolonged time period.

- Select **Customize** and enter the specified number of seconds, minutes, or hours.

7. In the Comparison Metric list, select the comparison metric to use for the comparison analysis:

- **Elapsed Time**
- **CPU Time**
- **User I/O Time**
- **Buffer Gets**
- **Physical I/O**
- **Optimizer Cost**
- **I/O Interconnect Bytes**

Optimizer Cost is the only comparison metric available if you chose to generate execution plans only in the SQL trials.

To perform the comparison analysis by using more than one comparison metric, perform separate comparison analyses by repeating this procedure with different metrics.

8. Ensure that pending optimizer statistics are collected, and select **Pending optimizer statistics collected**.

9. In the Schedule section:
 - a. In the Time Zone list, select your time zone code.
 - b. Select **Immediately** to start the task now, or **Later** to schedule the task to start at a time specified using the Date and Time fields.

10. Click **Submit**.

The SQL Performance Analyzer page appears.

In the SQL Performance Analyzer Tasks section, the status of this task is displayed. To refresh the status icon, click **Refresh**. After the task completes, the Status field changes to Completed.

SQL Performance Analyzer Tasks								
Delete		View Latest Report						
Select	Name	Owner	Last Modified	Current Step Name	Type	Last Run Status	SQLs Processed	Steps Completed
	SPA_OPTIMIZER_STATS	SYS	May 2, 2012 1:46:34 PM	EXEC_210	Compare	Completed	506 of 506	4 of 4

11. In the SQL Performance Analyzer Tasks section, select the task and click the link in the Name column.

The SQL Performance Analyzer Task page appears.

SQL Performance Analyzer Task: SYS.SPA_OPTIMIZER_STATS

View Latest Report

Page Refreshed May 2, 2012 12:53:51 PM PDT

Refresh

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

SQL Tuning Set

SQL Trials

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

Create SQL Trial

SQL Trial Name	Description	Created	SQL Executed	Status
INITIAL_SQL_TRIAL	parameter optimizer_use_pending_statistics set to FALSE	5/2/12 1:45 PM	Yes	COMPLETED
SECOND_SQL_TRIAL	parameter optimizer_use_pending_statistics set to TRUE	5/2/12 1:45 PM	Yes	COMPLETED

SQL Trial Comparisons

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

Run SQL Trial Comparison

Trial 1 Name	Trial 2 Name	Compare Metric	Created	Status	Comparison Report	SQL Tune Report
INITIAL_SQL_TRIAL	SECOND_SQL_TRIAL	Elapsed Time	5/2/12 1:46 PM	COMPLETED		

This page contains the following sections:

- **SQL Tuning Set**
This section summarizes information about the SQL tuning set, including its name, owner, description, and the number of SQL statements it contains.
- **SQL Trials**
This section includes a table that lists the SQL trials used in the SQL Performance Analyzer task.
- **SQL Trial Comparisons**
This section contains a table that lists the results of the SQL trial comparisons

12. Click the icon in the Comparison Report column.

The SQL Performance Analyzer Task Result page appears.

13. Review the results of the performance analysis, as described in ["Reviewing the SQL Performance Analyzer Report Using Oracle Enterprise Manager"](#) on page 6-3.

Any regressions found in performance can be fixed using SQL plan baselines and the SQL Tuning Advisor. If the pending optimizer statistics produce satisfactory performance, you can publish for use.

Using the Exadata Simulation Workflow

The Exadata simulation workflow enables you to simulate the effects of an Exadata Storage Server installation on the performance of a SQL workload.

Oracle Exadata provides extremely large I/O bandwidth coupled with a capability to offload SQL processing from the database to storage. This allows Oracle Database to significantly reduce the volume of data sent through the I/O interconnect, while at the same time offloading CPU resources to the Exadata storage cells.

SQL Performance Analyzer can analyze the effectiveness of Exadata SQL offload processing by simulating an Exadata Storage Server installation and measuring the reduction in I/O interconnect usage for the SQL workload.

Running the Exadata simulation does not require any hardware or configuration changes to your system. After you select a SQL tuning set, SQL Performance Analyzer creates a task and performs an initial trial with the Exadata Storage Server simulation disabled. SQL Performance Analyzer then performs a second trial with the Exadata Storage Server simulation enabled. SQL Performance Analyzer then compares the two trials using the I/O Interconnect Bytes comparison metric and generates a SQL Performance Analyzer report, which estimates the amount of data that would not need to be sent from the Exadata storage cells to the database if Oracle Exadata is being used. In both SQL trials, the SQL statements are executed to completion and I/O interconnect bytes measurements are taken as the actual and simulated Exadata values for the first and second trials, respectively. The measured change in I/O interconnect bytes provides a good estimate of how much filtering can be performed in the Exadata storage cells and, in turn, the amount of CPU that normally would be used to process this data, but now can be offloaded from the database.

Note: Using the Exadata simulation will not result in any plan changes. Execution plans do not change in an Exadata Storage Server installation because the simulation focuses on measuring the improvement in I/O interconnect usage. Moreover, I/O interconnect bytes will not increase, except when data compression is used (see next note), because Oracle Exadata will only decrease the amount of data sent to the database.

Note: Because I/O interconnect bytes is the only metric used to measure the performance change impact of using an Exadata Storage Server installation, it will not work properly if Oracle Exadata is used with data compression. Since Exadata storage cells also decompress data, the I/O interconnect bytes with Oracle Exadata (or the second SQL trial) of a SQL statement may be greater than the I/O interconnect bytes without Oracle Exadata (or the first SQL trial) where the data is compressed. This comparison will be misleading because the SQL statement will be reported as a regression; when in fact, it is not.

Note: The Exadata simulation workflow is used to simulate an Exadata Storage Server installation on non-Exadata hardware. To test changes on Exadata hardware, use the standard SQL Performance Analyzer workflows.

Note: The Exadata simulation is supported for DSS and data warehouse workloads only.

To use the SQL Performance Analyzer Exadata simulation workflow:

1. On the SQL Performance Analyzer page, under SQL Performance Analyzer Workflows, click **Exadata Simulation**.

The Exadata Simulation page appears.

2. In the Task Name field, enter the name of the task.
3. In the SQL Tuning Set field, enter the name of the SQL tuning set that contains the SQL workload to be analyzed.

Alternatively, click the search icon to search for a SQL tuning set using the Search and Select: SQL Tuning Set window.

The selected SQL tuning set now appears in the SQL Tuning Set field.

4. In the Description field, optionally enter a description of the task.
5. In the Per-SQL Time Limit list, determine the time limit for SQL execution during the trial by performing one of the following actions:
 - Select **5 minutes**.
The execution will run each SQL statement in the SQL tuning set up to 5 minutes and gather performance data.
 - Select **Unlimited**.

The execution will run each SQL statement in the SQL tuning set to completion and gather performance data. Collecting execution statistics provides greater accuracy in the performance analysis but takes a longer time. Using this setting is not recommended because the task may be stalled by one SQL statement for a prolonged time period.

- Select **Customize** and enter the specified number of seconds, minutes, or hours.
6. In the Schedule section:
 - a. In the Time Zone list, select your time zone code.
 - b. Select **Immediately** to start the task now, or **Later** to schedule the task to start at a time specified using the Date and Time fields.
 7. Click **Submit**.

The SQL Performance Analyzer page appears.

In the SQL Performance Analyzer Tasks section, the status of this task is displayed. To refresh the status icon, click **Refresh**. After the task completes, the Status field changes to Completed.

SQL Performance Analyzer Tasks								
Delete		View Latest Report						
Select	Name	Owner	Last Modified	Current Step Name	Type	Last Run Status	SQLs Processed	Steps Completed
	SPA_EXADATA_SIM	SYS	May 2, 2012 2:45:21 PM	EXEC_214	Compare	Completed	506 of 506	4 of 4

8. In the SQL Performance Analyzer Tasks section, select the task and click the link in the Name column.

The SQL Performance Analyzer Task page appears.

SQL Performance Analyzer Task: SYS.SPA_EXADATA_SIM

View Latest Report

Page Refreshed May 2, 2012 1:52:41 PM PDT

Refresh

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

SQL Tuning Set


SQL Trials

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

SQL Trial Name	Description	Created	SQL Executed	Status
INITIAL_SQL_TRIAL	Exadata Storage Server simulation disabled	5/2/12 2:44 PM	Yes	COMPLETED
SECOND_SQL_TRIAL	Exadata Storage Server simulation enabled	5/2/12 2:44 PM	Yes	COMPLETED

SQL Trial Comparisons

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

Trial 1 Name	Trial 2 Name	Compare Metric	Created	Status	Comparison Report
INITIAL_SQL_TRIAL	SECOND_SQL_TRIAL	I/O Interconnect Bytes	5/2/12 2:45 PM	COMPLETED	

This page contains the following sections:

- **SQL Tuning Set**
This section summarizes information about the SQL tuning set, including its name, owner, description, and the number of SQL statements it contains.
- **SQL Trials**

This section includes a table that lists the SQL trials used in the SQL Performance Analyzer task.

- SQL Trial Comparisons

This section contains a table that lists the results of the SQL trial comparisons

- Click the icon in the Comparison Report column.

The SQL Performance Analyzer Task Result page appears.

- Review the results of the performance analysis, as described in ["Reviewing the SQL Performance Analyzer Report Using Oracle Enterprise Manager"](#) on page 6-3.

Any SQL performance improvement with the Exadata simulation between the first and second trials is captured in the report. In general, you can expect a greater impact if the SQL workload contains queries that scan a large number of rows or a small subset of table columns. Conversely, a SQL workload that queries indexed tables or tables with fewer rows will result in a lesser impact from the Exadata simulation.

Using the Guided Workflow

The guided workflow enables you to test the performance effect of any types of system changes on a SQL workload, as listed in ["SQL Performance Analyzer"](#) on page 1-1.

Note: To create an analysis task to test database initialization parameter changes, use the simplified parameter change workflow instead, as described in ["Using the Parameter Change Workflow"](#) on page 3-3.

To use the SQL Performance Analyzer task guided workflow:

- On the SQL Performance Analyzer page, under SQL Performance Analyzer Workflows, click **Guided Workflow**.

The Guided Workflow page appears.

The guided workflow enables you to test the performance effect on a SQL workload when you perform any type of system changes, as described in ["SQL Performance Analyzer"](#) on page 1-1.

This page lists the required steps in the SQL Performance Analyzer task in sequential order. Each step must be completed in the order displayed before the next step can begin.

Guided Workflow
Page Refreshed May 2, 2012 2:52:20 PM PDT
Refresh
View Data
Real Time: Manual Refresh

The following guided workflow contains the sequence of steps necessary to execute a successful two-trial SQL Performance Analyzer test.
 Note: Be sure that the Trial environment matches the tests you want to conduct.

Step	Description	Executed	Status	Execute
1	Create SQL Performance Analyzer Task based on SQL Tuning Set		■	
2	Create SQL Trial in Initial Environment		■	
3	Create SQL Trial in Changed Environment		■	
4	Compare Step 2 and Step 3		■	
5	View Trial Comparison Report		■	

TIP For an explanation of the icons and symbols used in the following table, see the Icon Key

2. On the Guided Workflow page, click the **Execute** icon for the Step 1: Create SQL Performance Analyzer Task based on SQL Tuning Set.

The Create SQL Performance Analyzer Task page appears.

3. In the Name field, enter the name of the task.
4. In the Description field, optionally enter a description of the task.
5. Under SQL Tuning Set, in the Name field, enter the name the SQL tuning set that contains the SQL workload to be analyzed.

Alternatively, click the search icon to select a SQL tuning set from the Search and Select: SQL Tuning Set window.

6. Click **Create**.

The Guided Workflow page appears.

The Status icon of this step has changed to a check mark and the **Execute** icon for the next step is now enabled.

7. Once the analysis task is created, you can build the pre-change performance data by executing the SQL statements stored in the SQL tuning set, as described in [Chapter 4, "Creating a Pre-Change SQL Trial"](#).

Creating an Analysis Task Using APIs

This section describes how to create a SQL Performance Analyzer task by using the `DBMS_SQLPA.CREATE_ANALYSIS_TASK` function. A task is a database container for SQL Performance Analyzer execution inputs and results.

Tip: Before proceeding, capture the SQL workload to be used in the performance analysis into a SQL tuning set on the production system, then transport it to the test system where the performance analysis will be performed, as described in ["Capturing the SQL Workload"](#) on page 2-3.

To create an analysis task:

- Call the `CREATE_ANALYSIS_TASK` function using the following parameters:
 - Set `task_name` to specify an optional name for the SQL Performance Analyzer task.
 - Set `sqlset_name` to the name of the SQL tuning set.

- Set `sqlset_owner` to the owner of the SQL tuning set. The default is the current schema owner.
- Set `basic_filter` to the SQL predicate used to filter the SQL from the SQL tuning set.
- Set `order_by` to specify the order in which the SQL statements will be executed.

You can use this parameter to ensure that the more important SQL statements will be processed and not skipped if the time limit is reached.

- Set `top_sql` to consider only the top number of SQL statements after filtering and ranking.

The following example illustrates a function call:

```
VARIABLE t_name VARCHAR2(100);
EXEC :t_name := DBMS_SQLPA.CREATE_ANALYSIS_TASK(sqlset_name => 'my_sts', -
        task_name => 'my_spa_task');
```

Once the analysis task is created, you can build the pre-change performance data by executing the SQL statements stored in the SQL tuning set, as described in [Chapter 4, "Creating a Pre-Change SQL Trial"](#).

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* to learn more about the `DBMS_SQLPA.CREATE_ANALYSIS_TASK` function

Running the Exadata Simulation Using APIs

This section describes how to run the Oracle Exadata simulation using APIs. For information about how SQL Performance Analyzer simulates the effects of an Exadata Storage Server installation on the performance of a SQL workload, see ["Using the Exadata Simulation Workflow"](#) on page 3-9.

To enable Exadata simulation for an analysis task:

- Call the `SET_ANALYSIS_TASK_PARAMETER` procedure before creating the post-change SQL trial, as shown in the following example:

```
EXEC DBMS_SQLPA.SET_ANALYSIS_TASK_PARAMETER(task_name => 'my_spa_task', -
        parameter => 'CELL_SIMULATION_ENABLED', -
        value => 'TRUE');
```

This will enable Exadata simulation when you create the post-change SQL trial, which can then be compared to the pre-change SQL trial that was created with Exadata simulation disabled.

Alternatively, you can run the Exadata simulation using the `tcellsim.sql` script.

To run the Exadata simulation using `tcellsim.sql`:

1. At the SQL prompt, enter:

```
@$ORACLE_HOME/rdbms/admin/tcellsim.sql
```

2. Enter the name and owner of the SQL tuning set to use:

```
Enter value for sts_name: MY_STS
Enter value for sts_owner: IMMCHAN
```

The script then runs the following four steps automatically:

- Creates a SQL Performance Analyzer task
- Test executes SQL statements with Exadata simulation disabled
- Test executes SQL statements with Exadata simulation enabled
- Compares performance and generates analysis report

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_SQLPA.SET_ANALYSIS_TASK_PARAMETER` procedure

Remapping Multitenant Container Database Identifiers in an Analysis Task

You can store captured SQL statements in a SQL tuning set, and use it as an input source when creating a SQL Performance Analyzer task. SQL Performance Analyzer then uses the SQL tuning set as the source for test executing or generating execution plans for SQL trials.

If you use a SQL tuning set that was transported from a non-CDB to a multitenant container database (CDB) as the input source, the CDB identifiers of the SQL statements in the SQL tuning set must be remapped to make the STS usable in the CDB. Remapping CDB identifiers associates each SQL statement in the SQL tuning set with a CDB identifier that can be remapped to the corresponding pluggable databases (PDBs) within the CDB.

Typically, CDB identifiers should be remapped when the SQL tuning set is transported from a non-CDB to a CDB. In this case, you can simply use the SQL tuning set as an input source for SQL Performance Analyzer. However, if you are using a SQL tuning set whose CDB identifiers have not been remapped, you can specify the remapping as a SQL Performance Analyzer task property.

To remap CDB identifiers for an analysis task:

- Use the `SET_ANALYSIS_TASK_PARAMETER` procedure, as shown in the following example:

```
EXEC DBMS_SQLPA.SET_ANALYSIS_TASK_PARAMETER(task_name => 'non_cdb_spa1', -
      parameter => 'CON_DBID_MAPPING', -
      value => '1234:5678,1357:2468');
```

In this example, the CDB identifiers 1234 and 1357 are remapped to 5678 and 2468, respectively.

After the CDB identifiers are remapped, SQL Performance Analyzer uses the new CDB identifier when it finds a match for the old CDB identifier, and executes the SQL statements in the appropriate PDB within the CDB.

See Also:

- *Oracle Database SQL Tuning Guide* for information about transporting SQL tuning sets
- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_SQLPA.SET_ANALYSIS_TASK_PARAMETER` procedure

Creating a Pre-Change SQL Trial

After creating a SQL Performance Analyzer task and selecting a SQL tuning set as the input source, you need to establish the initial environment on the test system. Establishing the database environment on the test system involves manually making any necessary environmental changes that affect SQL optimization and performance. These changes may include changing initialization parameters, gathering or setting optimizer statistics, and creating indexes. It is recommended that you build a test system that is as similar to the production system as possible. The dedicated workflows in Enterprise Manager simplifies this process by creating both SQL trials automatically and performing the change restricted to the testing session. For information about setting up the database environment, see ["Setting Up the Test System"](#) on page 2-4.

Note: You can optionally run SQL trials on a remote system by providing access to a public database link. When conducting remote SQL trials, the database version of the remote database where the SQL statements are executed must be less than or equal to the database version of the database to which it connects. Starting with Oracle Database release 11.2.0.2, the remote database can be a read-only database, such as an Oracle Active Data Guard instance.

Once the environment on the test system is properly configured, you can build the pre-change version of performance data before performing the system change. You can build SQL trials using SQL Performance Analyzer by using one of the following methods:

- Executing the SQL statements in the workload
- Generating execution plans for the SQL statements in the workload
- Loading performance data and execution plans from a SQL tuning set (APIs only)

For more information, see ["Measuring the Pre-Change SQL Performance"](#) on page 2-5

This chapter described how to create the pre-change SQL trial and contains the following topics:

- [Creating a Pre-Change SQL Trial Using Enterprise Manager](#)
- [Creating a Pre-Change SQL Trial Using APIs](#)

Note: The primary interface for creating a pre-change SQL trial is Oracle Enterprise Manager. If for some reason Oracle Enterprise Manager is unavailable, you can create a pre-change SQL trial using the DBMS_SQLPA PL/SQL package.

Tip: Before creating a pre-change SQL trial, you need to create a SQL Performance Analyzer task, as described in [Chapter 3, "Creating an Analysis Task"](#).

Creating a Pre-Change SQL Trial Using Enterprise Manager

This section describes how to collect the pre-change SQL performance data using Oracle Enterprise Manager.

To create a pre-change SQL trial using Enterprise Manager:

1. On the Guided Workflow page, click the **Execute** icon for the Create SQL Trial in Initial Environment step.

The Create SQL Trial page appears. A summary of the selected SQL tuning set containing the SQL workload is displayed.

Create SQL Trial

SQL Trials capture execution performance of the SQL Tuning Set under a given optimizer environment.
 SQL Performance Analyzer Task: SYS.SPA_GUIDED_WORKFLOW
 SQL Tuning Set: SYS.DOCTEST3

* SQL Trial Name:

SQL Trial Description:

Creation Method:

Per-SQL Time Limit:
 TIP Time limit is on elapsed time of test execution of SQL.

Schedule

Time Zone:

☒ Immediately ☐ Later

Date:
 (example: May 2, 2012)

Time: : :
☐ AM ☒ PM

Trial environment determines results

The SQL Tuning Set remains constant under the SQL Performance Analyzer Task and its SQL is executed in isolation to create each SQL Trial. Performance differences between trials are thus attributed to environmental differences between trials.

Environmental changes affecting SQL optimization and performance may need to be made manually prior to execution of the Trial. These could include changing initialization parameters, gathering or setting optimizer statistics and creating indexes.

The Creation Method determines how the SQL Trial is created and what contents are generated, as follows:

- Executing SQLs generates both plans and statistics by actually running the SQL statements.
- Generating plans invokes the optimizer to create execution plans only without running the SQL statements.
- Remote execution and plan generation are done over a public database link on the remote system.
- Building from the SQL Tuning Set simply copies the plans and statistics from the Tuning Set directly into the Trial.

NOTE: Be sure trial environment has been established prior to submitting.

☐ Trial environment established

2. In the SQL Trial Name field, enter the name of the SQL trial.
3. In the SQL Trial Description field, enter a description of the SQL trial.
4. In the Creation Method list, determine how the SQL trial is created and what contents are generated by performing one of the following actions:
 - **Select Execute SQLs Locally.**
 The SQL trial generates both execution plans and statistics for each SQL statement in the SQL tuning set by actually running the SQL statements locally on the test system.
 - **Select Execute SQLs Remotely.**
 The SQL trial generates both execution plans and statistics for each SQL statement in the SQL tuning set by actually running the SQL statements remotely on another test system over a public database link.
 - **Select Generate Plans Locally.**

The SQL trial invokes the optimizer to create execution plans locally on the test system, after taking bind values and optimizer configuration into account, without actually running the SQL statements.

- Select **Generate Plans Remotely**.

The SQL trial invokes the optimizer to create execution plans remotely on another test system, after taking bind values and optimizer configuration into account, over a public database link without actually running the SQL statements.

- Select **Build From SQL Tuning Set**.

The SQL trial copies the execution plans and statistics from the SQL tuning set directly into the trial.

For more information about the different methods, see ["Measuring the Pre-Change SQL Performance"](#) on page 2-5.

5. In the Per-SQL Time Limit list, determine the time limit for SQL execution during the trial by performing one of the following actions:

- Select **5 minutes**.

The execution will run each SQL statement in the SQL tuning set up to 5 minutes and gather performance data.

- Select **Unlimited**.

The execution will run each SQL statement in the SQL tuning set to completion and gather performance data. Collecting execution statistics provides greater accuracy in the performance analysis but takes a longer time. Using this setting is not recommended because the task may be stalled by one SQL statement for a prolonged period.

- Select **Customize** and enter the specified number of seconds, minutes, or hours.

6. Ensure that the database environment on the test system matches the production environment as closely as possible, and select **Trial environment established**.

7. In the Schedule section:

- a. In the Time Zone list, select your time zone code.

- b. Select **Immediately** to start the task now, or **Later** to schedule the task to start at a time specified using the Date and Time fields.

8. Click **Submit**.

The Guided Workflow page appears when the execution begins.

The status icon of this step changes to a clock while the execution is in progress. To refresh the status icon, click **Refresh**. Depending on the options selected and the size of the SQL workload, the execution may take a long time to complete. After the execution is completed, the Status icon will change to a check mark and the Execute icon for the next step is enabled.

9. Once the pre-change performance data is built, you can make the system change and build the post-change performance data by re-executing the SQL statements in the SQL tuning set on the post-change test system, as described in [Chapter 5, "Creating a Post-Change SQL Trial"](#).

Creating a Pre-Change SQL Trial Using APIs

This section describes how to build the pre-change performance data by using the DBMS_SQLPA package.

To create a pre-change SQL trial:

- Call the EXECUTE_ANALYSIS_TASK procedure using the following parameters:
 - Set the `task_name` parameter to the name of the SQL Performance Analyzer task that you want to execute.
 - Set the `execution_type` parameter in one of the following ways:
 - * Set to `EXPLAIN PLAN` to generate execution plans for all SQL statements in the SQL tuning set without executing them.
 - * Set to `TEST EXECUTE` (recommended) to execute all statements in the SQL tuning set and generate their execution plans and statistics. When `TEST EXECUTE` is specified, the procedure generates execution plans and execution statistics. The execution statistics enable SQL Performance Analyzer to identify SQL statements that have improved or regressed. Collecting execution statistics in addition to generating execution plans provides greater accuracy in the performance analysis, but takes longer.
 - * Set to `CONVERT SQLSET` to refer to a SQL tuning set for the execution statistics and plans for the SQL trial. Values for the execution parameters `SQLSET_NAME` and `SQLSET_OWNER` should also be specified.
 - Specify a name to identify the execution using the `execution_name` parameter. If not specified, then SQL Performance Analyzer automatically generates a name for the task execution.
 - Specify execution parameters using the `execution_params` parameters. The `execution_params` parameters are specified as (*name, value*) pairs for the specified execution. For example, you can set the following execution parameters:
 - * The `time_limit` parameter specifies the global time limit to process all SQL statements in a SQL tuning set before timing out.
 - * The `local_time_limit` parameter specifies the time limit to process each SQL statement in a SQL tuning set before timing out.
 - * To perform a remote test execute, set the `DATABASE_LINK` task parameter to the global name of a public database link connecting to a user with the `EXECUTE` privilege for the DBMS_SQLPA package and the `ADVISOR` privilege on the test system.
 - * To fully execute DML statements—including acquiring row locks and modifying row—set the `EXECUTE_FULLDML` parameter to `TRUE`. SQL Performance Analyzer will issue a rollback after executing the DML statements to prevent persistent changes from being made. The default value for this parameter is `FALSE`, which executes only the query portion of the DML statement without modifying the data.
 - * To restore the relevant captured `init.ora` settings during a test execute, set the `APPLY_CAPTURED_COMPILEENV` parameter to `TRUE`. This is not the default behavior because typically you are running SQL trials to test changes when changing the environment. However, this method may be used in cases when the `init.ora` settings are not being changed (such as creating an index). This method is not supported for remote SQL trials.

The following example illustrates a function call made *before* a system change:

```
EXEC DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(task_name => 'my_spa_task', -  
    execution_type => 'TEST EXECUTE', -  
    execution_name => 'my_exec_BEFORE_change');
```

Once the pre-change performance data is built, you can make the system change and build the post-change performance data by re-executing the SQL statements in the SQL tuning set on the post-change test system, as described in [Chapter 5, "Creating a Post-Change SQL Trial"](#).

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* to learn about the `DBMS_SQLPA.EXECUTE_ANALYSIS_TASK` function

Creating a Post-Change SQL Trial

After computing the pre-change SQL performance data, you can perform the system change on the test system. Before making the system change, ensure that you have executed the SQL workload in the initial environment to generate the pre-change performance data. For example, if you are testing how changing a database initialization parameter will affect SQL performance, execute the SQL workload once before changing the database initialization parameter to a new value. Depending on the type of change you are making, it may be necessary to reconfigure the environment on the test system to match the new environment for which you want to perform SQL performance analysis. For more information, see ["Making a System Change"](#) on page 2-7.

Note: You can optionally run SQL trials on a remote system by providing access to a public database link. When conducting remote SQL trials, the database version of the remote database where the SQL statements are executed must be less than or equal to the database version of the database to which it connects. Starting with Oracle Database release 11.2.0.2, the remote database can be a read-only database, such as an Oracle Active Data Guard instance.

["SQL Performance Analyzer"](#) on page 1-1 lists examples of possible system changes that can be analyzed using SQL Performance Analyzer. For example, you may want to determine how a database initialization parameter change or database upgrade will affect SQL performance. You may also decide to change the system based on recommendations from an advisor such as Automatic Database Diagnostic Monitor (ADDM), SQL Tuning Advisor, or SQL Access Advisor.

After you have made the system change, you can build the post-change version of performance data by executing the SQL workload again. SQL Performance Analyzer will store the results from executing the SQL statements in a post-change SQL trial. For more information, see ["Measuring the Post-Change SQL Performance"](#) on page 2-7.

This section describes how to create the post-change SQL trial and contains the following topics:

- [Creating a Post-Change SQL Trial Using Oracle Enterprise Manager](#)
- [Creating a Post-Change SQL Trial Using APIs](#)

Note: The primary interface for creating a post-change SQL trial is Oracle Enterprise Manager. If for some reason Oracle Enterprise Manager is unavailable, you can create a post-change SQL trial using the `DBMS_SQLPA` PL/SQL package.

Tip: Before making the system change creating a post-change SQL trial, you need to create a pre-change SQL trial, as described in [Chapter 4, "Creating a Pre-Change SQL Trial"](#).

Creating a Post-Change SQL Trial Using Oracle Enterprise Manager

This section describes how to collect the post-change SQL performance data using Oracle Enterprise Manager.

To create a post-change SQL trial using Enterprise Manager:

1. On the Guided Workflow page, click the **Execute** icon for the Create SQL Trial in Changed Environment step.

The Create SQL Trial page appears.

2. In the SQL Trial Name field, enter the name of the SQL trial.
3. In the SQL Trial Description field, enter a description of the SQL trial.
4. In the Creation Method list, determine how the SQL trial is created and what contents are generated by performing one of the following actions:

- Select **Execute SQLs Locally**.

The SQL trial generates both execution plans and statistics for each SQL statement in the SQL tuning set by actually running the SQL statements locally on the test system.

- Select **Execute SQLs Remotely**.

The SQL trial generates both execution plans and statistics for each SQL statement in the SQL tuning set by actually running the SQL statements remotely on another test system over a public database link.

- Select **Generate Plans Locally**.

The SQL trial invokes the optimizer to create execution plans locally on the test system without actually running the SQL statements.

- Select **Generate Plans Remotely**.

The SQL trial invokes the optimizer to create execution plans remotely on another test system over a public database link without actually running the SQL statements.

For each of these creation methods, the application schema and data should already exist on the local or remote test system.

5. In the Per-SQL Time Limit list, determine the time limit for SQL execution during the trial by performing one of the following actions:

- Select **5 minutes**.

The execution will run each SQL statement in the SQL tuning set up to 5 minutes and gather performance data.

- Select **Unlimited**.
The execution will run each SQL statement in the SQL tuning set to completion and gather performance data. Collecting execution statistics provides greater accuracy in the performance analysis but takes a longer time. Using this setting is not recommended because the task may be stalled by one SQL statement for a prolonged time period.
 - Select **Customize** and enter the specified number of seconds, minutes, or hours.
6. Ensure that the system change you are testing has been performed on the test system, and select **Trial environment established**.
 7. In the Schedule section:
 - a. In the Time Zone list, select your time zone code.
 - b. Select **Immediately** to start the task now, or **Later** to schedule the task to start at a time specified using the Date and Time fields.
 8. Click **Submit**.
The Guided Workflow page appears when the execution begins.

The status icon of this step changes to a clock while the execution is in progress. To refresh the status icon, click **Refresh**. Depending on the options selected and the size of the SQL workload, the execution may take a long time to complete. After the execution is completed, the Status icon will change to a check mark and the Execute icon for the next step is enabled.
 9. Once the post-change performance data is built, you can compare the pre-change SQL trial to the post-change SQL trial by running a comparison analysis, as described in [Chapter 6, "Comparing SQL Trials"](#).

Creating a Post-Change SQL Trial Using APIs

This section describes how to collect the post-change SQL performance data using the DBMS_SQLPA package.

Note: If you are running the SQL statements remotely on another test system over a database link, the remote user calling this procedure needs to have the EXECUTE privilege for the DBMS_SQLPA package.

To create a post-change SQL trial:

- Call the EXECUTE_ANALYSIS_TASK procedure using the parameters described in ["Creating a Pre-Change SQL Trial Using APIs"](#) on page 4-4.

Be sure to specify a different value for the execution_name parameter. It is also highly recommended that you create the post-change SQL trial using the same method as the pre-change SQL trial by using the same value for the execution_type parameter.

Note: If you want to run an Oracle Exadata simulation, you should first set the CELL_SIMULATION_ENABLED task parameter to TRUE. For more information, see ["Running the Exadata Simulation Using APIs"](#) on page 3-14.

The following example illustrates a function call made after a system change:

```
EXEC DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(task_name => 'my_spa_task', -  
    execution_type => 'TEST EXECUTE', -  
    execution_name => 'my_exec_AFTER_change');
```

Once the post-change performance data is built, you can compare the pre-change SQL trial to the post-change SQL trial by running a comparison analysis, as described in [Chapter 6, "Comparing SQL Trials"](#).

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* to learn about the `DBMS_SQLPA.EXECUTE_ANALYSIS_TASK` function

Comparing SQL Trials

After the post-change SQL performance data is built, you can compare the performance data collected in the pre-change SQL trial to the post-change SQL trial by running a comparison analysis using SQL Performance Analyzer. After the comparison analysis is completed, you can generate a report to identify the SQL statements that have improved, remained unchanged, or regressed due to the system change. The SQL Performance Analyzer report calculates two chief impact measurements for the change in performance of each SQL statement:

- **Impact on workload**

This represents the percentage of impact that this change to the SQL statement has on the cumulative execution time of the workload, after accounting for execution frequency. For example, a change that causes a SQL statement's cumulative execution time to improve from 101 seconds to 1 second—where the rest of the workload had a total execution time of 99 seconds before the change—would have a 50% (2x) value for this measurement.

- **Impact on SQL**

This represents the percentage of impact that this change to the SQL statement has on the SQL statement's response time. For example, a change that causes a SQL statement's response time to improve from 10 seconds to 1 second will have a 90% (10x) value for this measurement.

For more information, see ["Comparing Performance Measurements"](#) on page 2-7.

This chapter describes how to compare and analyze the performance data from the pre-change and post-change SQL trials and contains the following topics:

- [Comparing SQL Trials Using Oracle Enterprise Manager](#)
- [Comparing SQL Trials Using APIs](#)

Note: The primary interface for comparing SQL trials is Oracle Enterprise Manager. If for some reason Oracle Enterprise Manager is unavailable, you can compare SQL trials using the `DBMS_SQLPA` PL/SQL package.

Tip: Before comparing SQL trials, you need to create a post-change SQL trial, as described in [Chapter 5, "Creating a Post-Change SQL Trial"](#).

Comparing SQL Trials Using Oracle Enterprise Manager

Comparing SQL trials using Oracle Enterprise Manager involves the following steps:

- [Analyzing SQL Performance Using Oracle Enterprise Manager](#)
- [Reviewing the SQL Performance Analyzer Report Using Oracle Enterprise Manager](#)
- [Tuning Regressed SQL Statements Using Oracle Enterprise Manager](#)

Analyzing SQL Performance Using Oracle Enterprise Manager

This section describes how to analyze SQL performance before and after the system change using Oracle Enterprise Manager.

To analyze SQL performance using Enterprise Manager:

1. On the Guided Workflow page, click the **Execute** icon for Compare Step 2 and Step 3.

The Run SQL Trial Comparison page appears.

Run SQL Trial Comparison

Task Name: SYS.SPA_GUIDED_WORKFLOW
SQL Tuning Set: SYS.DOCTEST3

Trial 1 Name:
Description:
SQL Executed: Yes

Trial 2 Name:
Description:
SQL Executed: Yes

Comparison Metric:

Schedule

Time Zone:

☒ Immediately
☐ Later

Date:
(example: May 2, 2012)

Time: ☐ AM ☒ PM

Compare trials to assess change impact

SQL Performance Analyzer trial comparison allows you to assess the impact on SQL Tuning Set performance of changes made between two trials.

It is important to know the difference between Trial 1 and Trial 2 execution environments in order to properly assign impacts to the changes between trials. Tracking environmental changes between trials is currently a user responsibility.

The selected comparison metric is used as the basis for comparison, and defaults to execute elapsed time when both trials contain test execution statistics. When execution statistics are not available, a less accurate comparison can be made using optimizer cost.

In this example, the SQL_TRIAL_1241213421833 and SQL_TRIAL_1241213881923 trials are selected for comparison.

2. To compare trials other than those listed by default, select the desired trials in the **Trial 1 Name** and **Trial 2 Name** lists.

Note that you cannot compare a statistical trial with a trial that tests the explain plan only.

3. In the Comparison Metric list, select the comparison metric to use for the comparison analysis:
 - **Elapsed Time**
 - **CPU Time**
 - **User I/O Time**
 - **Buffer Gets**

- **Physical I/O**
- **Optimizer Cost**
- **I/O Interconnect Bytes**

Optimizer Cost is the only comparison metric available if you generated execution plans only in the SQL trials.

To perform the comparison analysis by using more than one comparison metric, perform separate comparison analyses by repeating this procedure with different metrics.

4. In the Schedule section:
 - a. In the Time Zone list, select your time zone code.
 - b. Select **Immediately** to start the task now, or **Later** to schedule the task to start at a time specified using the Date and Time fields.
5. Click **Submit**.

The Guided Workflow page appears when the comparison analysis begins.

The status icon of this step changes to an arrow icon while the comparison analysis is in progress. To refresh the status icon, click **Refresh**. Depending on the amount of performance data collected from the pre-change and post-change executions, the comparison analysis may take a long time to complete. After the comparison analysis is completed, the Status icon changes to a check mark and the Execute icon for the next step is enabled.

6. Once SQL Performance Analyzer has analyzed the pre-change and post-change performance data, generate a SQL Performance Analyzer report that you can use for further analysis.

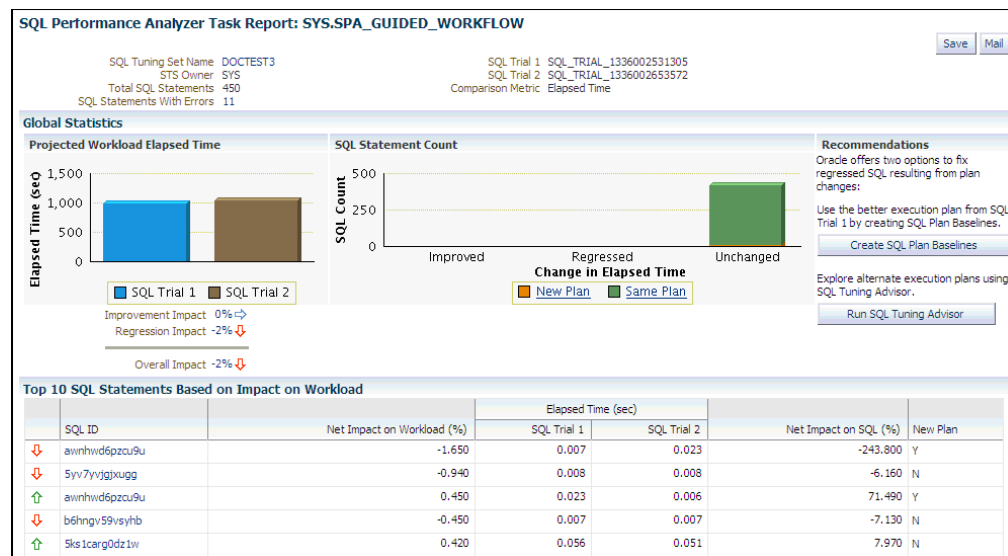
On the Guided Workflow page, click the **Execute** icon for View Trial Comparison Report.

The SQL Performance Analyzer Task Report page appears. Review the report, as described in ["Reviewing the SQL Performance Analyzer Report Using Oracle Enterprise Manager"](#) on page 6-3.

Reviewing the SQL Performance Analyzer Report Using Oracle Enterprise Manager

When a SQL Performance Analyzer task is completed, the resulting data is generated into a SQL Performance Analyzer report that compares the pre-change and post-change SQL performance.

[Figure 6-1](#) shows a sample SQL Performance Analyzer report. This sample report uses the elapsed time comparison metric to compare the pre-change and post-change executions of a SQL workload.

Figure 6–1 SQL Performance Analyzer Report

Tip: Before you can view the SQL Performance Analyzer report, compare the pre-change version of performance data with the post-change version, as described in ["Comparing SQL Trials Using Oracle Enterprise Manager"](#) on page 6-2

To generate and review the SQL Performance Analyzer report:

1. From the **Performance** menu, select **SQL**, then **SQL Performance Analyzer**.

If the Database Login page appears, then log in as a user with administrator privileges.

The SQL Performance Analyzer page appears. A list of existing SQL Performance Analyzer tasks are displayed.

2. Under SQL Performance Analyzer Tasks, select the task for which you want to view a SQL Performance Analyzer report and click **View Latest Report**.

The SQL Performance Analyzer Task Report page appears.

3. Review the general information about the performance analysis, as described in ["Reviewing the SQL Performance Analyzer Report: General Information"](#) on page 6-5.
4. Review general statistics, as described in ["Reviewing the SQL Performance Analyzer Report: Global Statistics"](#) on page 6-5.
5. Optionally, review the detailed statistics, as described in ["Reviewing the SQL Performance Analyzer Report: Global Statistics Details"](#) on page 6-6.
6. To generate an active report, click **Save** to generate and save the report, or **Mail** to generate and mail the report as an HTML attachment.

Active reports include information about the top SQL statements from each category (such as improved, regressed, and changed plans) with pre-change and post-change statistics, explain plans, and task summary.

For more information, see ["About SQL Performance Analyzer Active Reports"](#) on page 6-7.

Reviewing the SQL Performance Analyzer Report: General Information

The General Information section contains basic information and metadata about the workload comparison performed by SQL Performance Analyzer.

To review general information:

1. On the SQL Performance Analyzer Task Report page, review the summary at the top of the page.

SQL Tuning Set Name	DOCTEST3	SQL Trial 1	INITIAL_SQL_TRIAL
STS Owner	SYS	SQL Trial 2	SECOND_SQL_TRIAL
Total SQL Statements	450	Comparison Metric	I/O Interconnect Bytes
SQL Statements With Errors	11		

This summary includes the following information:

- The name and owner of the SQL tuning set
 - The total number of SQL statements in the tuning set and the number of SQL statements that had errors, are unsupported, or timed out
 - The names of the SQL trials and the comparison metric used
2. Optionally, click the link next to SQL Tuning Set Name.

The SQL Tuning Set page appears.

This page contains information—such as SQL ID and SQL text—about every SQL statement in the SQL tuning set.

3. Click the link next to SQL Statements With Errors if errors were found.

The Errors table reports all errors that occurred while executing a given SQL workload. An error may be reported at the SQL tuning set level if it is common to all SQL executions in the SQL tuning set, or at the execution level if it is specific to a SQL statement or execution plan.

4. Review the global statistics, as described in ["Reviewing the SQL Performance Analyzer Report: Global Statistics"](#) on page 6-5.

Reviewing the SQL Performance Analyzer Report: Global Statistics

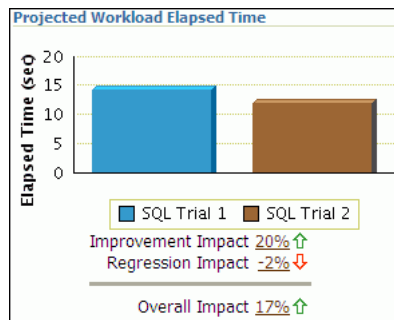
The Global Statistics section reports statistics that describe the overall performance of the entire SQL workload. This section is a very important part of the SQL Performance Analyzer analysis, because it reports on the impact of the system change on the overall performance of the SQL workload. Use the information in this section to understand the tendency of the workload performance, and determine how it will be affected by the system change.

To review global statistics:

1. Review the chart in the Projected Workload Elapsed Time subsection.

Note: The name of the subsection may vary based on the comparison metric that is selected.

The chart shows the two trials on the x-axis and the elapsed time (in seconds) on the y-axis.



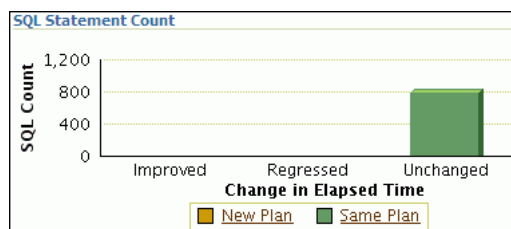
The most important statistic is the overall impact, which is given as a percentage. The overall impact is the difference between the improvement impact and the regression impact. You can click the link for any impact statistic to obtain more details, as described in ["Reviewing the SQL Performance Analyzer Report: Global Statistics Details"](#) on page 6-6.

In this example, the improvement impact is 20%, while the regression impact is -2%, so the overall impact of the system change is an improvement of approximately 18%. This means that if all regressions are fixed in this example, the overall impact of the change will be an improvement of 20%.

Note: The overall impact percentage may sometimes be off by 1% compared to the sum of the improvement impact and the regression impact. This discrepancy may be caused by rounding or if the SQL and workload time limits are set at 1%, which is the recommended value. This enables the analysis to focus on SQL statements with higher impact by filtering out those that have a minimal impact.

2. Review the chart in the SQL Statement Count subsection.

The x-axis of the chart shows the number of SQL statements whose performance improved, regressed, or remain unchanged after the system change. The y-axis shows the number of SQL statements. The chart also indicates whether the explain plans changed for the SQL statements.



This chart enables you to quickly weigh the relative performance of the SQL statements. You can click any bar in the chart to obtain more details about the SQL statements, as described in ["Reviewing the SQL Performance Analyzer Report: Global Statistics Details"](#) on page 6-6. Only up to the top 100 SQL statements will be displayed, even if the actual number of SQL statements exceeds 100.

In this example, all SQL statements were unchanged after the system change.

Reviewing the SQL Performance Analyzer Report: Global Statistics Details

You can use the SQL Performance Analyzer Report to obtain detailed statistics for the SQL workload comparison. The details chart enables you to drill down into the

performance of SQL statements that appears in the report. Use the information in this section to investigate why the performance of a particular SQL statement regressed.

Note: The report displays only up to the top 100 SQL statements, even if the actual number of SQL statements exceeds 100.

To review global statistics details:

1. In the Projected Workload Elapsed Time subsection, click the impact percentage of the SQL statements for which you want to view details. To view SQL statements whose performance:

- Improved, click the percentage for Improvement Impact
- Regressed, click the percentage for Regression Impact
- Improved or regressed, click the percentage for Overall Impact

A table including the detailed statistics appears. Depending on the type of SQL statements chosen, the following columns are included:

- SQL ID

This column indicates the ID of the SQL statement.

- Net Impact on Workload (%)

This column indicates the impact of the system change relative to the performance of the SQL workload.

- Elapsed Time

This column indicates the total time (in seconds) of the SQL statement execution.

- Net Impact on SQL (%)

This column indicates the local impact of the change on the performance of a particular SQL statement.

- New Plan

This column indicates whether the SQL execution plan changed.

2. To view details about a particular SQL statement, click the SQL ID link for the SQL statement that you are interested in.

The SQL Details page appears.

You can use this page to access the SQL text and obtain low-level details about the SQL statement, such as its execution statistics and execution plan.

About SQL Performance Analyzer Active Reports

SQL Performance Analyzer active reports are HTML files that display all reporting data using a Web-hosted interactive user interface. Similar to the SQL Performance Analyzer reports available in Oracle Enterprise Manager, active reports include information about the top SQL statements from each category (such as improved, regressed, and changed plans) with pre-change and post-change statistics, explain plans, and task summary.

SQL Performance Analyzer active reports are more useful than traditional HTML or text reports because they offer a similar user interface as Oracle Enterprise Manager, yet they can be viewed even when the database is unavailable, or even after a database

is dropped. Hence active reports offer the advantages of traditional reporting and dynamic Oracle Enterprise Manager analysis, but eliminates the disadvantages of both. Moreover, active reports contain more information about the comparison analysis and provide more user interactive options. It is strongly recommended that you use active reports instead of HTML or text reports.

The active report user interface components are very similar to those displayed in Oracle Enterprise Manager. For descriptions of the user interface components, see the related sections described in ["Reviewing the SQL Performance Analyzer Report Using Oracle Enterprise Manager"](#) on page 6-3.

Tuning Regressed SQL Statements Using Oracle Enterprise Manager

After reviewing the SQL Performance Analyzer report, you should tune any regressed SQL statements that are identified after comparing the SQL performance. If there are large numbers of SQL statements that appear to have regressed, you should try to identify the root cause and make system-level changes to rectify the problem. In cases when only a few SQL statements have regressed, consider using one of the following tuning methods to implement a point solution for them:

- [Creating SQL Plan Baselines](#)
- [Running SQL Tuning Advisor](#)

After tuning the regressed SQL statements, you should test these changes using SQL Performance Analyzer. Run a new SQL trial on the test system, followed by a second comparison (between this new SQL trial and the first SQL trial) to validate your results. Once SQL Performance Analyzer shows that performance has stabilized, the testing is complete. Implement the fixes from this step to your production system.

Starting with Oracle Database 11g Release 1, SQL Tuning Advisor performs an alternative plan analysis when tuning a SQL statement. SQL Tuning Advisor searches the current system for previous execution plans, including the plans from the first SQL trial. If the execution plans from the first SQL trial differ from those of the second SQL trial, SQL Tuning Advisor will recommend the plans from the first SQL trial. If these execution plans produce better performance, you can create plan baselines using the plans from the first SQL trial.

Note: SQL Performance Analyzer does not provide the option to create SQL plan baselines or run SQL Tuning Advisor directly after completing a remote SQL trial. In such cases, you need to use APIs to manually transport the SQL tuning set and complete the appropriate procedure on the remote database.

See Also:

- [Oracle Database SQL Tuning Guide](#) for information about alternative plan analysis

Creating SQL Plan Baselines

Creating SQL plan baselines enables the optimizer to avoid performance regressions by using execution plans with known performance characteristics. If a performance regression occurs due to plan changes, a SQL plan baseline can be created and used to prevent the optimizer from picking a new, regressed execution plan.

To create SQL plan baselines:

1. On the SQL Performance Analyzer Task Result page, under Recommendations, click **Create SQL Plan Baselines**.

The Create SQL Plan Baselines page appears. The Regressed SQL Statements section lists the regressed SQL statements that will be associated with the new SQL plan baselines.

2. Under Job Parameters, specify the parameters for the job:
 - a. In the Job Name field, enter a name for the job.
 - b. In the Description field, optionally enter a description for the job.
3. Under Schedule, select:
 - **Immediately** to start the job now.
 - **Later** to schedule the job to start at a time specified using the Time Zone, Date, and Time fields.
4. Click **OK**.

The SQL Performance Analyzer Task Result page appears. A message is displayed to inform you that the job has been submitted successfully.

See Also:

- *Oracle Database 2 Day + Performance Tuning Guide* for information about creating and managing SQL plan baselines

Running SQL Tuning Advisor

The SQL Tuning Advisor performs an in-depth analysis of regressed SQL statements and attempts to fix the root cause of the problem.

To run SQL Tuning Advisor:

1. On the SQL Performance Analyzer Task Result page, under Recommendations, click **Run SQL Tuning Advisor**.

The Schedule SQL Tuning Task page appears.

2. In the Tuning Task Name field, enter a name for the SQL tuning task.
3. In the Tuning Task Description field, optionally enter a name for the SQL tuning task.
4. Under Schedule, select:
 - **Immediately** to start the job now.
 - **Later** to schedule the job to start at a time specified using the Time Zone, Date, and Time fields.
5. Click **OK**.

The SQL Performance Analyzer Task Result page appears. A link to the SQL tuning report appears under Recommendations.

6. To view the SQL tuning report, click the SQL Tune Report link.

The SQL Tuning Results page appears.

See Also:

- *Oracle Database 2 Day + Performance Tuning Guide* for information about running the SQL Tuning Advisor

Comparing SQL Trials Using APIs

Comparing SQL trials using APIs involves the following steps:

- [Analyzing SQL Performance Using APIs](#)
- [Reviewing the SQL Performance Analyzer Report in Command-Line](#)
- [Comparing SQL Tuning Sets Using APIs](#)
- [Tuning Regressed SQL Statements Using APIs](#)
- [Tuning Regressed SQL Statements From a Remote SQL Trial Using APIs](#)
- [Creating SQL Plan Baselines Using APIs](#)
- [Using SQL Performance Analyzer Views](#)

Analyzing SQL Performance Using APIs

After the post-change SQL performance data is built, you can compare the pre-change version of performance data to the post-change version. Run a comparison analysis using the `DBMS_SQLPA.EXECUTE_ANALYSIS_TASK` procedure or function.

To compare the pre-change and post-change SQL performance data:

1. Call the `EXECUTE_ANALYSIS_TASK` procedure or function using the following parameters:
 - Set the `task_name` parameter to the name of the SQL Performance Analyzer task.
 - Set the `execution_type` parameter to `COMPARE PERFORMANCE`. This setting will analyze and compare two versions of SQL performance data.
 - Specify a name to identify the execution using the `execution_name` parameter. If not specified, it will be generated by SQL Performance Analyzer and returned by the function.
 - Specify two versions of SQL performance data using the `execution_params` parameters. The `execution_params` parameters are specified as *(name, value)* pairs for the specified execution. Set the execution parameters that are related to comparing and analyzing SQL performance data as follows:
 - Set the `execution_name1` parameter to the name of the first execution (before the system change was made). This value should correspond to the value of the `execution_name` parameter specified in ["Creating a Pre-Change SQL Trial Using APIs"](#) on page 4-4.
 - Set the `execution_name2` parameter to the name of the second execution (after the system change was made). This value should correspond to the value of the `execution_name` parameter specified in ["Creating a Post-Change SQL Trial Using APIs"](#) on page 5-3 when you executed the SQL workload after the system change. If the caller does not specify the executions, then by default SQL Performance Analyzer will always compare the last two task executions.

- Set the `comparison_metric` parameter to specify an expression of execution statistics to use in the performance impact analysis. Possible values include the following metrics or any combination of them: `elapsed_time` (default), `cpu_time`, `buffer_gets`, `disk_reads`, `direct_writes`, `optimizer_cost`, and `io_interconnect_bytes`.

For other possible parameters that you can set for comparison, see the description of the `DBMS_SQLPA` package in *Oracle Database PL/SQL Packages and Types Reference*.

The following example illustrates a function call:

```
EXEC DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(task_name => 'my_spa_task', -
    execution_type => 'COMPARE PERFORMANCE', -
    execution_name => 'my_exec_compare', -
    execution_params => dbms_advisor.arglist(-
        'comparison_metric', 'buffer_gets'));
```

2. Call the `REPORT_ANALYSIS_TASK` function using the following parameters:

- Set the `task_name` parameter to the name of the SQL Performance Analyzer task.
- Set the `execution_name` parameter to the name of the execution to use. This value should match the `execution_name` parameter of the execution for which you want to generate a report.

To generate a report to display the results of:

- Execution plans generated for the SQL workload, set this value to match the `execution_name` parameter of the desired `EXPLAIN PLAN` execution.
- Execution plans and execution statistics generated for the SQL workload, set this parameter to match the value of the `execution_name` parameter used in the desired `TEST EXECUTE` execution.
- A comparison analysis, set this value to match the `execution_name` parameter of the desired `ANALYZE PERFORMANCE` execution.

If unspecified, SQL Performance Analyzer generates a report for the last execution.

- Set the `type` parameter to specify the type of report to generate. Possible values include `TEXT` (default), `HTML`, `XML`, and `ACTIVE`.

Active reports provides in-depth reporting using an interactive user interface that enables you to perform detailed analysis even when disconnected from the database or Oracle Enterprise Manager. It is recommended that you use active reports instead of HTML or text reports when possible.

For information about active reports, see ["About SQL Performance Analyzer Active Reports"](#) on page 6-7.

- Set the `level` parameter to specify the format of the recommendations. Possible values include `TYPICAL` (default), `ALL`, `BASIC`, `CHANGED`, `CHANGED_PLANS`, `ERRORS`, `IMPROVED`, `REGRESSED`, `TIMEOUT`, `UNCHANGED`, `UNCHANGED_PLANS`, and `UNSUPPORTED`.
- Set the `section` parameter to specify a particular section to generate in the report. Possible values include `SUMMARY` (default) and `ALL`.
- Set the `top_sql` parameter to specify the number of SQL statements in a SQL tuning set to generate in the report. By default, the report shows the top 100 SQL statements impacted by the system change.

To generate an active report, run the following script:

```
set trimspool on
set trim on
set pages 0
set linesize 1000
set long 1000000
set longchunksize 1000000
spool spa_active.html
SELECT DBMS_SQLPA.REPORT_ANALYSIS_TASK(task_name => 'my_spa_task',
                                     type => 'active', section => 'all') FROM dual;
spool off
```

The following example illustrates a portion of a SQL script that you could use to create and display a comparison summary report in text format:

```
VAR rep CLOB;
EXEC :rep := DBMS_SQLPA.REPORT_ANALYSIS_TASK('my_spa_task', -
                                             'text', 'typical', 'summary');
SET LONG 100000 LONGCHUNKSIZE 100000 LINESIZE 130
PRINT :rep
```

- 3. Review the SQL Performance Analyzer report, as described in ["Reviewing the SQL Performance Analyzer Report in Command-Line"](#) on page 6-12.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_SQLPA.EXECUTE_ANALYSIS_TASK` and `DBMS_SQLPA.REPORT_ANALYSIS_TASK` functions

Reviewing the SQL Performance Analyzer Report in Command-Line

The SQL Performance Analyzer report is divided into the following sections:

- [General Information](#)
- [Result Summary](#)
- [Result Details](#)

This section uses a sample report to illustrate how to review the SQL Performance Analyzer report. The sample report uses `buffer_gets` as the comparison metric to compare the pre-change and post-change executions of a SQL workload.

General Information

The General Information section contains basic information and metadata about the SQL Performance Analyzer task, the SQL tuning set used, and the pre-change and post-change executions. [Example 6-1](#) shows the General Information section of a sample report.

Example 6-1 General Information

General Information	

Task Information:	Workload Information:

Task Name : my_spa_task	SQL Tuning Set Name : my_sts
Task Owner : APPS	SQL Tuning Set Owner : APPS
Description :	Total SQL Statement Count : 101

Execution Information:

```

-----
Execution Name   : my_exec_compare      Started       : 05/21/2007 11:30:09
Execution Type   : ANALYZE PERFORMANCE Last Updated  : 05/21/2007 11:30:10
Description      :                      Global Time Limit : UNLIMITED
Scope           : COMPREHENSIVE        Per-SQL Time Limit : UNUSED
Status          : COMPLETED           Number of Errors  : 0

```

Analysis Information:

```

-----
Comparison Metric: BUFFER_GETS
-----

```

```

Workload Impact Threshold: 1%
-----

```

```

SQL Impact Threshold: 1%
-----

```

Before Change Execution:

```

-----
Execution Name   : my_exec_BEFORE_change
Execution Type    : TEST EXECUTE
Description      :
Scope           : COMPREHENSIVE
Status          : COMPLETED
Started         : 05/21/2007 11:22:06
Last Updated    : 05/21/2007 11:24:01
Global Time Limit : 1800
Per-SQL Time Limit : UNUSED
Number of Errors : 0

```

After Change Execution:

```

-----
Execution Name   : my_exec_AFTER_change
Execution Type    : TEST EXECUTE
Description      :
Scope           : COMPREHENSIVE
Status          : COMPLETED
Started         : 05/21/2007 11:25:56
Last Updated    : 05/21/2007 11:28:30
Global Time Limit : 1800
Per-SQL Time Limit : UNUSED
Number of Errors : 0

```

In [Example 6–1](#), the Task Information section indicates that the task name is `my_spa_task`. The Workload Information section indicates that the task compares executions of the `my_sts` SQL tuning set, which contains 101 SQL statements. As shown in the Execution Information section, the comparison execution is named `my_exec_compare`.

The Analysis Information sections shows that SQL Performance Analyzer compares two executions of the `my_sts` SQL tuning set, `my_exec_BEFORE_change` and `my_exec_AFTER_change`, using `buffer_gets` as a comparison metric.

Result Summary

The Result Summary section summarizes the results of the SQL Performance Analyzer task. The Result Summary section is divided into the following subsections:

- [Overall Performance Statistics](#)
- [Performance Statistics of SQL Statements](#)
- [Errors](#)

Overall Performance Statistics The Overall Performance Statistics subsection displays statistics about the overall performance of the entire SQL workload. This section is a very important part of the SQL Performance Analyzer analysis because it shows the impact of the system change on the overall performance of the SQL workload. Use the information in this section to understand the change of the workload performance, and determine whether the workload performance will improve or degrade after making the system change.

[Example 6–2](#) shows the Overall Performance Statistics subsection of a sample report.

Example 6–2 Overall Performance Statistics

Report Summary

Projected Workload Change Impact:

```

Overall Impact      :  47.94%
Improvement Impact  :  58.02%
Regression Impact   : -10.08%

```

SQL Statement Count

```

-----
SQL Category  SQL Count  Plan Change Count
Overall       101        6
Improved      2          2
Regressed     1          1
Unchanged     98         3

```

This example indicates that the overall performance of the SQL workload improved by 47.94%, even though regressions had a negative impact of -10.08%. This means that if all of the regressions are fixed in this example, the overall change impact will be 58.02%. After the system change, 2 of the 101 SQL statements ran faster, while 1 ran slower. Performance of 98 statements remained unchanged.

Performance Statistics of SQL Statements The Performance Statistics subsection highlights the SQL statements that are the most impacted by the system change. The pre-change and post-change performance data for each SQL statement in the workload are compared based on the following criteria:

- Execution frequency, or importance, of each SQL statement
- Impact of the system change on each SQL statement relative to the entire SQL workload
- Impact of the system change on each SQL statement
- Whether the structure of the execution plan for each SQL statement has changed

[Example 6–3](#) shows the Performance Statistics of SQL Statements subsection of a sample report. The report has been altered slightly to fit on the page.

Example 6–3 Performance Statistics of SQL Statements

SQL Statements Sorted by their Absolute Value of Change Impact on the Workload

```

-----
| object_id | sql_id          | Impact on | Execution | Metric | Metric | Impact | Plan |
| object_id | sql_id          | Workload  | Frequency | Before | After  | on SQL | Change |
-----
| 205       | 73s2sgy2svfrw   | 29.01%    | 100000    | 1681683 | 220590 | 86.88% | y     |
| 206       | gq2a407mv2hsy   | 29.01%    | 949141    | 1681683 | 220590 | 86.88% | y     |
| 204       | 2wtgxbjz6u2by   | -10.08%   | 478254    | 1653012 | 2160529 | -30.7% | y     |
-----

```

The SQL statements are sorted in descending order by the absolute value of the net impact on the SQL workload, that is, the sort order does not depend on whether the impact was positive or negative.

Errors The Errors subsection reports all errors that occurred during an execution. An error may be reported at the SQL tuning set level if it is common to all executions in the SQL tuning set, or at the execution level if it is specific to a SQL statement or execution plan.

[Example 6-4](#) shows an example of the Errors subsection of a SQL Performance Analyzer report.

Example 6-4 Errors

SQL STATEMENTS WITH ERRORS	
SQL ID	Error
47bjmcdtw6htn	ORA-00942: table or view does not exist
br61bjp4tnf7y	ORA-00920: invalid relational operator

Result Details

The Result Details section represents a drill-down into the performance of SQL statements that appears in the Result Summary section of the report. Use the information in this section to investigate why the performance of a particular SQL statement regressed.

This section will contain an entry of every SQL statement processed in the SQL performance impact analysis. Each entry is organized into the following subsections:

- [SQL Details](#)
- [Execution Statistics](#)
- [Execution Plans](#)

SQL Details This section of the report summarizes the SQL statement, listing its information and execution details.

[Example 6-5](#) shows the SQL Details subsection of a sample report.

Example 6-5 SQL Details

SQL Details:

```
-----
Object ID      : 204
Schema Name    : APPS
SQL ID         : 2wtgxbjz6u2by
Execution Frequency : 1
SQL Text       : SELECT /* my_query_14_scott */ /*+ ORDERED INDEX(t1)
                USE_HASH(t1) */ 'B' || t2.pg_featurevalue_05_id
                pg_featurevalue_05_id, 'r' || t4.elementrange_id
                pg_featurevalue_15_id, 'G' || t5.elementgroup_id
                pg_featurevalue_01_id, 'r' || t6.elementrange_id . . .
.
.
.
-----
```

In [Example 6-5](#), the report summarizes the regressed SQL statement whose ID is 2wtgxbjz6u2by and corresponding object ID is 204.

Execution Statistics The Execution Statistics subsection compares execution statistics of the SQL statement from the pre-change and post-change executions and then summarizes the findings.

[Example 6-6](#) shows the Execution Statistics subsection of a sample report.

Example 6-6 Execution Statistics

Execution Statistics:

Stat Name	Impact on Workload	Value Before	Value After	Impact on SQL	% Workload Before	% Workload After
elapsed_time	-95.54%	36.484	143.161	-292.39%	32.68%	94.73%
parse_time	-12.37%	.004	.062	-1450%	.85%	11.79%
exec_elapsed	-95.89%	36.48	143.099	-292.27%	32.81%	95.02%
exec_cpu	-19.73%	36.467	58.345	-59.99%	32.89%	88.58%
buffer_gets	-10.08%	1653012	2160529	-30.7%	32.82%	82.48%
cost	12.17%	11224	2771	75.31%	16.16%	4.66%
reads	-1825.72%	4091	455280	-11028.82%	16.55%	96.66%
writes	-1500%	0	15	-1500%	0%	100%
rows		135	135			

Notes:

Before Change:

1. The statement was first executed to warm the buffer cache.
2. Statistics shown were averaged over next 9 executions.

After Change:

1. The statement was first executed to warm the buffer cache.
2. Statistics shown were averaged over next 9 executions.

Findings (2):

1. The performance of this SQL has regressed.
2. The structure of the SQL execution plan has changed.

Execution Plans The Execution Plans subsection displays the pre-change and post-change execution plans for the SQL statement. In cases when the performance regressed, this section also contains findings on root causes and symptoms.

[Example 6-7](#) shows the Execution Plans subsection of a sample report.

Example 6-7 Execution Plans

Execution Plan Before Change:

Plan Id : 1
Plan Hash Value : 3412943215

Id	Operation	Name	Rows	Bytes	Cost	Time
0	SELECT STATEMENT		1	126	11224	00:02:15
1	HASH GROUP BY		1	126	11224	00:02:15
2	NESTED LOOPS		1	126	11223	00:02:15
* 3	HASH JOIN		1	111	11175	00:02:15
* 4	TABLE ACCESS FULL	LU_ELEMENTGROUP_REL	1	11	162	00:00:02
* 5	HASH JOIN		487	48700	11012	00:02:13
6	MERGE JOIN		14	924	1068	00:00:13

7	SORT JOIN		5391	274941	1033	00:00:13
* 8	HASH JOIN		5391	274941	904	00:00:11
* 9	TABLE ACCESS FULL	LU_ELEMENTGROUP_REL	123	1353	175	00:00:03
* 10	HASH JOIN		5352	214080	729	00:00:09
* 11	TABLE ACCESS FULL	LU_ITEM_293	5355	128520	56	00:00:01
* 12	TABLE ACCESS FULL	ADM_PG_FEATUREVALUE	1629	26064	649	00:00:08
* 13	FILTER					
* 14	SORT JOIN		1	15	36	00:00:01
* 15	TABLE ACCESS FULL	LU_ELEMENTRANGE_REL	1	15	35	00:00:01
16	INLIST ITERATOR					
* 17	TABLE ACCESS BY INDEX ROWID	FACT_PD_OUT_ITM_293	191837	6522458	9927	00:02:00
18	BITMAP CONVERSION TO ROWIDS					
* 19	BITMAP INDEX SINGLE VALUE	FACT_274_PER_IDX				
* 20	TABLE ACCESS FULL	LU_ELEMENTRANGE_REL	1	15	49	00:00:01

.

.

.

Execution Plan After Change:

Plan Id : 102
Plan Hash Value : 1923145679

Id	Operation	Name	Rows	Bytes	Cost	Time
0	SELECT STATEMENT		1	126	2771	00:00:34
1	HASH GROUP BY		1	126	2771	00:00:34
2	NESTED LOOPS		1	126	2770	00:00:34
* 3	HASH JOIN		1	111	2722	00:00:33
* 4	HASH JOIN		1	100	2547	00:00:31
* 5	TABLE ACCESS FULL	LU_ELEMENTGROUP_REL	1	11	162	00:00:02
6	NESTED LOOPS					
7	NESTED LOOPS		484	43076	2384	00:00:29
* 8	HASH JOIN		14	770	741	00:00:09
9	NESTED LOOPS		4	124	683	00:00:09
* 10	TABLE ACCESS FULL	LU_ELEMENTRANGE_REL	1	15	35	00:00:01
* 11	TABLE ACCESS FULL	ADM_PG_FEATUREVALUE	4	64	649	00:00:08
* 12	TABLE ACCESS FULL	LU_ITEM_293	5355	128520	56	00:00:01
13	BITMAP CONVERSION TO ROWIDS					
* 14	BITMAP INDEX SINGLE VALUE	FACT_274_ITEM_IDX				
* 15	TABLE ACCESS BY INDEX ROWID	FACT_PD_OUT_ITM_293	36	1224	2384	00:00:29
* 16	TABLE ACCESS FULL	LU_ELEMENTGROUP_REL	123	1353	175	00:00:03
* 17	TABLE ACCESS FULL	LU_ELEMENTRANGE_REL	1	15	49	00:00:01

Comparing SQL Tuning Sets Using APIs

You can compare two SQL tuning sets using the DBMS_SQLPA package. For example, while using Database Replay, you may have captured a SQL tuning set on the production system during workload capture, and another SQL tuning set on a test system during workload replay. You can then use SQL Performance Analyzer to compare these SQL tuning sets, without having to re-execute the SQL statements. This is useful in cases where you already have another utility to run your workload before and after making the system change, such as a custom script.

When comparing SQL tuning sets, SQL Performance Analyzer uses the runtime statistics captured in the SQL tuning sets to perform its comparison analysis, and reports on any new or missing SQL statements that are found in one SQL tuning set, but not in the other. Any changes in execution plans between the two SQL tuning sets are also reported. For each SQL statement in both SQL tuning sets, improvement and

regression findings are reported for each SQL statement—calculated based on the average statistic value per execution—and for the entire workload—calculated based on the cumulative statistic value.

To compare SQL tuning sets using APIs:

1. Create a SQL Performance Analyzer task:

```
VAR aname varchar2(30);
EXEC :aname := 'compare_s2s';
EXEC :aname := DBMS_SQLPA.CREATE_ANALYSIS_TASK(task_name => :aname);
```

It is not necessary to associate a SQL tuning set to the task during creation.

2. Create the first SQL trial and convert the first SQL tuning set:

```
EXEC DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(task_name => :aname, -
    execution_type => 'convert sqlset', -
    execution_name => 'first trial', -
    execution_params => DBMS_ADVISOR.ARGLIST(
        'sqlset_name', 'my_first_sts', -
        'sqlset_owner', 'APPS'));
```

Specify the name and owner of the SQL tuning set using the `SQLSET_NAME` and `SQLSET_OWNER` task parameters. The content of the SQL tuning set will not be duplicated by the SQL Performance Analyzer task. Instead, a reference to the SQL tuning set is recorded in association to the new SQL trial, which in this example is "first trial".

3. Create a second SQL trial and associate it to the second SQL tuning set to which you want to compare:

```
EXEC DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(task_name => :aname, -
    execution_type => 'convert sqlset', -
    execution_name => 'second trial', -
    execution_params => DBMS_ADVISOR.ARGLIST(
        'sqlset_name', 'my_second_sts', -
        'sqlset_owner', 'APPS'));
```

4. Compare the performance data from the two SQL trials (or SQL tuning sets) by running a comparison analysis:

```
EXEC DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(task_name => :aname, -
    execution_type => 'compare', -
    execution_name => 'comparison', -
    execution_params => DBMS_ADVISOR.ARGLIST(
        'workload_impact_threshold', 0, -
        'sql_impact_threshold', 0));
```

In this example, the workload and per-SQL impact threshold are set to 0% for comparison (the default value is 1%).

5. After the comparison analysis is complete, generate a SQL Performance Analyzer report using the `DBMS_SQLPA.REPORT_ANALYSIS_TASK` function.

For information about generating a SQL Performance Analyzer report using APIs, see ["Analyzing SQL Performance Using APIs"](#) on page 6-10.

Once the report is generated, review it to identify any differences between the contents of the two SQL tuning sets. [Example 6–8](#) shows the Analysis Information and Report Summary sections of a sample report generated by comparing two SQL tuning sets:

Example 6–8 Analysis Information and Report Summary

Analysis Information:

Before Change Execution:		After Change Execution:	
Execution Name	: first trial	Execution Name	: second trial
Execution Type	: CONVERT SQLSET	Execution Type	: CONVERT SQLSET
Status	: COMPLETED	Status	: COMPLETED
Started	: ...		
Last Updated	: ...		
Before Change Workload:		After Change Workload:	
SQL Tuning Set Name	: my_first_sts	SQL Tuning Set Name	: my_second_sts
SQL Tuning Set Owner	: APPS	SQL Tuning Set Owner	: APPS
Total SQL Statement Count	: 5	Total SQL Statement Count	: 6

Report Summary

Projected Workload Change Impact:

Overall Impact	: 72.32%
Improvement Impact	: 47.72%
Regression Impact	: -.02%
Missing-SQL Impact	: 33.1%
New-SQL Impact	: -8.48%

SQL Statement Count

SQL Category	SQL Count	Plan Change Count
Overall	7	1
Common	4	1
Improved	3	1
Regressed	1	0
Different	3	0
Missing SQL	1	0
New SQL	2	0

As shown in [Example 6–8](#), this report contains two additional categories that are not found in standard SQL Performance Analyzer reports; both categories are grouped under the heading **Different**:

- **Missing SQL**

This category represents all SQL statements that are present in the first SQL tuning set, but are not found in the second SQL tuning set. In this example, only one SQL statement is missing. As shown in [Example 6–9](#), this SQL statement has:

- A `sql_id` value of `gv7xb8tyd1v91`
- A performance impact on the workload of 33.1% based on the change
- No performance impact on the SQL statement based on the change because its "Total Metric After" change value is missing

- **New SQL**

This category represents all SQL statements that are present in the second SQL tuning set, but are not found in the first SQL tuning set. In this example, only two

SQL statements are new in the second SQL tuning set. As shown in [Example 6–9](#), these SQL statements have:

- sql_id values of 4c8nrqxhtb2sf and 9utadgu5udmh4
- A total performance impact on the workload of -8.48%
- Missing "Total Metric Before" change values

[Example 6–9](#) shows a table in the sample report that lists the missing and new SQL statements, as well as other top SQL statements as determined by their impact on the workload:

Example 6–9 Top 7 SQL Sorted by Absolute Value of Change Impact on the Workload

Top 7 SQL Sorted by Absolute Value of Change Impact on the Workload

object_id	sql_id	Impact on Workload	Total Metric Before	Total Metric After	Impact on SQL	Plan Change
4	7gj3w9ya4d9sj	41.04%	812791	36974	95%	y
7	gv7xb8tydlv91	33.1%	625582			n
2	4c8nrqxhtb2sf	-8.35%		157782		n
1	22u3tvrt0yr6g	4.58%	302190	215681	28.63%	n
6	fgdd0fd56qmt0	2.1%	146128	106369	27.21%	n
5	9utadgu5udmh4	-.13%		2452		n
3	4dtv43awxnmv3	-.02%	3520	3890	-47.35%	n

Once you have identified a SQL statement of interest, you can generate a report for the SQL statement to perform more detailed investigation. For example, you may want to investigate the SQL statement with the sql_id value of 7gj3w9ya4d9sj and object_id value of 4 because it has the highest impact on the workload:

```
SELECT DBMS_SQLPA.REPORT_ANALYSIS_TASK(task_name => :aname, object_id => 4) rep
FROM dual;
```

[Example 6–10](#) shows a sample report generated for this SQL statement:

Example 6–10 Sample Report for SQL Statement

SQL Details:

```
Object ID   : 4
SQL ID      : 7gj3w9ya4d9sj
SQL Text    : /* my_csts_query1 */ select * FROM emp where empno=2
```

SQL Execution Statistics (average):

Stat Name	Impact on Workload	Value Before	Value After	Impact on SQL
elapsed_time	41.04%	.036945	.001849	95%
cpu_time	13.74%	.004772	.00185	61.24%
buffer_gets	9.59%	8	2	69.01%
cost	11.76%	1	1	10%
reads	4.08%	0	0	63.33%
writes	0%	0	0	0%
rows		0	0	
executions		22	20	

plan_count		3	2	
------------	--	---	---	--

Findings (2):

1. The performance of this SQL has improved.
2. The structure of the SQL execution plan has changed.

Plan Execution Statistics (average):

Statistic Name	Plans Before Change			Plans After Change	
plan hash value	440231712	571903972	3634526668	571903972	3634526668
schema name	APPS1	APPS2	APPS2	APPS2	APPS2
executions	7	5	10	10	10
cost	2	1	2	1	2
elapsed_time	.108429	.000937	.00491	.000503	.003195
cpu_time	.00957	.0012	.0032	.0005	.0032
buffer_gets	18	0	5	0	5
reads	0	0	0	0	0
writes	0	0	0	0	0
rows	0	0	0	0	0

Execution Plans Before Change:

Plan Hash Value : 440231712

Id	Operation	Name	Rows	Bytes	Cost	Time
0	SELECT STATEMENT				2	
1	PX COORDINATOR					
2	PX SEND QC (RANDOM)	:TQ10000	1	87	2	00:00:01
3	PX BLOCK ITERATOR		1	87	2	00:00:01
4	TABLE ACCESS FULL	EMP	1	87	2	00:00:01

Note

- dynamic sampling used for this statement

Plan Hash Value : 571903972

Id	Operation	Name	Rows	Bytes	Cost	Time
0	SELECT STATEMENT				1	
1	TABLE ACCESS BY INDEX ROWID	EMP	1	87	1	00:00:01
2	INDEX UNIQUE SCAN	MY_EMP_IDX	1		0	

Plan Hash Value : 3634526668

Id	Operation	Name	Rows	Bytes	Cost	Time
0	SELECT STATEMENT				2	
1	TABLE ACCESS FULL	EMP	1	87	2	00:00:01

Note

- dynamic sampling used for this statement

Executions Plan After Change:

Plan Hash Value : 571903972

Id	Operation	Name	Rows	Bytes	Cost	Time
0	SELECT STATEMENT				1	
1	TABLE ACCESS BY INDEX ROWID	EMP	1	87	1	00:00:01
2	INDEX UNIQUE SCAN	MY_EMP_IDX	1		0	

Plan Hash Value : 3634526668

Id	Operation	Name	Rows	Bytes	Cost	Time
0	SELECT STATEMENT				2	
1	TABLE ACCESS FULL	EMP	1	87	2	00:00:01

Note

- dynamic sampling used for this statement

The SQL Execution Statistics section shows the average runtime statistics (per execution) of the SQL statement. The data in this table reveals that this SQL statement is present in both SQL tuning sets, but that it has only three execution plans in the first SQL tuning set and two execution plans in the second SQL tuning set. Furthermore, the SQL statement was executed 22 times in the first SQL tuning set, but only 20 times in the second SQL tuning set.

The Plan Execution Statistics section shows runtime statistics per execution plan (or plan hash value). The Plans Before Change column lists plans and their associated execution statistics for the first SQL tuning set; the Plans After Change columns lists these values for the second SQL tuning set. Execution plans structures for both SQL tuning sets are shown at the end of the report.

You can use these sections in the report to identify changes in execution plans between two SQL tuning sets. This is important because changes in execution plans may be a result of test changes that can have a direct impact to performance. When comparing two SQL tuning sets, SQL Performance Analyzer reports execution plan changes when a SQL statement has:

- One plan in both SQL tuning sets, but the plan structure is different
- More than one plan, and the number of plans in both SQL tuning sets are:
 - The same, but at least one plan in the second SQL tuning set is different from all plans in the first SQL tuning set
 - Different

After evaluating the SQL statement and plan changes, determine if further action is required. If the SQL statement has regressed, perform one of the following actions:

- Tune the regressed SQL statement, as described in ["Tuning Regressed SQL Statements Using APIs"](#) on page 6-22
- Create SQL plan baselines, as described in ["Creating SQL Plan Baselines Using APIs"](#) on page 6-26

Tuning Regressed SQL Statements Using APIs

After reviewing the SQL Performance Analyzer report, you should tune any regressed SQL statements that are identified after comparing the SQL performance. If there are

large numbers of SQL statements that appear to have regressed, you should try to identify the root cause and make system-level changes to rectify the problem. In cases when only a few SQL statements have regressed, consider using the SQL Tuning Advisor to implement a point solution for them, or creating SQL plan baselines to instruct the optimizer to select the original execution plan in the future.

To tune regressed SQL statements using APIs:

- Create a SQL tuning task for the SQL Performance Analyzer execution by using the `CREATE_TUNING_TASK` function in the `DBMS_SQLTUNE` package:

```
BEGIN
  DBMS_SQLTUNE.CREATE_TUNING_TASK (
    spa_task_name => 'my_spa_task',
    spa_task_owner => 'immchan',
    spa_compare_exec => 'my_exec_compare');
  DBMS_SQLTUNE.EXECUTE_TUNING_TASK(spa_task_name => 'my_spa_task');
END;
/
```

This example creates and executes a SQL tuning task to tune the SQL statements that regressed in the compare performance execution named `my_exec_compare` of the SQL Performance Analyzer task named `my_spa_task`. In this case, it is important to use this version of the `CREATE_TUNING_TASK` function call. Otherwise, SQL statements may be tuned in the environment from the production system where they were captured, which will not reflect the system change.

Note: If you chose to execute the SQL workload remotely on a separate database, you should not use this version of the `CREATE_TUNING_TASK` function call to tune regressed SQL statements. Instead, you should tune any regressions identified by the SQL trials on the remote database, because the application schema is not on the database running SQL Performance Analyzer. Therefore, you need to run SQL Tuning Advisor on the database where the schema resides and where the change was made. For more information, see ["Tuning Regressed SQL Statements From a Remote SQL Trial Using APIs"](#) on page 6-24.

[Table 6–1](#) lists the SQL Performance Analyzer parameters that can be used with the `DBMS_SQLTUNE.CREATE_TUNING_TASK` function.

Table 6–1 CREATE_TUNING_TASK Function SQL Performance Analyzer Parameters

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

After tuning the regressed SQL statements, you should test these changes using SQL Performance Analyzer. Run a new SQL trial on the test system, followed by a second comparison (between this new SQL trial and the first SQL trial) to validate your

results. Once SQL Performance Analyzer shows that performance has stabilized, implement the fixes from this step to your production system.

Starting with Oracle Database 11g Release 2, SQL Tuning Advisor performs an alternative plan analysis when tuning a SQL statement. SQL Tuning Advisor reviews the execution history of the SQL statement, including any historical plans stored in the Automatic Workload Repository. If SQL Tuning Advisor finds alternate plans, it allows you to choose a specific plan and create a plan baseline to ensure that the desired execution plan is used for that SQL statement.

See Also:

- *Oracle Database SQL Tuning Guide* for information about using the SQL Tuning Advisor
- *Oracle Database SQL Tuning Guide* for information about alternative plan analysis
- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_SQLTUNE` package

Tuning Regressed SQL Statements From a Remote SQL Trial Using APIs

If you chose to execute the SQL workload remotely on a separate database, then you should tune any regressions identified by the SQL trials on the remote database, instead of the system where the SQL Performance Analyzer task resides.

To tune regressed SQL statements from a remote SQL trial using APIs:

1. On the system running SQL Performance Analyzer, create a subset of the regressed SQL statements as a SQL tuning set:

```
DECLARE
    sqlset_cur  DBMS_SQLTUNE.SQLSET_CURSOR;
BEGIN
    DBMS_SQLTUNE.CREATE_SQLSET('SUB_STS1', 'test purpose');

    OPEN sqlset_cur FOR
        SELECT value(p)
        FROM table(
            DBMS_SQLTUNE.SELECT_SQLPA_TASK(
                task_name => 'SPA_TASK1',
                execution_name => 'COMP',
                level_filter => 'REGRESSED')) p;

    DBMS_SQLTUNE.LOAD_SQLSET('SUB_STS1', sqlset_cur);

    CLOSE sqlset_cur;
END;
/
```

Other than 'REGRESSED', you can use other filters to select SQL statements for the SQL tuning set, such as 'CHANGED', 'ERRORS', or 'CHANGED_PLANS'. For more information, see *Oracle Database PL/SQL Packages and Types Reference*.

2. Create a staging table to where the SQL tuning set will be exported:

```
BEGIN
    DBMS_SQLTUNE.CREATE_STGTAB_SQLSET(
        table_name => 'STG_TAB1',
        schema_name => 'JOHNDOE',
        tablespace_name => 'TBS_1',
```



```

        db_version => DBMS_SQLTUNE.STS_STGTAB_11_1_VERSION);
END;
/

```

Use the `db_version` parameter to specify the appropriate database version to where the SQL tuning set will be exported and tuned. In this example, the staging table will be created with a format so that it can be exported to a system running Oracle Database 11g Release 1, where it will later be tuned using SQL Tuning Advisor. For other database versions, see *Oracle Database PL/SQL Packages and Types Reference* for that release.

3. Export the SQL tuning set into the staging table:

```

BEGIN
    DBMS_SQLTUNE.PACK_STGTAB_SQLSET(
        sqlset_name => 'SUB_STS1',
        sqlset_owner => 'JOHNDOE',
        staging_table_name => 'STG_TAB1',
        staging_schema_owner => 'JOHNDOE',
        db_version => DBMS_SQLTUNE.STS_STGTAB_11_1_VERSION);
END;
/

```

4. Move the staging table to the remote database (where the SQL workload was executed) using the mechanism of choice (such as Oracle Data Pump or database link).

5. On the remote database, import the SQL tuning set from the staging table:

```

BEGIN
    DBMS_SQLTUNE.UNPACK_STGTAB_SQLSET(
        sqlset_name => 'SUB_STS1',
        staging_table_name => 'STG_TAB1',
        replace => TRUE);
END;
/

```

6. Tune the regressed SQL statements in the SQL tuning set by running SQL Tuning Advisor:

```

BEGIN
    sts_name := 'SUB_STS1';
    sts_owner := 'JOHNDOE';
    tune_task_name := 'TUNE_TASK1';
    tname := DBMS_SQLTUNE.CREATE_TUNING_TASK(sqlset_name => sts_name,
                                              sqlset_owner => sts_owner,
                                              task_name => tune_task_name);
    EXEC DBMS_SQLTUNE.SET_TUNING_TASK_PARAMETER(:tname,
                                              'APPLY_CAPTURED_COMPILEENV',
                                              'FALSE');
    exec_name := DBMS_SQLTUNE.EXECUTE_TUNING_TASK(tname);
END;
/

```

Note: The `APPLY_CAPTURED_COMPILEENV` parameter used in this example is only supported by Oracle Database 11g Release 1 and newer releases. If you are testing a database upgrade from an earlier version of Oracle Database, SQL Tuning Advisor will use the environment variables stored in the SQL tuning set instead.

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

See Also:

- *Oracle Database SQL Tuning Guide* for information about using the SQL Tuning Advisor and transporting SQL tuning sets
- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_SQLTUNE` package

Creating SQL Plan Baselines Using APIs

Creating SQL plan baselines for regressed SQL statements with plan changes is another option to running the SQL Tuning Advisor. Doing so instructs the optimizer to use the original execution plans for these SQL statements in the future.

To create SQL plan baselines for the original plans:

1. Create a subset of a SQL tuning set of only the regressed SQL statements.
2. Create SQL plan baselines for this subset of SQL statements by loading their plans using the `LOAD_PLANS_FROM_SQLSET` function of the `DBMS_SPM` package, as shown in the following example:

```
DECLARE
    my_plans PLS_INTEGER;
BEGIN
    my_plans := DBMS_SPM.LOAD_PLANS_FROM_SQLSET(
        sqlset_name => 'regressed_sql');
END;
/
```

See Also:

- *Oracle Database SQL Tuning Guide* for information about using SQL plan baselines
- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_SPM` package

Using SQL Performance Analyzer Views

You can query the following views to monitor SQL Performance Analyzer and view its analysis results:

Note: The information available in these views are also contained in the SQL Performance Analyzer report. It is recommended that you use the SQL Performance Analyzer report to view analysis results instead. Consider using these views only for performing more advanced analysis of the results.

- The `DBA_ADVISOR_TASKS` and `USER_ADVISOR_TASKS` views display descriptive information about the SQL Performance Analyzer task that was created.

- The `DBA_ADVISOR_EXECUTIONS` and `USER_ADVISOR_EXECUTIONS` views display information about task executions. SQL Performance Analyzer creates at least three executions to analyze the SQL performance impact caused by a database change on a SQL workload. The first execution collects a pre-change version of the performance data. The second execution collects a post-change version of the performance data. The third execution performs the comparison analysis.
- The `DBA_ADVISOR_FINDINGS` and `USER_ADVISOR_FINDINGS` views display the SQL Performance Analyzer findings. SQL Performance Analyzer generates the following types of findings:
 - Problems, such as performance regression
 - Symptoms, such as when the structure of an execution plan has changed
 - Errors, such as nonexistence of an object or view
 - Informative messages, such as when the structure of an execution plan in the pre-change version is different than the one stored in the SQL tuning set
- The `DBA_ADVISOR_SQLPLANS` and `USER_ADVISOR_SQLPLANS` views display a list of all execution plans.
- The `DBA_ADVISOR_SQLSTATS` and `USER_ADVISOR_SQLSTATS` views display a list of all SQL compilations and execution statistics.
- The `V$ADVISOR_PROGRESS` view displays the operation progress of SQL Performance Analyzer. Use this view to monitor how many SQL statements have completed or are awaiting execution in a SQL trial. The `SO FAR` column indicates the number of SQL statements processed so far, and the `TOTAL WORK` column shows the total number of SQL statements to be processed by the task execution.

You must have the `SELECT_CATALOG_ROLE` role to access the DBA views.

See Also:

- *Oracle Database Reference* for information about the `DBA_ADVISOR_TASKS`, `DBA_ADVISOR_EXECUTIONS`, and `DBA_ADVISOR_SQLPLANS` views

Testing a Database Upgrade

SQL Performance Analyzer supports testing database upgrades from Oracle9i and later releases to Oracle Database 10g Release 2 or newer releases. The methodology used to test a database upgrade from Oracle9i Database and Oracle Database 10g Release 1 is slightly different from the one used to test a database upgrade from Oracle Database 10g Release 2 and later releases, so both methodologies are described here.

This chapter describes how to use SQL Performance Analyzer in a database upgrade and contains the following sections:

- [Upgrading from Oracle9i Database and Oracle Database 10g Release 1](#)
- [Upgrading from Oracle Database 10g Release 2 and Newer Releases](#)
- [Tuning Regressed SQL Statements After Testing a Database Upgrade](#)

See Also:

- ["SQL Performance Analyzer"](#) on page 1-1 for information about using SQL Performance Analyzer in other cases

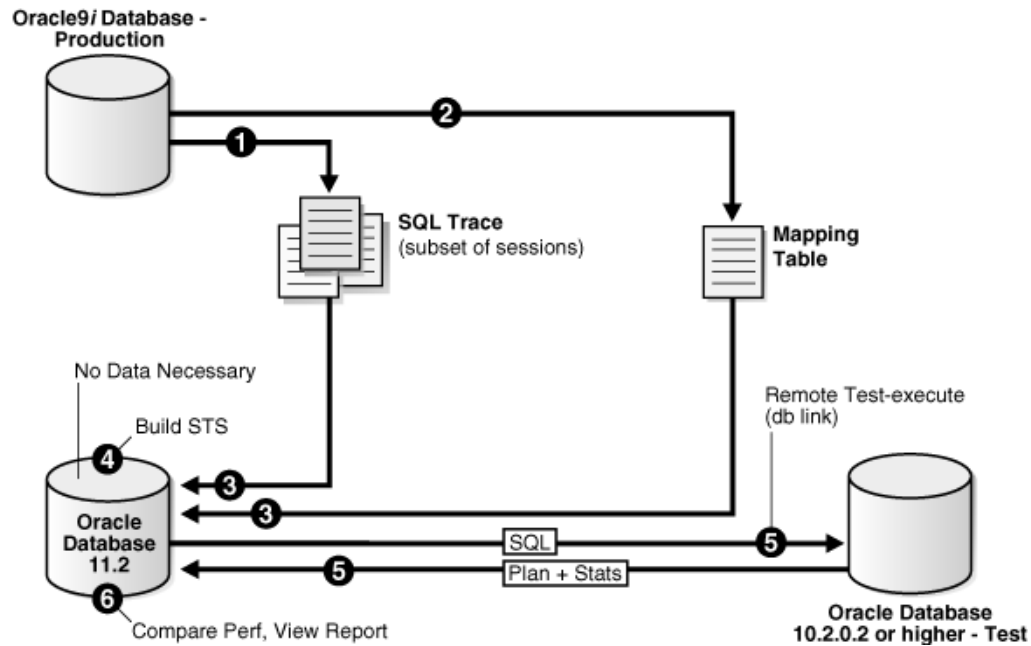
Upgrading from Oracle9i Database and Oracle Database 10g Release 1

As illustrated in [Figure 7-1](#), SQL Performance Analyzer supports testing database upgrades of Oracle9i Database and Oracle Database 10g Release 1 to Oracle Database 10g Release 2 and newer releases by:

- Building a SQL tuning set from SQL trace files captured on the production system
- Executing the SQL tuning set on the upgraded database remotely over a database link
- Comparing the results to those captured on the production system

Because SQL Performance Analyzer only accepts a set of SQL statements stored in a SQL tuning set as its input source, and SQL tuning sets are not supported for Oracle9i Database, a SQL tuning set must be constructed so that it can be used as an input source for SQL Performance Analyzer if you are upgrading from Oracle9i Database.

Figure 7-1 SQL Performance Analyzer Workflow for Database Upgrade from Oracle9i to Oracle Database 10g Release 2



Before the database upgrade can be tested, ensure that the following conditions are met:

- The production system which you are upgrading from is running Oracle9i or Oracle Database 10g Release 1.
- The test system which you are upgrading to is running Oracle Database 10g Release 2 or a newer release.
The database version can be release 10.2.0.2 or higher. If you are upgrading to Oracle Database 10g release 10.2.0.2, 10.2.0.3, or 10.2.0.4, you will also need to install a one-off patch before proceeding.
- The test system must resemble the production system as closely as possible because the performance on both systems will be compared to each other.
- The hardware configurations on both systems must also be as similar as possible.

You will also need to set up a separate SQL Performance Analyzer system running Oracle Database 11g Release 2. You will be using this system to build a SQL tuning set and to run SQL Performance Analyzer. Neither your production data or schema need to be available on this system, since the SQL tuning set will be built using statistics stored in the SQL trace files from the production system. SQL Performance Analyzer tasks will be executed remotely on the test system to generate the execution plan and statistics for the SQL trial over a database link that you specify. The database link must be a public database link that connects to a user with the EXECUTE privilege for the DBMS_SQLPA package and the ADVISOR privilege on the test system. You should also drop any existing PLAN_TABLE from the user's schema on the test system.

Once the upgrade environment is configured as described, perform the steps as described in the following procedure to use SQL Performance Analyzer in a database upgrade from Oracle9i or Oracle Database 10g Release 1 to a newer release.

1. Enable the SQL Trace facility on the production system, as described in ["Enabling SQL Trace on the Production System"](#) on page 7-3.

To minimize the performance impact on the production system and still be able to fully capture a representative set of SQL statements, consider enabling SQL Trace for only a subset of the sessions, for as long as required, to capture all important SQL statements at least once.

2. On the production system, create a mapping table, as described in ["Creating a Mapping Table"](#) on page 7-4.

This mapping table will be used to convert the user and object identifier numbers in the SQL trace files to their string equivalents.

3. Move the SQL trace files and the mapping table from the production system to the SQL Performance Analyzer system, as described in ["Creating a Mapping Table"](#) on page 7-4.
4. On the SQL Performance Analyzer system, construct a SQL tuning set using the SQL trace files, as described in ["Building a SQL Tuning Set"](#) on page 7-5.

The SQL tuning set will contain the SQL statements captured in the SQL trace files, along with their relevant execution context and statistics.

5. On the SQL Performance Analyzer system, use SQL Performance Analyzer to create a SQL Performance Analyzer task and convert the contents in the SQL tuning set into a pre-upgrade SQL trial that will be used as a baseline for comparison, then remotely test execute the SQL statements on the test system over a database link to build a post-upgrade SQL trial, as described in ["Testing Database Upgrades from Oracle9i Database and Oracle Database 10g Release 1"](#) on page 7-6.
6. Compare SQL performance and fix regressed SQL.

SQL Performance Analyzer compares the performance of SQL statements read from the SQL tuning set during the pre-upgrade SQL trial to those captured from the remote test execution during the post-upgrade SQL trial. A report is produced to identify any changes in execution plans or performance of the SQL statements.

If the report reveals any regressed SQL statements, you can make further changes to fix the regressed SQL, as described in ["Tuning Regressed SQL Statements After Testing a Database Upgrade"](#) on page 7-17.

Repeat the process of executing the SQL tuning set and comparing its performance to a previous execution to test any changes made until you are satisfied with the outcome of the analysis.

Enabling SQL Trace on the Production System

Oracle9i uses the SQL Trace facility to collect performance data on individual SQL statements. The information generated by SQL Trace is stored in SQL trace files. SQL Performance Analyzer consumes the following information from these files:

- SQL text and username under which parse occurred
- Bind values for each execution
- CPU and elapsed times
- Physical reads and logical reads
- Number of rows processed
- Execution plan for each SQL statement (only captured if the cursor for the SQL statement is closed)

Although it is possible to enable SQL Trace for an instance, it is recommended that you enable SQL Trace for a subset of sessions instead. When the SQL Trace facility is enabled for an instance, performance statistics for all SQL statements executed in the instance are stored into SQL trace files. Using SQL Trace in this way can have a severe performance impact and may result in increased system overhead, excessive CPU usage, and inadequate disk space. It is required that trace level be set to 4 to capture bind values, along with the execution plans.

For production systems running Oracle Database 10g Release 1, use the `DBMS_MONITOR.SESSION_TRACE_ENABLE` procedure to enable SQL Trace transparently in another session. You should also enable binds explicitly by setting the `binds` procedure parameter to `TRUE` (its default value is `FALSE`).

After enabling SQL Trace, identify the SQL trace files containing statistics for a representative set of SQL statements that you want to use with SQL Performance Analyzer. You can then copy the SQL trace files to the SQL Performance Analyzer system. Once the SQL workload is captured in the SQL trace files, disable SQL Trace on the production system.

See Also:

- *Oracle Database SQL Tuning Guide* for additional considerations when using SQL Trace, such as setting initialization parameters to manage SQL trace files
- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_MONITOR` package

Creating a Mapping Table

To convert the user and object identifier numbers stored in the SQL trace files to their respective names, you need to provide a table that specifies each mapping. The SQL Performance Analyzer system will read this mapping table when converting the trace files into a SQL tuning set.

To create a mapping table:

- Run the following SQL statements on the production database:

```
CREATE TABLE mapping AS
  SELECT object_id id, owner, SUBSTR(object_name, 1, 30) name FROM dba_
objects
  WHERE object_type NOT IN ('CONSUMER GROUP', 'EVALUATION CONTEXT',
'FUNCTION',
                           'INDEXTYPE', 'JAVA CLASS', 'JAVA DATA',
                           'JAVA RESOURCE', 'LIBRARY', 'LOB', 'OPERATOR',
                           'PACKAGE', 'PACKAGE BODY', 'PROCEDURE', 'QUEUE',
                           'RESOURCE PLAN', 'SYNONYM', 'TRIGGER', 'TYPE',
                           'TYPE BODY')
UNION ALL
  SELECT user_id id, username owner, null name FROM dba_users;
```

Once the mapping table is created, you can use Data Pump to transport it to the SQL Performance Analyzer system.

See Also:

- *Oracle Database Utilities* for information about using Data Pump

Building a SQL Tuning Set

Once the SQL trace files and mapping table are moved to the SQL Performance Analyzer system, you can build a SQL tuning set using the `DBMS_SQLTUNE` package.

To build a SQL tuning set:

1. Copy the SQL trace files to a directory on the SQL Performance Analyzer system.
2. Create a directory object for this directory.
3. Use the `DBMS_SQLTUNE.SELECT_SQL_TRACE` function to read the SQL statements from the SQL trace files.

For each SQL statement, only information for a single execution is collected. The execution frequency of each SQL statement is not captured. Therefore, when performing a comparison analysis for a production system running Oracle Database 10g Release 1 and older releases, you should ignore the workload-level statistics in the SQL Performance Analyzer report and only evaluate performance changes on an execution level.

The following example reads the contents of SQL trace files stored in the `sql_trace_prod` directory object and loads them into a SQL tuning set.

```
DECLARE
    cur sys_refcursor;
BEGIN
    DBMS_SQLTUNE.CREATE_SQLSET('my_sts_9i');
    OPEN cur FOR
        SELECT VALUE (P)
        FROM table(DBMS_SQLTUNE.SELECT_SQL_TRACE('sql_trace_prod', '%ora%')) P;
    DBMS_SQLTUNE.LOAD_SQLSET('my_sts_9i', cur);
    CLOSE cur;
END;
/
```

The syntax for the `SELECT_SQL_TRACE` function is as follows:

```
DBMS_SQLTUNE.SELECT_SQL_TRACE (
    directory           IN VARCHAR2,
    file_name           IN VARCHAR2 := NULL,
    mapping_table_name  IN VARCHAR2 := NULL,
    mapping_table_owner IN VARCHAR2 := NULL,
    select_mode         IN POSITIVE := SINGLE_EXECUTION,
    options             IN BINARY_INTEGER := LIMITED_COMMAND_TYPE,
    pattern_start       IN VARCHAR2 := NULL,
    parttern_end        IN VARCHAR2 := NULL,
    result_limit        IN POSITIVE := NULL)
RETURN sys.sqlset PIPELINED;
```

[Table 7–1](#) describes the available parameters for the `SELECT_SQL_TRACE` function.

Table 7–1 DBMS_SQLTUNE.SELECT_SQL_TRACE Function Parameters

Parameter	Description
<code>directory</code>	Specifies the directory object pointing to the directory where the SQL trace files are stored.
<code>file_name</code>	Specifies all or part of the name of the SQL trace files to process. If unspecified, the current or most recent trace file in the specified directory will be used. % wildcards are supported for matching trace file names.

Table 7–1 (Cont.) DBMS_SQLTUNE.SELECT_SQL_TRACE Function Parameters

Parameter	Description
mapping_table_name	Specifies the name of the mapping table. If set to the default value of NULL, mappings from the current database will be used. Note that the mapping table name is not case-sensitive.
mapping_table_owner	Specifies the schema where the mapping table resides. If set to NULL, the current schema will be used.
select_mode	Specifies the mode for selecting SQL statements from the trace files. The default value is SINGLE_EXECUTION. In this mode, only statistics for a single execution per SQL statement will be loaded into the SQL tuning set. The statistics are not cumulative, as is the case with other SQL tuning set data source table functions.
options	Specifies the options for the operation. The default value is LIMITED_COMMAND_TYPE, only SQL types that are meaningful to SQL Performance Analyzer (such as SELECT, INSERT, UPDATE, and DELETE) are returned from the SQL trace files.
pattern_start	Specifies the opening delimiting pattern of the trace file sections to consider. This parameter is currently not used.
pattern_end	Specifies the closing delimiting pattern of the trace file sections to process. This parameter is currently not used.
result_limit	Specifies the top SQL from the (filtered) source. The default value is 2 ³¹ , which represents unlimited.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about the DBMS_SQLTUNE package

Testing Database Upgrades from Oracle9i Database and Oracle Database 10g Release 1

Once the SQL tuning set is built, you can use SQL Performance Analyzer to build a pre-upgrade SQL trial from the execution plans and run-time statistics in the SQL tuning set. After the pre-upgrade SQL trial is built, perform a test execute or generate plans of SQL statements in the SQL tuning set on the test system to build a post-upgrade SQL trial. SQL Performance Analyzer test executes the SQL statements using a public database link that you specify by connecting to the test system remotely and generating the execution plans and statistics for the SQL trial. The database link should exist on the SQL Performance Analyzer system and connect to a remote user with privileges to execute the SQL tuning set on the test system.

You can run SQL Performance Analyzer to test a database upgrade from Oracle9i Database or Oracle Database 10g Release 1 using Oracle Enterprise Manager or APIs, as described in the following sections:

- [Testing Database Upgrades from Releases 9.x and 10.1 Using Cloud Control](#)
- [Testing Database Upgrades from Releases 9.x and 10.1 Using APIs](#)

Testing Database Upgrades from Releases 9.x and 10.1 Using Cloud Control

To test a database upgrade from Oracle9i Database or Oracle Database 10g Release 1 using SQL Performance Analyzer:

1. From the **Performance** menu, select **SQL**, then **SQL Performance Analyzer**.

If the Database Login page appears, then log in as a user with administrator privileges.

The SQL Performance Analyzer page appears.

2. Under SQL Performance Analyzer Workflows, click **Upgrade from 9i or 10.1**.

The Upgrade from 9i or 10.1 page appears.

Upgrade from 9i or 10.1

Task Information

* Task Name

* SQL Tuning Set

Description

Pre-upgrade Trial

Creation Method Build From SQL Tuning Set

Post-upgrade Trial

Creation Method

Per-SQL Time Limit

TIP Time limit is on elapsed time of test execution of SQL.

* Database Link

TIP Provide a PUBLIC database link connecting to a remote user with privileges to execute the Tuning Set SQL.

Trial Comparison

Comparison Metric

Schedule

Time Zone

☒ Immediately

☐ Later

Date

(example: May 2, 2012)

Time ☐ AM ☒ PM

3. Under Task Information:
 - a. In the Task Name field, enter the name of the task.
 - b. In the SQL Tuning Set field, enter the name of the SQL tuning set that was built.

Alternatively, click the search icon to search for the SQL tuning set using the Search and Select: SQL Tuning Set window.

The selected SQL tuning set now appears in the SQL Tuning Set field.
 - c. In the Description field, optionally enter a description of the task.
4. In the Creation Method field, select:
 - **Execute SQLs** to generate both execution plans and statistics for each SQL statement in the SQL tuning set by actually running the SQL statements remotely on the test system over a public database link.
 - **Generate Plans** to create execution plans remotely on the test system over a public database link without actually running the SQL statements.
5. In the Per-SQL Time Limit list, determine the time limit for SQL execution during the trial by performing one of the following actions:
 - Select **5 minutes**.

The execution will run each SQL statement in the SQL tuning set up to 5 minutes and gather performance data.

- Select **Unlimited**.

The execution will run each SQL statement in the SQL tuning set to completion and gather performance data. Collecting execution statistics provides greater accuracy in the performance analysis but takes a longer time. Using this setting is not recommended because the task may be stalled by one SQL statement for a prolonged time period.

- Select **Customize** and enter the specified number of seconds, minutes, or hours.

6. In the Database Link field, enter the global name of a public database link connecting to a user with the `EXECUTE` privilege for the `DBMS_SQLPA` package and the `ADVISOR` privilege on the test system.

Alternatively, click the search icon to search for and select a database link, or click **Create Database Link** to create a database link using the Create Database Link page.

7. In the Comparison Metric list, select the comparison metric to use for the comparison analysis:

- **Elapsed Time**
- **CPU Time**
- **User I/O Time**
- **Buffer Gets**
- **Physical I/O**
- **Optimizer Cost**
- **I/O Interconnect Bytes**

Optimizer Cost is the only comparison metric available if you generated execution plans only in the SQL trials.

To perform the comparison analysis by using more than one comparison metric, perform separate comparison analyses by repeating this procedure with different metrics.

8. Under Schedule:

- a. In the Time Zone list, select your time zone code.
- b. Select **Immediately** to start the task now, or **Later** to schedule the task to start at a time specified using the Date and Time fields.

9. Click **Submit**.

The SQL Performance Analyzer page appears.

In the SQL Performance Analyzer Tasks section, the status of this task is displayed. To refresh the status icon, click **Refresh**. After the task completes, the Status field changes to Completed.

10. Under SQL Performance Analyzer Tasks, select the task and click the link in the Name column.

The SQL Performance Analyzer Task page appears.

This page contains the following sections:

- **SQL Tuning Set**
This section summarizes information about the SQL tuning set, including its name, owner, description, and the number of SQL statements it contains.
 - **SQL Trials**
This section includes a table that lists the SQL trials used in the SQL Performance Analyzer task.
 - **SQL Trial Comparisons**
This section contains a table that lists the results of the SQL trial comparisons
11. Click the icon in the Comparison Report column.
The SQL Performance Analyzer Task Result page appears.
 12. Review the results of the performance analysis, as described in ["Reviewing the SQL Performance Analyzer Report Using Oracle Enterprise Manager"](#) on page 6-3.
If regressed SQL statements are found following the database upgrade, tune them as described in ["Tuning Regressed SQL Statements After Testing a Database Upgrade"](#) on page 7-17.

Testing Database Upgrades from Releases 9.x and 10.1 Using APIs

This section describes how to test database upgrades from Oracle Database releases 9.x and 10.1 using APIs.

To test a database upgrade from releases 9.x and 10.1:

1. On the system running SQL Performance Analyzer, create an analysis task.
2. Build the pre-upgrade SQL trial from the execution plans and run-time statistics in the SQL tuning set by calling the `EXECUTE_ANALYSIS_TASK` procedure using the following parameters:
 - Set the `task_name` parameter to the name of the SQL Performance Analyzer task that you want to execute.
 - Set the `execution_type` parameter to `CONVERT SQLSET` to direct SQL Performance Analyzer to treat the statistics in the SQL tuning set as a trial execution.
 - Specify a name to identify the execution using the `execution_name` parameter. If not specified, then SQL Performance Analyzer automatically generates a name for the task execution.

The following example executes the SQL Performance Analyzer task named `my_spa_task` as a trial execution:

```
EXEC DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(task_name => 'my_spa_task', -
    execution_type => 'CONVERT SQLSET', -
    execution_name => 'my_trial_9i');
```

3. Build the post-upgrade SQL trial by performing an explain plan or test execute using the `EXECUTE_ANALYSIS_TASK` procedure:
 - Set the `execution_type` parameter to `EXPLAIN PLAN` or `TEST EXECUTE`:
 - If you choose to use `EXPLAIN PLAN`, only execution plans will be generated. Subsequent comparisons will only be able to yield a list of changed plans without making any conclusions about performance changes.

- If you choose to use `TEST EXECUTE`, the SQL workload will be executed to completion. This effectively builds the post-upgrade SQL trial using the statistics and execution plans generated from the test system. Using `TEST EXECUTE` is recommended to capture the SQL execution plans and performance data at the source, thereby resulting in a more accurate analysis.
- Set the `DATABASE_LINK` task parameter to the global name of a public database link connecting to a user with the `EXECUTE` privilege for the `DBMS_SQLPA` package and the `ADVISOR` privilege on the test system.

The following example performs a test execute of the SQL statements remotely over a database link:

```
EXEC DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(task_name => 'my_spa_task', -
    execution_type => 'TEST EXECUTE', -
    execution_name => 'my_remote_trial_10g', -
    execution_params => dbms_advisor.arglist('database_link',
                                            'LINK.A.B.C.BIZ.COM'));
```

See Also:

- ["Creating an Analysis Task Using APIs"](#) on page 3-13
- ["Creating a Pre-Change SQL Trial Using APIs"](#) on page 4-4
- ["Creating a Post-Change SQL Trial Using APIs"](#) on page 5-3
- *Oracle Database PL/SQL Packages and Types Reference* to learn about the `DBMS_SQLPA.EXECUTE_ANALYSIS_TASK` function

Upgrading from Oracle Database 10g Release 2 and Newer Releases

You can use SQL Performance Analyzer to test the impact on SQL response time of a database upgrade from Oracle Database 10g Release 2 or a newer release to any later release by capturing a SQL tuning set on the production system, then executing it twice remotely over a database link on a test system—first to create a pre-change SQL trial, then again to create a post-change SQL trial.

Before the database upgrade can be tested, ensure that the following conditions are met:

- The production system which you are upgrading from is running Oracle Database 10g Release 2 or a newer release.
- Initially, the test system should also be running the same release of Oracle Database.
- The test system must contain an exact copy of the production data found on the production system.
- The hardware configuration must be as similar to the production system as possible.

You will also need to set up a separate SQL Performance Analyzer system running Oracle Database 11g Release 2. You will be using this system to run SQL Performance Analyzer. Neither your production data or schema need to be available on this system, since the SQL tuning set will be built using statistics stored in the SQL trace files from the production system. SQL Performance Analyzer tasks will be executed remotely on the test system to generate the execution plan and statistics for the SQL trial over a database link that you specify. The database link must be a public database link that

connects to a user with the EXECUTE privilege for the DBMS_SQLPA package and the ADVISOR privilege on the test system. You should also drop any existing PLAN_TABLE from the user's schema on the test system.

Once the upgrade environment is configured as described, perform the steps as described in the following procedure to use SQL Performance Analyzer in a database upgrade from Oracle Database 10g Release 2 or a newer release to any later release.

1. On the production system, capture the SQL workload that you intend to analyze and store it in a SQL tuning set, as described in ["Capturing the SQL Workload"](#) on page 2-3.
2. Set up the test system so that it matches the production environment as closely as possible, as described in ["Setting Up the Test System"](#) on page 2-4.
3. Transport the SQL tuning set to the SQL Performance Analyzer system.

For information about transporting SQL tuning sets using:

- Oracle Enterprise Manager, see *Oracle Database 2 Day + Performance Tuning Guide*
- APIs, see *Oracle Database SQL Tuning Guide*

4. On the SQL Performance Analyzer system, create a SQL Performance Analyzer task using the SQL tuning set as its input source.

Remotely test execute the SQL statements in the SQL tuning set on the test system over a database link to build a pre-upgrade SQL trial that will be used as a baseline for comparison, as described in ["Testing Database Upgrades from Oracle Database 10g Release 2 and Newer Releases"](#) on page 7-11.

5. Upgrade the test system.
6. Remotely test execute the SQL statements a second time on the upgraded test system over a database link to build a post-upgrade SQL trial, as described in ["Testing Database Upgrades from Oracle Database 10g Release 2 and Newer Releases"](#) on page 7-11.
7. Compare SQL performance and fix regressed SQL.

SQL Performance Analyzer compares the performance of SQL statements read from the SQL tuning set during the pre-upgrade SQL trial to those captured from the remote test execution during the post-upgrade SQL trial. A report is produced to identify any changes in execution plans or performance of the SQL statements.

If the report reveals any regressed SQL statements, you can make further changes to fix the regressed SQL, as described in ["Tuning Regressed SQL Statements After Testing a Database Upgrade"](#) on page 7-17.

Repeat the process of executing the SQL tuning set and comparing its performance to a previous execution to test any changes made until you are satisfied with the outcome of the analysis.

Testing Database Upgrades from Oracle Database 10g Release 2 and Newer Releases

Once the SQL tuning set is transported to the SQL Performance Analyzer system, you can use SQL Performance Analyzer to build a pre-upgrade SQL trial by executing or generating plans of SQL statements in the SQL tuning set on the test system. SQL Performance Analyzer test executes the SQL statements using a database link that you specify by connecting to the test system remotely and generating the execution plans and statistics for the SQL trial. The database link should exist on the SQL Performance

Analyzer system and connect to a remote user with privileges to execute the SQL tuning set on the test system.

After the pre-upgrade SQL trial is built, you need to upgrade the test system. Once the database has been upgraded, SQL Performance Analyzer will need to execute or generate plans of SQL statements in the SQL tuning set a second time on the upgraded test system to build a post-upgrade SQL trial. Alternatively, if hardware resources are available, you can use another upgraded test system to execute the second remote SQL trial. This method can be useful in helping you investigate issues identified by SQL Performance Analyzer.

You can run SQL Performance Analyzer to test a database upgrade from Oracle Database 10g Release 2 or a newer release using Oracle Enterprise Manager or APIs, as described in the following sections:

- [Testing Database Upgrades from Releases 10.2 and Higher Using Cloud Control](#)
- [Testing Database Upgrades from Releases 10.2 and Higher Using APIs](#)

Testing Database Upgrades from Releases 10.2 and Higher Using Cloud Control


To test a database upgrade from Oracle Database 10g Release 2 or a newer release using SQL Performance Analyzer:

1. From the **Performance** menu, select **SQL**, then **SQL Performance Analyzer**.
If the Database Login page appears, then log in as a user with administrator privileges.
The SQL Performance Analyzer page appears.
2. Under SQL Performance Analyzer Workflows, click **Upgrade from 10.2 or 11g**.
The Upgrade from 10.2 or 11g page appears.

Upgrade from 10.2 or 11g

Task Information

* Task Name


* SQL Tuning Set 


Description


Pre-upgrade Trial

Creation Method

Per-SQL Time Limit

 **TIP** Time limit is on elapsed time of test execution of SQL.


* Database Link 

 **TIP** Provide a PUBLIC database link connecting to a remote user with privileges to execute the Tuning Set SQL.

Post-upgrade Trial

☒ Use the same system as in the pre-upgrade trial

* Database Link

 **TIP** Same creation method and per-SQL time limit as in the pre-upgrade trial will be applied.

Trial Comparison


Comparison Metric

Schedule

Time Zone

☒ Immediately

☐ Later

Date 
(example: May 2, 2012)

Time ☐ AM ☒ PM

3. Under Task Information:

- In the Task Name field, enter the name of the task.
- In the SQL Tuning Set field, enter the name of the SQL tuning set that was built.

Alternatively, click the search icon to search for the SQL tuning set using the Search and Select: SQL Tuning Set window.

The selected SQL tuning set now appears in the SQL Tuning Set field.

- In the Description field, optionally enter a description of the task.

4. In the Creation Method field, select:

- Execute SQLs** to generate both execution plans and statistics for each SQL statement in the SQL tuning set by actually running the SQL statements remotely on the test system over a public database link.
- Generate Plans** to create execution plans remotely on the test system over a public database link without actually running the SQL statements.

5. In the Per-SQL Time Limit list, determine the time limit for SQL execution during the trial by performing one of the following actions:

- Select **5 minutes**.

The execution will run each SQL statement in the SQL tuning set up to 5 minutes and gather performance data.

- Select **Unlimited**.

The execution will run each SQL statement in the SQL tuning set to completion and gather performance data. Collecting execution statistics provides greater accuracy in the performance analysis but takes a longer time. Using this setting is not recommended because the task may be stalled by one SQL statement for a prolonged time period.

- Select **Customize** and enter the specified number of seconds, minutes, or hours.
- 6. In the Database Link field, enter the global name of a public database link connecting to a user with the EXECUTE privilege for the DBMS_SQLPA package and the ADVISOR privilege on the pre-upgrade system.

Alternatively, click the search icon to search for and select a database link, or click **Create Database Link** to create a database link using the Create Database Link page.
- 7. Under Post-upgrade Trial:
 - a. Select **Use the same system as in the pre-upgrade trial** to use the same system for executing both the pre-upgrade and post-upgrade trials.

Oracle recommends using this option to avoid possible errors due to different system configurations. When using this option, you will need to upgrade the test database to the higher database version before the post-upgrade trial is executed.
 - b. In the Database Link field, enter the global name of a public database link connecting to a user with the EXECUTE privilege for the DBMS_SQLPA package and the ADVISOR privilege on the post-upgrade system.
- 8. In the Comparison Metric list, select the comparison metric to use for the comparison analysis:
 - **Elapsed Time**
 - **CPU Time**
 - **User I/O Time**
 - **Buffer Gets**
 - **Physical I/O**
 - **Optimizer Cost**
 - **I/O Interconnect Bytes**

Optimizer Cost is the only comparison metric available if you generated execution plans only in the SQL trials.

To perform the comparison analysis by using more than one comparison metric, perform separate comparison analyses by repeating this procedure with different metrics.
- 9. Under Schedule:
 - a. In the Time Zone list, select your time zone code.
 - b. Select **Immediately** to start the task now, or **Later** to schedule the task to start at a time specified using the Date and Time fields.
- 10. Click **Submit**.

The SQL Performance Analyzer page appears.

In the SQL Performance Analyzer Tasks section, the status of this task is displayed. To refresh the status icon, click **Refresh**.

If you are using the same system to execute both the pre-upgrade and post-upgrade trials, you will need to upgrade the database after the pre-upgrade trial step is completed. After the database is upgraded, the post-upgrade trial can be executed. After the task completes, the Status field changes to Completed.

11. Under SQL Performance Analyzer Tasks, select the task and click the link in the Name column.

The SQL Performance Analyzer Task page appears.

This page contains the following sections:

- SQL Tuning Set

This section summarizes information about the SQL tuning set, including its name, owner, description, and the number of SQL statements it contains.

- SQL Trials

This section includes a table that lists the SQL trials used in the SQL Performance Analyzer task.

- SQL Trial Comparisons

This section contains a table that lists the results of the SQL trial comparisons

12. Click the icon in the Comparison Report column.

The SQL Performance Analyzer Task Result page appears.

13. Review the results of the performance analysis, as described in ["Reviewing the SQL Performance Analyzer Report Using Oracle Enterprise Manager"](#) on page 6-3.

If regressed SQL statements are found following the database upgrade, tune them as described in ["Tuning Regressed SQL Statements After Testing a Database Upgrade"](#) on page 7-17.

Testing Database Upgrades from Releases 10.2 and Higher Using APIs

This section describes how to test database upgrades from Oracle Database releases 10.2 and higher using APIs.

To test a database upgrade from releases 10.2 and higher:

1. On the system running SQL Performance Analyzer, create an analysis task.
2. Build the pre-upgrade SQL trial by performing an explain plan or test execute of SQL statements in the SQL tuning set.

Call the EXECUTE_ANALYSIS_TASK procedure using the following parameters:

- Set the task_name parameter to the name of the SQL Performance Analyzer task that you want to execute.
- Set the execution_type parameter to EXPLAIN PLAN or TEST EXECUTE:
 - If you choose to use EXPLAIN PLAN, only execution plans will be generated. Subsequent comparisons will only be able to yield a list of changed plans without making any conclusions about performance changes.
 - If you choose to use TEST EXECUTE, the SQL workload will be executed to completion. This effectively builds the pre-upgrade SQL trial using the statistics and execution plans generated from the test system. Using TEST EXECUTE is recommended to capture the SQL execution plans and

performance data at the source, thereby resulting in a more accurate analysis.

- Specify a name to identify the execution using the `execution_name` parameter. If not specified, then SQL Performance Analyzer automatically generates a name for the task execution.
- Set the `DATABASE_LINK` task parameter to the global name of a public database link connecting to a user with the `EXECUTE` privilege for the `DBMS_SQLPA` package and the `ADVISOR` privilege on the test system.

The following example executes the SQL Performance Analyzer task named `my_spa_task` and performs a test execute of the SQL statements remotely over a database link:

```
EXEC DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(task_name => 'my_spa_task', -
    execution_type => 'TEST EXECUTE', -
    execution_name => 'my_remote_trial_10g', -
    execution_params => dbms_advisor.arglist('database_link',
                                            'LINK.A.B.C.BIZ.COM'));
```

3. Build the post-upgrade SQL trial by performing an explain plan or test execute using the `EXECUTE_ANALYSIS_TASK` procedure:
 - Set the `execution_type` parameter to `EXPLAIN PLAN` or `TEST EXECUTE`:
 - If you choose to use `EXPLAIN PLAN`, only execution plans will be generated. Subsequent comparisons will only be able to yield a list of changed plans without making any conclusions about performance changes.
 - If you choose to use `TEST EXECUTE`, the SQL workload will be executed to completion. This effectively builds the post-upgrade SQL trial using the statistics and execution plans generated from the test system. Using `TEST EXECUTE` is recommended to capture the SQL execution plans and performance data at the source, thereby resulting in a more accurate analysis.
 - Set the `DATABASE_LINK` task parameter to the global name of a public database link connecting to a user with the `EXECUTE` privilege for the `DBMS_SQLPA` package and the `ADVISOR` privilege on the test system.

The following example performs a test execute of the SQL statements remotely over a database link:

```
EXEC DBMS_SQLPA.EXECUTE_ANALYSIS_TASK(task_name => 'my_spa_task', -
    execution_type => 'TEST EXECUTE', -
    execution_name => 'my_remote_trial_12c', -
    execution_params => dbms_advisor.arglist('database_link',
                                            'LINK.A.B.C.BIZ.COM'));
```

See Also:

- ["Creating an Analysis Task Using APIs"](#) on page 3-13
- ["Creating a Pre-Change SQL Trial Using APIs"](#) on page 4-4
- ["Creating a Post-Change SQL Trial Using APIs"](#) on page 5-3
- *Oracle Database PL/SQL Packages and Types Reference* to learn about the `DBMS_SQLPA.EXECUTE_ANALYSIS_TASK` function
- *Oracle Database PL/SQL Packages and Types Reference* to learn about the `DBMS_SQLPA.EXECUTE_ANALYSIS_TASK` function

Tuning Regressed SQL Statements After Testing a Database Upgrade

In some cases, SQL Performance Analyzer may identify SQL statements whose performance regressed after you upgrade the database on the test system.

You can tune the regressed SQL statements by using the SQL Tuning Advisor or SQL plan baselines, as described in [Chapter 6, "Comparing SQL Trials"](#). This involves using APIs to build a subset of a SQL tuning set with only the regressed SQL statements, transport this subset of regressed SQL statements to the remote database, and running the SQL Tuning Advisor on the remote database.

Oracle Enterprise Manager does not provide support for fixing regressions after running SQL Performance Analyzer involving one or more remote SQL trials. For more information, see ["Tuning Regressed SQL Statements From a Remote SQL Trial Using APIs"](#) on page 6-24.

If you are upgrading from Oracle Database 10g Release 2 and newer releases, you can also create SQL plan baselines to instruct the optimizer to select existing execution plans in the future. For more information, see ["Creating SQL Plan Baselines Using APIs"](#) on page 6-26.

Part II

Database Replay

Database Replay enables you to replay a full production workload on a test system to assess the overall impact of system changes.

Part II covers Database Replay and contains the following chapters:

- [Chapter 8, "Introduction to Database Replay"](#)
- [Chapter 9, "Capturing a Database Workload"](#)
- [Chapter 10, "Preprocessing a Database Workload"](#)
- [Chapter 11, "Replaying a Database Workload"](#)
- [Chapter 12, "Analyzing Captured and Replayed Workloads"](#)
- [Chapter 13, "Using Workload Intelligence"](#)
- [Chapter 14, "Using Consolidated Database Replay"](#)
- [Chapter 15, "Using Workload Scale-Up"](#)

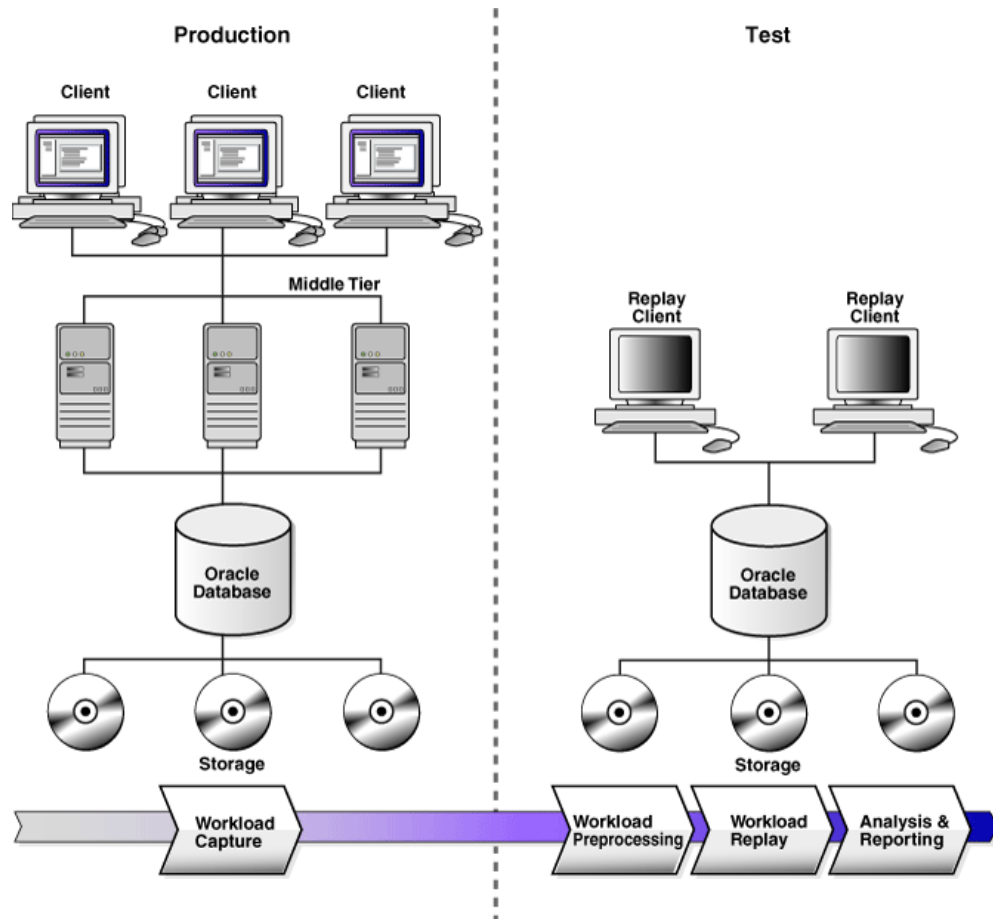
Introduction to Database Replay

You can use Database Replay to capture a workload on the production system and replay it on a test system with the exact timing, concurrency, and transaction characteristics of the original workload. This enables you to test the effects of a system change without affecting the production system.

Database Replay supports workload capture on a system running Oracle Database 10g Release 2 and newer releases. In order to capture a workload on a system running Oracle Database 10g Release 2, the database version must be 10.2.0.4 or higher. Workload replay is only supported on systems running Oracle Database 11g Release 1 and newer releases.

Note: To use the workload capture feature on a system running Oracle9i Database or an earlier version of Oracle Database 10g, refer to My Oracle Support note ID 560977.1 for information about the required patches, or contact Oracle Support for more information.

Analyzing the effect of system changes using Database Replay involves the following steps, as illustrated in [Figure 8-1](#):

Figure 8–1 Database Replay Workflow

1. On the production system, capture the workload into capture files, as described in ["Workload Capture"](#) on page 8-2.
2. Copy the capture files to the test system and preprocess them, as described in ["Workload Preprocessing"](#) on page 8-3.
3. On the test system, replay the preprocessed files, as described in ["Workload Replay"](#) on page 8-3.
4. Using the reports generated by Database Replay, perform detailed analysis of both the workload capture and workload replay, as described in ["Analysis and Reporting"](#) on page 8-3.

Workload Capture

The first step in using Database Replay is to capture the production workload. Capturing a workload involves recording all requests made by external clients to Oracle Database.

When workload capture is enabled, all external client requests directed to Oracle Database are tracked and stored in binary files—called capture files—on the file system. You can specify the location where the capture files will be stored. Once workload capture begins, all external database calls are written to the capture files. The capture files contain all relevant information about the client request, such as SQL

text, bind values, and transaction information. Background activities and database scheduler jobs are not captured. These capture files are platform independent and can be transported to another system.

See Also:

- [Chapter 9, "Capturing a Database Workload"](#) for information about how to capture a workload on the production system

Workload Preprocessing

Once the workload has been captured, the information in the capture files must be preprocessed. Preprocessing creates all necessary metadata needed for replaying the workload. This must be done once for every captured workload before they can be replayed. After the captured workload is preprocessed, it can be replayed repeatedly on a replay system running the same version of Oracle Database. Typically, the capture files should be copied to a test system for preprocessing. As workload preprocessing can be time consuming and resource intensive, it is recommended that this step be performed on the test system where the workload will be replayed.

See Also:

- [Chapter 10, "Preprocessing a Database Workload"](#) for information about how to preprocess a captured workload

Workload Replay

After a captured workload has been preprocessed, it can be replayed on a test system. During the workload replay phase, Oracle Database performs the actions recorded during the workload capture phase on the test system by re-creating all captured external client requests with the same timing, concurrency, and transaction dependencies of the production system.

Database Replay uses a client program called the replay client to re-create all external client requests recorded during workload capture. Depending on the captured workload, you may need one or more replay clients to properly replay the workload. A calibration tool is provided to help determine the number of replay clients needed for a particular workload. Because the entire workload is replayed—including DML and SQL queries—the data in the replay system should be as logically similar to the data in the capture system as possible. This will minimize replay divergence and enable a more reliable analysis of the replay.

See Also:

- [Chapter 11, "Replaying a Database Workload"](#) for information about how to replay a preprocessed workload on the test system

Analysis and Reporting

Once the workload is replayed, in-depth reporting is provided for you to perform detailed analysis of both workload capture and replay.

The workload capture report and workload replay report provide basic information about the workload capture and replay, such as errors encountered during replay and data divergence in rows returned by DML or SQL queries. A comparison of several statistics—such as database time, average active sessions, and user calls—between the workload capture and the workload replay is also provided.

The replay compare period report can be used to perform a high-level comparison of one workload replay to its capture or to another replay of the same capture. A divergence summary with an analysis of whether any data divergence occurred and if there were any significant performance changes is also provided. Furthermore, Automatic Database Diagnostic Monitor (ADDM) findings are incorporated into these reports.

For advanced analysis, Automatic Workload Repository (AWR) reports are available to enable detailed comparison of performance statistics between the workload capture and the workload replay. The information available in these reports is very detailed, and some differences between the workload capture and replay can be expected. Furthermore, Workload Intelligence operates on data recorded during a workload capture to create a model that describes the workload. This model can be used to identify significant patterns in templates that are executed as part of the workload. For each pattern, you can view important statistics, such as the number of executions of a given pattern and the database time consumed by the pattern during its execution.

The SQL Performance Analyzer report can be used to compare a SQL tuning set from a workload capture to another SQL tuning set from a workload replay, or two SQL tuning sets from two workload replays. Comparing SQL tuning sets with Database Replay provides more information than SQL Performance Analyzer test-execute because it considers and shows all execution plans for each SQL statement, while SQL Performance Analyzer test-execute generates only one execution plan per SQL statement for each SQL trial. Moreover, the SQL statements are executed in a more authentic environment because Database Replay captures all bind values and reproduces dynamic session state such as PL/SQL package state more accurately. It is recommended that you run SQL Performance Analyzer test-execute first as a sanity test to ensure SQL statements have not regressed and the test system is set up properly before using Database Replay to perform load and currency testing.

Besides using replay divergence information to analyze replay characteristics of a given system change, you should also use an application-level validation procedure to assess the system change. Consider developing a script to assess the overall success of the replay. For example, if 10,000 orders are processed during workload capture, you should validate that a similar number of orders are also processed during replay.

After the replay analysis is complete, you can restore the database to its original state at the time of workload capture and repeat workload replay to test other changes to the system.

See Also:

- [Chapter 12, "Analyzing Captured and Replayed Workloads"](#) for information about how to analyze data and performance divergence using Database Replay reports

Capturing a Database Workload

This chapter describes how to capture a database workload on the production system. The first step in using Database Replay is to capture the production workload. For more information about how capturing a database workload fits within the Database Replay architecture, see "[Workload Capture](#)" on page 8-2.

This chapter contains the following sections:

- [Prerequisites for Capturing a Database Workload](#)
- [Setting Up the Capture Directory](#)
- [Workload Capture Options](#)
- [Workload Capture Restrictions](#)
- [Enabling and Disabling the Workload Capture Feature](#)
- [Enterprise Manager Privileges and Roles](#)
- [Capturing a Database Workload Using Enterprise Manager](#)
- [Monitoring Workload Capture Using Enterprise Manager](#)
- [Capturing a Database Workload Using APIs](#)
- [Monitoring Workload Capture Using Views](#)

Prerequisites for Capturing a Database Workload

Before starting a workload capture, you should have a strategy in place to restore the database on the test system. Before a workload can be replayed, the logical state of the application data on the replay system should be similar to that of the capture system when replay begins. To accomplish this, consider using one of the following methods:

- Recovery Manager (RMAN) `DUPLICATE` command
- Snapshot standby
- Data Pump Import and Export

This will allow you to restore the database on the replay system to the application state as of the workload capture start time.

See Also:

- *Oracle Database Backup and Recovery User's Guide* for information about duplicating a database using RMAN
- *Oracle Data Guard Concepts and Administration* for information about managing snapshot standby databases
- *Oracle Database Utilities* for information about using Data Pump

Setting Up the Capture Directory

Determine the location and set up a directory where the captured workload will be stored. Before starting the workload capture, ensure that the directory is empty and has ample disk space to store the workload. If the directory runs out of disk space during a workload capture, the capture will stop. To estimate the amount of disk space that is required, you can run a test capture on your workload for a short duration (such as a few minutes) to extrapolate how much space you will need for a full capture. To avoid potential performance issues, you should also ensure that the target replay directory is mounted on a separate file system.

For Oracle RAC, consider using a shared file system. Alternatively, you can set up one capture directory path that resolves to separate physical directories on each instance, but you will need to consolidate the files created in each of these directories into a single directory. For captures on an Oracle RAC database, Enterprise Manager only supports Oracle RAC configured with a shared file system. The entire content of the local capture directories on each instance (not only the capture files) must be copied to the shared directory before it can be used for preprocessing or data masking. For example, assume that you are:

- Running an Oracle RAC environment in Linux with two database instances named `host1` and `host2`
- Using a capture directory object named `CAPDIR` that resolves to `/$ORACLE_HOME/rdbms/capture` on both instances
- Using a shared directory that resides in `/nfs/rac_capture`

You will need to login into each host and run the following command:

```
cp -r $ORACLE_HOME/rdbms/capture/* /nfs/rac_capture
```

After this is done for both instances, the `/nfs/rac_capture` shared directory is ready to be preprocessed or masked.

Workload Capture Options

Proper planning before workload capture is required to ensure that the capture will be accurate and useful when replayed in another environment.

Before capturing a database workload, carefully consider the following options:

- [Restarting the Database](#)
- [Using Filters with Workload Capture](#)

Restarting the Database

While this step is not required, Oracle recommends that the database be restarted before capturing the workload to ensure that ongoing and dependent transactions are allowed to be completed or rolled back before the capture begins. If the database is not

restarted before the capture begins, transactions that are in progress or have yet to be committed will not be fully captured in the workload. Ongoing transactions will thus not be replayed properly, because only the part of the transaction whose calls were captured will be replayed. This may result in undesired replay divergence when the workload is replayed. Any subsequent transactions with dependencies on the incomplete transactions may also generate errors during replay. On a busy system, it is normal to see some replay divergence, but the replay can still be used to perform meaningful analysis of a system change if the diverged calls do not make up a significant portion of the replay in terms of DB time and other such key attributes.

Before restarting the database, determine an appropriate time to shut down the production database before the workload capture when it is the least disruptive. For example, you may want to capture a workload that begins at 8:00 a.m. However, to avoid service interruption during normal business hours, you may not want to restart the database during this time. In this case, you should consider starting the workload capture at an earlier time, so that the database can be restarted at a time that is less disruptive.

Once the database is restarted, it is important to start the workload capture before any user sessions reconnect and start issuing any workload. Otherwise, transactions performed by these user sessions will not be replayed properly in subsequent database replays, because only the part of the transaction whose calls were executed after the workload capture is started will be replayed. To avoid this problem, consider restarting the database in `RESTRICTED` mode using `STARTUP RESTRICT`, which will only allow the `SYS` user to login and start the workload capture. By default, once the workload capture begins, any database instance that are in `RESTRICTED` mode will automatically switch to `UNRESTRICTED` mode, and normal operations can continue while the workload is being captured.

Only one workload capture can be performed at any given time. If you have a Oracle Real Application Clusters (Oracle RAC) configuration, workload capture is performed for the entire database. Once you enable capture for one of the Oracle RAC nodes, workload capture is started on all database instances (the workload capture process is Oracle RAC aware). Although it is not required, restarting all instances in a Oracle RAC configuration before workload capture is recommended to avoid capturing ongoing transactions.

To restart all instances in a Oracle RAC configuration before workload capture:

1. Shut down all the instances.
2. Restart all the instances.
3. Start workload capture.
4. Connect the application and start the user workload.

See Also:

- *Oracle Database Administrator's Guide* for information about restricting access to an instance at startup

Using Filters with Workload Capture

By default, all user sessions are recorded during workload capture. You can use workload filters to specify which user sessions to include in or exclude from the workload during workload capture. There are two types of workload filters: inclusion filters and exclusion filters. You can use either inclusion filters or exclusion filters in a workload capture, but not both.

Inclusion filters enable you to specify user sessions that will be captured in the workload. This is useful if you want to capture only a subset of the database workload.

Exclusion filters enable you to specify user sessions that will not be captured in the workload. This is useful if you want to filter out session types that do not need to be captured in the workload, such as those that monitor the infrastructure—like Oracle Enterprise Manager (EM) or Statspack—or other such processes that are already running on the test system. For example, if the system where the workload will be replayed is running EM, replaying captured EM sessions on the system will result in duplication of workload. In this case, you may want to use exclusion filters to filter out EM sessions.

Workload Capture Restrictions

Certain types of user sessions and client requests may sometimes be captured in a workload, but they are not supported by Database Replay. Capturing these session and request types in a workload may result in errors during workload replay.

The following types of user sessions and client requests are not supported by Database Replay:

- Direct path load of data from external files using utilities such as SQL*Loader
- Non-PL/SQL based Advanced Queuing (AQ)
- Flashback queries
- Oracle Call Interface (OCI) based object navigations
- Non SQL-based object access
- Distributed transactions

Any distributed transactions that are captured will be replayed as local transactions.

- XA transactions

XA transactions are not captured or replayed. All local transactions are captured.

- JAVA_XA transactions

If the workload uses the JAVA_XA package, JAVA_XA function and procedure calls are captured as normal PL/SQL workload. To avoid problems during workload replay, consider dropping the JAVA_XA package on the replay system to enable the replay to complete successfully.

- Database Resident Connection Pooling (DRCP)
- Workloads using OUT binds
- Multi-threaded Server (MTS) and shared server sessions with synchronization mode set to OBJECT_ID
- Migrated sessions

The workload is captured for migrated sessions. However, user logins or session migration operations are not captured. Without a valid user login or session migration, the replay may cause errors because the workload may be replayed by a wrong user.

Typically, Database Replay refrains from capturing these types of non-supported user sessions and client requests. Even when they are captured, Database Replay will not replay them. Therefore, it is usually not necessary to manually filter out

non-supported user sessions and client requests. In cases where they are captured and found to cause errors during replay, consider using workload capture filters to exclude them from the workload.

See Also:

- ["Using Filters with Workload Capture"](#) on page 9-3 for information about using workload capture filters
- ["Using Filters with Workload Replay"](#) on page 11-4 for information about using workload replay filters

Enabling and Disabling the Workload Capture Feature

Oracle Database 10g Release 2 supports using Database Replay to capture a database workload that can be used to test database upgrades to Oracle Database 11g and subsequent releases. To use this feature, it must be enabled on the capture system running Oracle Database 10g Release 2 before a workload can be captured. By default, the workload capture feature is not enabled in Oracle Database 10g Release 2 (10.2). You can enable or disable this feature by specifying the `PRE_11G_ENABLE_CAPTURE` initialization parameter.

Note: It is only necessary to enable the workload capture feature if you are capturing a database workload on a system running Oracle Database 10g Release 2.

If you are capturing a database workload on a system running Oracle Database 11g Release 1 or a later release, it is not necessary to enable the workload capture feature because it is enabled by default. Furthermore, the `PRE_11G_ENABLE_CAPTURE` initialization parameter is only valid with Oracle Database 10g Release 2 (10.2) and cannot be used with subsequent releases.

To enable the workload capture feature on a system running Oracle Database 10g Release 2, run the `wrrenbl.sql` script at the SQL prompt:

```
@$ORACLE_HOME/rdbms/admin/wrrenbl.sql
```

The `wrrenbl.sql` script calls the `ALTER SYSTEM SQL` statement to set the `PRE_11G_ENABLE_CAPTURE` initialization parameter to `TRUE`. If a server parameter file (spfile) is being used, the `PRE_11G_ENABLE_CAPTURE` initialization parameter will be modified for the currently running instance and recorded in the spfile, so that the new setting will persist when the database is restarted. If a spfile is not being used, the `PRE_11G_ENABLE_CAPTURE` initialization parameter will only be modified for the currently running instance, and the new setting will not persist when the database is restarted. To make the setting persistent without using a spfile, you will need to manually specify the parameter in the initialization parameter file (`init.ora`).

To disable workload capture, run the `wrrdsbl.sql` script at the SQL prompt:

```
@$ORACLE_HOME/rdbms/admin/wrrdsbl.sql
```

The `wrrdsbl.sql` script calls the `ALTER SYSTEM SQL` statement to set the `PRE_11G_ENABLE_CAPTURE` initialization parameter to `FALSE`. If a server parameter file (spfile) is being used, the `PRE_11G_ENABLE_CAPTURE` initialization parameter will be modified for the currently running instance and also recorded in the spfile, so that the new setting will persist when the database is restarted. If a spfile is not being used, the `PRE_11G_`

ENABLE_CAPTURE initialization parameter will only be modified for the currently running instance, and the new setting will not persist when the database is restarted. To make the setting persistent without using a spfile, you will need to manually specify the parameter in the initialization parameter file (`init.ora`).

Note: The `PRE_11G_ENABLE_CAPTURE` initialization parameter can only be used with Oracle Database 10g Release 2 (10.2). This parameter is not valid in subsequent releases. After upgrading the database, you will need to remove the parameter from the server parameter file (spfile) or the initialization parameter file (`init.ora`); otherwise, the database will fail to start up.

See Also:

- *Oracle Database Reference* for more information about the `PRE_11G_ENABLE_CAPTURE` initialization parameter

Enterprise Manager Privileges and Roles

The Database Replay resource type privileges enable you to view or operate any Database Replay entities. Additionally, you need the target operator privilege for the target from which the workload was captured to access the entities associated with the workload. For a target that does not exist anymore, the Enterprise Manager user who owns the entities or the Enterprise Manager super user can still access the entities.

The two security roles discussed in the following sections make it easier to grant or revoke privileges related to Database Replay entities.

Database Replay Viewer Role

Users who have the Database Replay Viewer role can view any Database Replay entity. By default, no Enterprise Manager user is granted this role. However, the `EM_ALL_VIEWER` role includes this role by default.

The Database Replay Viewer role consists of the following privilege:

- Database Replay Viewer (resource type privilege)

Database Replay Operator Role

The Database Replay Operator role includes the Database Replay Viewer role and thus its privileges. Users who have the Database Replay Operator role can also edit and delete any Database Replay entity. By default, no Enterprise Manager user is granted this role. However, the `EM_ALL_OPERATOR` role includes this role by default.

The Database Replay Operator role consists of the following privileges:

- Database Replay Operator (resource type privilege)
- Create new Named Credential (resource type privilege)
- Create new job (resource type privilege)
- Connect to any viewable target (target type privilege)
- Execute Command Anywhere (target type privilege)

To capture or replay a workload on a database target, an Enterprise Manager user needs all the privileges granted by the Database Replay Operator role plus the target operator privilege for the database target.

Capturing a Database Workload Using Enterprise Manager

This section describes how to capture a database workload using Enterprise Manager. The primary tool for capturing database workloads is Oracle Enterprise Manager.

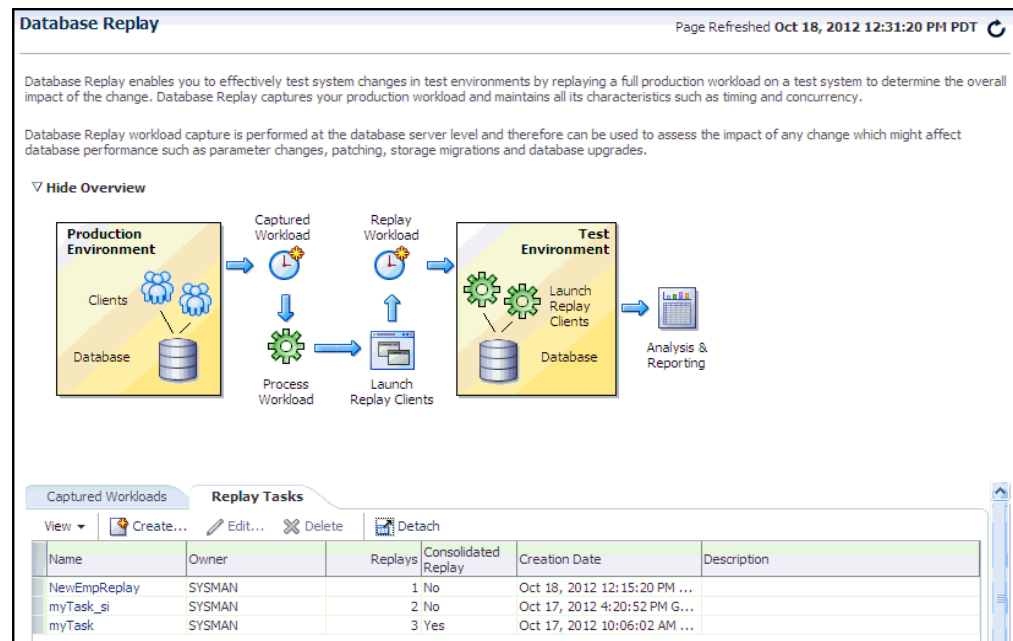
Tip: If Oracle Enterprise Manager is unavailable, you can capture database workloads using APIs, as described in ["Capturing a Database Workload Using APIs"](#) on page 9-16.

To capture a database workload using Enterprise Manager:

1. From the Enterprise menu of the Enterprise Manager Cloud Control console, select **Quality Management**, then **Database Replay**.

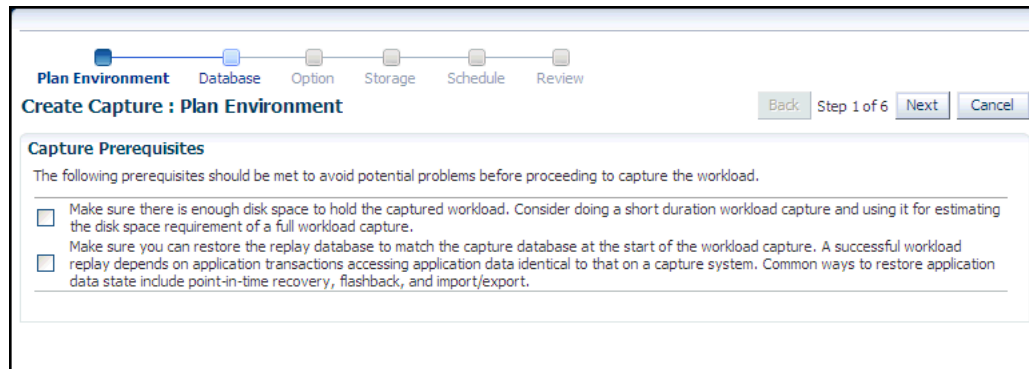
If the Database Login page appears, log in as a user with administrator privileges.

The Database Replay page appears.



2. From the Database Replay page, click the **Captured Workloads** tab, then click **Create** in the toolbar.

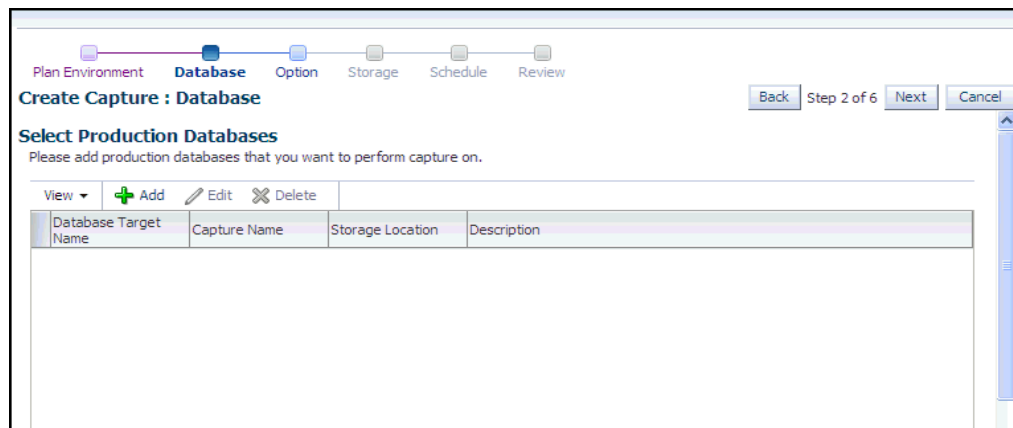
The Create Capture: Plan Environment page appears.



3. Verify that you have met both prerequisites described on this page, then enable both checkboxes and click **Next**.

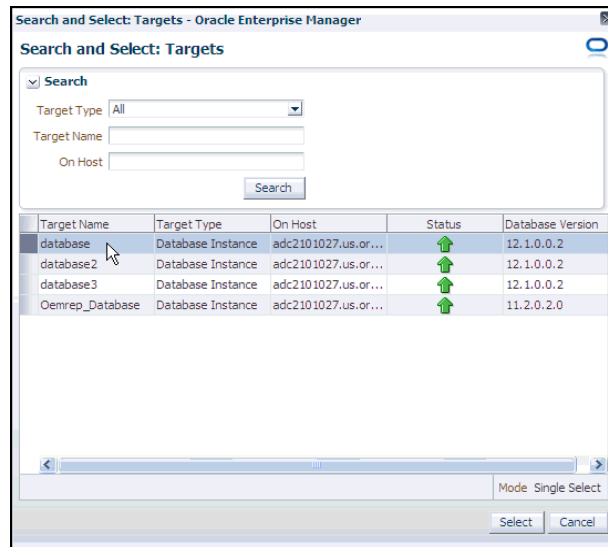
The Create Capture: Database page appears.

Tip: For information about the prerequisites, see "[Prerequisites for Capturing a Database Workload](#)" on page 9-1.



4. Click **Add**.
5. Provide a capture name, provide an optional description, then click the Target Database search icon.

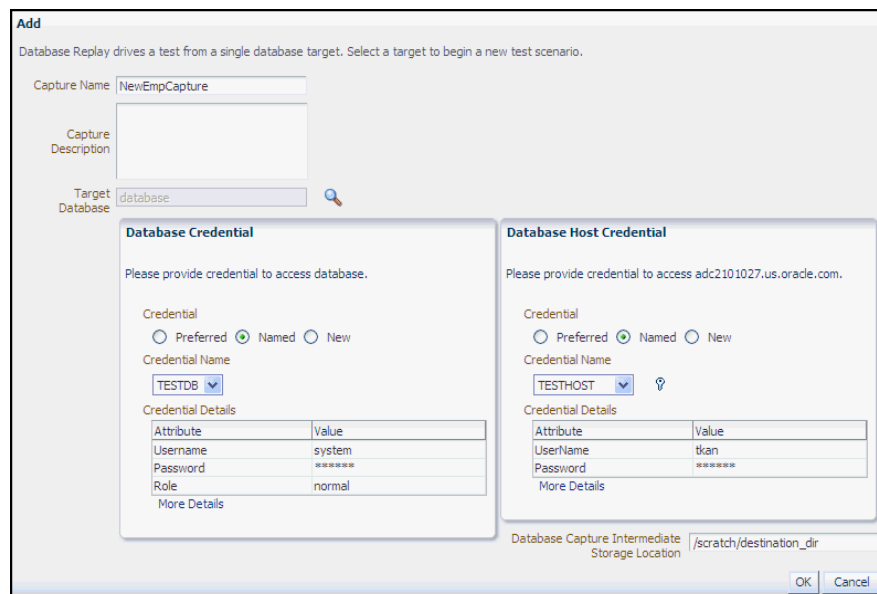
The Search and Select: Targets pop-up appears.



6. Choose a target database from the list, then click **Select**.

The Add pop-up reappears with added sections for Database Credential and Database Host Credential.

7. Provide database credentials, database host credentials, a Database Capture Intermediate Storage Location, then click **OK**.
 - The capture database places the capture files in the intermediate storage location. After the capture, the files are copied to the storage location. For captures on an Oracle RAC database, Enterprise Manager only supports Oracle RAC configured with a shared file system.



The selected target database now appears in the list of databases in the Select Production Databases table.

8. Click **Next**.

The **Create Capture: Options** page appears.

Create Capture : Option

Plan Environment Database **Option** Storage Schedule Review

Back Step 3 of 6 Next Cancel

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

☒ Capture SQL statements into a SQL Tuning Set during workload capture.

Workload Filters
Workload filters can customize the workload to be captured. By default, most external client requests made to the database are captured. Refer to the Oracle Real Application Testing User's Guide for more information.

Filter Mode: **Exclusion**

Excluded Sessions
All sessions will be captured except for those listed below.

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

9. Select the workload capture options:

- Under the SQL Performance Analyzer section, select whether to capture SQL statements into a SQL tuning set during workload capture.

While Database Replay provides an analysis of how a change affects your entire system, you can use a SQL tuning set in conjunction with the SQL Performance Analyzer to gain a more SQL-centric analysis of how the change affects SQL statements and execution plans.

By capturing a SQL tuning set during workload capture and another SQL tuning set during workload replay, you can use the SQL Performance Analyzer to compare these SQL tuning sets to each other without having to re-execute the SQL statements. This enables you to obtain a SQL Performance Analyzer report and compare the SQL performance, before and after changes, while running Database Replay.

Note: Capturing SQL statements into a SQL Tuning Set is the default, and is the recommended workload capture option. Capturing SQL statements into a SQL Tuning Set is not available for Oracle RAC.

Tip: For information about comparing SQL tuning sets using SQL Performance Analyzer reports, see ["Generating SQL Performance Analyzer Reports Using APIs"](#) on page 12-14.

- Under the Workload Filters section, select whether to use exclusion filters by selecting **Exclusion** in the Filter Mode list, or inclusion filters by selecting **Inclusion** in the Filter Mode list.

To add filters, click **Add** and enter the filter name, session attribute, and value in the corresponding fields.

Tip: For more information, see ["Using Filters with Workload Capture"](#) on page 9-3.

After selecting the desired workload capture options, click **Next**.

The Create Capture: Storage page appears.

Create Capture : Storage

Ensure there is sufficient free disk space on the selected host system to store the capture. You should consider performing a short duration capture, and using it as the basis for estimating the requirements for a full capture.

Storage Host

Back Step 4 of 6 Next Cancel

10. Click the Storage Host icon, choose a target from the list, then click **Select**.

The Storage page now requests Host Credentials and a Storage Location.

11. Provide Host Credentials, click **Browse** to select a Storage Location, select the location and click **OK**, then click **Next**.

The Create Capture: Schedule page appears.

Create Capture : Schedule

Schedule start time and duration for capture.

Start ☒ Immediately ☐ Later (GMT-08:00) Los Angeles - Pacific Time (PT)

Duration ☐ Indefinitely ☒ For minutes ☐ Until

Automatic Workload Repository

Exporting of AWR data during capture enables performance comparison of the database between capture and replay. Because AWR data exportation can have a significant impact on the production system, it should be scheduled at a time that will not negatively impact the production system. Selecting "Immediately" will schedule AWR data to be exported immediately after capture is completed (as scheduled in the subsequent Schedule step).

Start ☒ Immediately ☐ Later (GMT-08:00) Los Angeles - Pacific Time (PT)

Back Step 5 of 6 Next Cancel

12. Schedule the starting time and duration for the capture, schedule the exporting of AWR data, then click **Next**.

- The default capture duration is 5 minutes. Change the capture duration to capture representative activity over a time period of interest that needs to be tested.

The Create Capture: Review page appears.

Create Capture : Review

Database

Concurrent Capture No

Select Production Databases

View

Database Target Name	Name	Description	Storage Location
cdb121	MyCapture		/scratch/userxyz

Back Step 6 of 6 Submit Cancel

13. If all of the parameters appear as you have intended, click **Submit** to start the capture job.

- The "Capture SQL statements into a SQL Tuning Set during workload capture" option is enabled by default. Uncheck this option if you do not want to compare SQL tuning sets at the end of the Replay.

The Database Replay page reappears, displays a message that the capture was created successfully, and displays the status of the capture in the Captures list, such as "Scheduled."

14. For detailed information about the capture, double-click the name of the capture.

The Capture Summary page appears, and displays several attributes, including the average active sessions, a workload comparison, and related concurrent captures, if any.

Tip: After capturing a workload on the production system, you need to preprocess the captured workload, as described in [Chapter 10](#), "Preprocessing a Database Workload".

Capturing Workloads from Multiple Databases Concurrently

Concurrent capture refers to capturing the workload on multiple databases simultaneously.

To capture a concurrent database replay workload:

1. From the Enterprise menu of the Enterprise Manager Cloud Control console, select **Quality Management**, then **Database Replay**.

If the Database Login page appears, log in as a user with administrator privileges.

The Database Replay page appears.

2. From the Database Replay page, click the **Captured Workloads** tab, then click **Create** in the toolbar.

The Create Capture: Plan Environment page appears.

3. Make sure that you have met both prerequisites described on this page, then enable both checkboxes and click **Next**.

The Create Capture: Database page appears.

Tip: For information about the prerequisites, see "[Prerequisites for Capturing a Database Workload](#)" on page 9-1.

4. Click **Add**.

The Add pop-up appears.

5. Provide a Capture name, provide an optional description, then click the Target Database search icon.

The Search and Select: Targets pop-up appears.

6. Choose a target database from the list, then click **Select**.

The Add pop-up reappears with added sections for Database Credential and Database Host Credential.

7. Provide database credentials, database host credentials, a Database Capture Intermediate Storage Location, then click **OK**.

- The Capture database places the Capture files in the intermediate storage location. After the Capture, the files are copied to the storage location.

The selected target database now appears in the list of databases in the Select Production Databases table.

8. Add another database for concurrent capture:

- a. Follow the instructions in steps 4 through 7.

The Create Capture: Database page reappears, and displays the additional database for capture along with the first database you specified in steps 4 through 7.

- b. Provide a name and optional description for the concurrent capture, then click **Next**.

The Create Capture: Options page appears.

9. Select the workload capture options:

- Under the SQL Performance Analyzer section, select whether to capture SQL statements into a SQL tuning set during workload capture.

While Database Replay provides an analysis of how a change affects your entire system, you can use a SQL tuning set in conjunction with the SQL Performance Analyzer to gain a more SQL-centric analysis of how the change affects SQL statements and execution plans.

By capturing a SQL tuning set during workload capture and another SQL tuning set during workload replay, you can use the SQL Performance Analyzer to compare these SQL tuning sets to each other without having to re-execute the SQL statements. This enables you to obtain a SQL Performance Analyzer report and compare the SQL performance, before and after changes, while running Database Replay.

Note: Capturing SQL statements into a SQL tuning set is the default and recommended workload capture option.

- Under the Workload Filters section, select whether to use exclusion filters by selecting **Exclusion** in the Filter Mode list, or inclusion filters by selecting **Inclusion** in the Filter Mode list.

To add filters, click **Add** and enter the filter name, session attribute, and value in the corresponding fields.

After selecting the desired workload capture options, click **Next**.

The Create Capture: Storage page appears.

10. Click the Storage Host icon, choose a target from the list, then click **Select**.

The Storage page now requests Host Credentials and a Storage Location.

11. Provide Host Credentials, click **Browse** to select a Storage Location, then click **Next**.

The Create Capture: Schedule page appears.

12. Schedule the starting time and duration for the capture, schedule the exporting of AWR data, then click **Next**.

- The default capture duration is 5 minutes. Change the capture duration to capture representative activity over a time period of interest that needs to be tested.

The Create Capture: Review page appears.

13. If all of the parameters appear as you have intended, click **Submit** to start the Capture job.

- The "Capture SQL statements into a SQL Tuning Set during workload capture" option is enabled by default. Uncheck this option if you do not want to compare SQL tuning sets at the end of the Replay.

The Database Replay page reappears, displays a message that the capture was created successfully, and displays the status of the capture in the Captures list, such as "Scheduled."

14. For detailed information about the capture, double-click the name of the capture.

The Capture Summary page appears, and displays several attributes, including the average active sessions, a workload comparison, and related concurrent captures.

Monitoring Workload Capture Using Enterprise Manager

This section describes how to monitor workload capture using Enterprise Manager. The primary tool for monitoring workload capture is Oracle Enterprise Manager. Using Enterprise Manager, you can:

- Monitor or stop an active workload capture
- View a completed workload capture

Tip: If Oracle Enterprise Manager is unavailable, you can monitor workload capture using views, as described in "[Monitoring Workload Capture Using Views](#)" on page 9-20.

This section contains the following topics:

- [Monitoring an Active Workload Capture](#)
- [Stopping an Active Workload Capture](#)
- [Viewing a Completed Workload Capture](#)

Monitoring an Active Workload Capture

This section describes how to monitor an active workload capture using Enterprise Manager.

To monitor an active workload capture:

1. From the Enterprise menu of the Enterprise Manager Cloud Control console, select **Quality Management**, then **Database Replay**.

If the Database Login page appears, log in as a user with administrator privileges.

The Database Replay page appears.

2. From the Captured Workloads tab of the Database Replay page for a capture that has a Status other than Completed, click the name of the desired capture from the Capture table.

The Summary tab of the Database Replay page appears, showing detailed statistics, a chart for average active sessions that updates dynamically while the capture is in progress, a comparison of data for captured elements versus the same elements not captured, and related concurrent captures, if any.

- The Not Captured data shown in the Active Average Sessions chart shows the database activity (database sessions) that is not being captured.

- The values for the Total column in the Comparison section shows all of the captured and uncaptured activity in the database. Filtering, as determined by the Workload Filters you provided in the Options step of the Create Capture wizard, is primarily why some activity is captured or not. Additionally, background activities, database scheduler jobs, and unreplayable calls are not captured.
 - You can click the refresh icon in the upper right corner to update the capture while it is running.
3. To return to the Database Replay page, click the Database Replay breadcrumb.

Stopping an Active Workload Capture

This section describes how to stop an active workload capture using Enterprise Manager.

To stop an active workload capture:

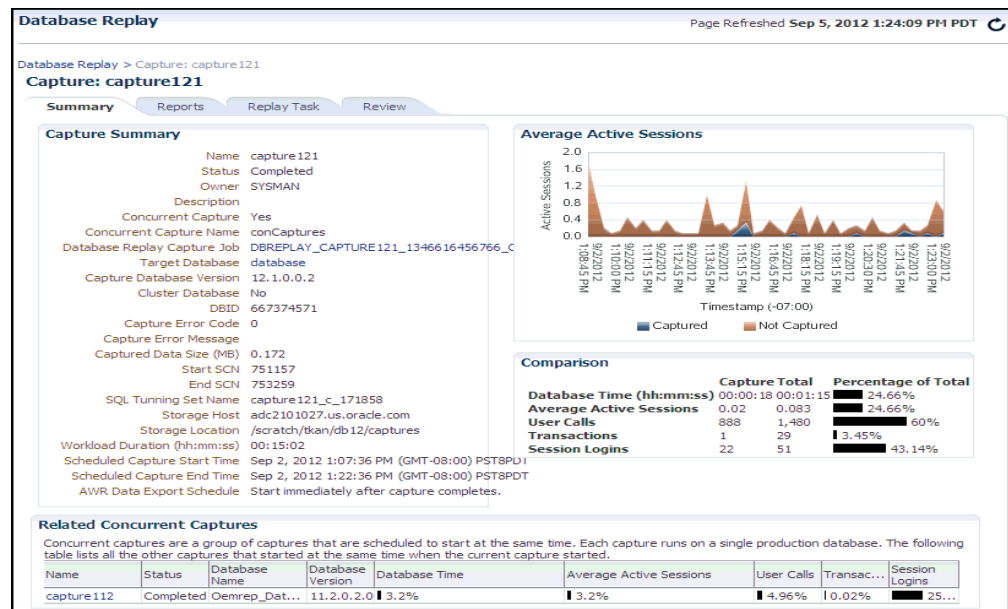
1. From the Enterprise menu of the Enterprise Manager Cloud Control console, select **Quality Management**, then **Database Replay**.
If the Database Login page appears, log in as a user with administrator privileges.
The Database Replay page appears.
2. From the Captured Workloads tab of the Database Replay page for a capture that has a Status of Draft, click the name in the Capture table of the capture you want to stop.
The Capture Summary page appears.
3. Click the **Stop Capture** button.
The button label changes to Stopping Capture. When the process completes, the Status changes to Stopped.

Viewing a Completed Workload Capture

This section describes how to manage a completed workload capture using Enterprise Manager.

To view a completed workload capture:

1. From the Enterprise menu of the Enterprise Manager Cloud Control console, select **Quality Management**, then **Database Replay**.
If the Database Login page appears, log in as a user with administrator privileges.
The Database Replay page appears.
2. From the Capture Workloads tab of the Database Replay page, click the name of a capture that has a Status of Completed.
The contents of the Summary tab of the Database Replay page appears as for a capture in progress, except the Average Active Sessions chart shows aggregated data for the capture time period, rather than dynamic data recorded during the capture.



The Average Active Sessions chart provides a graphic display of the captured session activity compared to the uncaptured session activity (such as background activities or filtered sessions). This chart appears only when Active Session History (ASH) data is available for the capture period.

Under Comparison, various statistics for the workload capture are displayed:

- **Capture column**
Displays the statistics for the captured session activity.
- **Total column**
Displays the statistics for the total session activity.
- **Percentage of Total column**
Displays the percentage of total session activity that has been captured in the workload.

3. To return to the Database Replay page, click the Database Replay breadcrumb.

Tip: See [Chapter 12, "Analyzing Captured and Replayed Workloads"](#) for information about accessing workload capture reports.

Capturing a Database Workload Using APIs

This section describes how to capture a database workload using APIs. You can also use Oracle Enterprise Manager to capture database workloads, as described in ["Capturing a Database Workload Using Enterprise Manager"](#) on page 9-7.

Capturing a database workload using the DBMS_WORKLOAD_CAPTURE package involves:

- [Defining Workload Capture Filters](#)
- [Starting a Workload Capture](#)
- [Stopping a Workload Capture](#)
- [Exporting AWR Data for Workload Capture](#)
- [Importing AWR Data for Workload Capture](#)

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about the DBMS_WORKLOAD_CAPTURE package

Defining Workload Capture Filters

This section describes how to add and remove workload capture filters. For information about using workload filters with workload capture, see ["Using Filters with Workload Capture"](#) on page 9-3.

To add filters to a workload capture:

- Use the ADD_FILTER procedure:

```
BEGIN
  DBMS_WORKLOAD_CAPTURE.ADD_FILTER (
                                fname => 'user_ichan',
                                fattribute => 'USER',
                                fvalue => 'ICHAN');
END;
/
```

In this example, the ADD_FILTER procedure adds a filter named user_ichan, which can be used to filter out all sessions belonging to the user name ICHAN.

The ADD_FILTER procedure in this example uses the following parameters:

- The fname required parameter specifies the name of the filter that will be added.
- The fattribute required parameter specifies the attribute on which the filter will be applied. Valid values include PROGRAM, MODULE, ACTION, SERVICE, INSTANCE_NUMBER, and USER.
- The fvalue required parameter specifies the value for the corresponding attribute on which the filter will be applied. It is possible to use wildcards such as % with some of the attributes, such as modules and actions.

To remove filters from a workload capture:

- Use the DELETE_FILTER procedure:

```
BEGIN
  DBMS_WORKLOAD_CAPTURE.DELETE_FILTER (fname => 'user_ichan');
END;
/
```

In this example, the DELETE_FILTER procedure removes the filter named user_ichan from the workload capture.

The DELETE_FILTER procedure in this example uses the fname required parameter, which specifies the name of the filter to be removed. The DELETE_FILTER procedure will not remove filters that belong to completed captures; it only applies to filters of captures that have yet to start.

Starting a Workload Capture

Before starting a workload capture, you must first complete the prerequisites for capturing a database workload, as described in ["Prerequisites for Capturing a Database Workload"](#) on page 9-1. You should also review the workload capture options, as described in ["Workload Capture Options"](#) on page 9-2.

It is important to have a well-defined starting point for the workload so that the replay system can be restored to that point before initiating a replay of the captured workload. To have a well-defined starting point for the workload capture, it is preferable not to have any active user sessions when starting a workload capture. If active sessions perform ongoing transactions, those transactions will not be replayed properly in subsequent database replays, since only that part of the transaction whose calls were executed after the workload capture is started will be replayed. To avoid this problem, consider restarting the database in restricted mode using `STARTUP RESTRICT` before starting the workload capture. Once the workload capture begins, the database will automatically switch to unrestricted mode and normal operations can continue while the workload is being captured. For more information about restarting the database before capturing a workload, see ["Restarting the Database"](#) on page 9-2.

To start a workload capture:

- Use the `START_CAPTURE` procedure:

```
BEGIN
  DBMS_WORKLOAD_CAPTURE.START_CAPTURE (name => 'dec10_peak',
                                       dir => 'dec10',
                                       duration => 600,
                                       capture_sts => TRUE,
                                       sts_cap_interval => 300);
END;
/
```

In this example, a workload named `dec10_peak` will be captured for 600 seconds and stored in the operating system defined by the database directory object named `dec10`. A SQL tuning set will also be captured in parallel with the workload capture.

The `START_CAPTURE` procedure in this example uses the following parameters:

- The `name` required parameter specifies the name of the workload that will be captured.
- The `dir` required parameter specifies a directory object pointing to the directory where the captured workload will be stored.
- The `duration` parameter specifies the number of seconds before the workload capture will end. If a value is not specified, the workload capture will continue until the `FINISH_CAPTURE` procedure is called.
- The `capture_sts` parameter specifies whether to capture a SQL tuning set in parallel with the workload capture. If this parameter is set to `TRUE`, you can capture a SQL tuning set during workload capture, then capture another SQL tuning set during workload replay, and use SQL Performance Analyzer to compare the SQL tuning sets without having to re-execute the SQL statements. This enables you to obtain a SQL Performance Analyzer report and compare the SQL performance—before and after the change—while running Database Replay. You can also export the resulting SQL tuning set with its AWR data using the `EXPORT_AWR` procedure, as described in ["Exporting AWR Data for Workload Capture"](#) on page 9-19.

This feature is not supported for Oracle RAC. Workload capture filters that are defined using `DBMS_WORKLOAD_CAPTURE` do not apply to the SQL tuning set capture. The default value for this parameter is `FALSE`.

- The `sts_cap_interval` parameter specifies the duration of the SQL tuning set capture from the cursor cache in seconds. The default value is 300. Setting the

value of this parameter below the default value may cause additional overhead with some workloads and is not recommended.

Stopping a Workload Capture

This section describes how to stop a workload capture.

To stop a workload capture:

- Use the `FINISH_CAPTURE` procedure:

```
BEGIN
    DBMS_WORKLOAD_CAPTURE.FINISH_CAPTURE ();
END;
/
```

In this example, the `FINISH_CAPTURE` procedure finalizes the workload capture and returns the database to a normal state.

Tip: After capturing a workload on the production system, you need to preprocess the captured workload, as described in [Chapter 10](#), "Preprocessing a Database Workload".

Exporting AWR Data for Workload Capture

Exporting AWR data enables detailed analysis of the workload. This data is also required if you plan to run the Replay Compare Period report or the AWR Compare Period report on a pair of workload captures or replays.

To export AWR data:

- Use the `EXPORT_AWR` procedure:

```
BEGIN
    DBMS_WORKLOAD_CAPTURE.EXPORT_AWR (capture_id => 2);
END;
/
```

In this example, the AWR snapshots that correspond to the workload capture with a capture ID of 2 are exported, along with any SQL tuning set that may have been captured during workload capture.

The `EXPORT_AWR` procedure uses the `capture_id` required parameter, which specifies the ID of the capture whose AWR snapshots will be exported. The value of the `capture_id` parameter is displayed in the `ID` column of the `DBA_WORKLOAD_CAPTURES` view.

Note: This procedure works only if the corresponding workload capture was performed in the current database and the AWR snapshots that correspond to the original capture time period are still available.

See Also: *Oracle Database Reference* for information about the `DBA_WORKLOAD_CAPTURES` view

Importing AWR Data for Workload Capture

After AWR data is exported from the capture system, you can import the AWR data into another system, such as a test system where the captured workload will be

replayed. Importing AWR data enables detailed analysis of the workload. This data is also required if you plan to run the Replay Compare Period report or the AWR Compare Period report on a pair of workload captures or replays.

To import AWR data:

- Use the `IMPORT_AWR` function, as shown in the following example:

```
CREATE USER capture_awr
SELECT DBMS_WORKLOAD_CAPTURE.IMPORT_AWR (capture_id => 2,
                                         staging_schema => 'capture_awr')
FROM DUAL;
```

In this example, the AWR snapshots that correspond to the workload capture with a capture ID of 2 are imported using a staging schema named `capture_awr`.

The `IMPORT_AWR` procedure in this example uses the following parameters:

- The `capture_id` required parameter specifies the ID of the capture whose AWR snapshots will be import. The value of the `capture_id` parameter is displayed in the `ID` column of the `DBA_WORKLOAD_CAPTURES` view.
- The `staging_schema` required parameter specifies the name of a valid schema in the current database which can be used as a staging area while importing the AWR snapshots from the capture directory to the `SYS AWR` schema.

Note: This function fails if the schema specified by the `staging_schema` parameter contains any tables with the same name as any of the AWR tables.

See Also: *Oracle Database Reference* for information about the `DBA_WORKLOAD_CAPTURES` view

Monitoring Workload Capture Using Views

This section summarizes the views that you can display to monitor workload capture. You can also use Oracle Enterprise Manager to monitor workload capture, as described in "[Monitoring Workload Capture Using Enterprise Manager](#)" on page 9-14.

To access these views, you need DBA privileges:

- The `DBA_WORKLOAD_CAPTURES` view lists all the workload captures that have been captured in the current database.
- The `DBA_WORKLOAD_FILTERS` view lists all workload filters used for workload captures defined in the current database.

See Also:

- *Oracle Database Reference* for information about the `DBA_WORKLOAD_CAPTURES` view
- *Oracle Database Reference* for information about the `DBA_WORKLOAD_FILTERS` view

Preprocessing a Database Workload

After a workload is captured and setup of the test system is complete, the captured data must be preprocessed. Preprocessing a captured workload creates all necessary metadata for replaying the workload. This must be done once for every captured workload before they can be replayed. After the captured workload is preprocessed, it can be replayed repeatedly on a replay system.

To preprocess a captured workload, you will first need to move all captured data files from the directory where they are stored on the capture system to a directory on the instance where the preprocessing will be performed. Preprocessing is resource intensive and should be performed on a system that is:

- Separate from the production system
- Running the same version of Oracle Database as the replay system

For Oracle Real Application Clusters (Oracle RAC), select one database instance of the replay system for the preprocessing. This instance must have access to the captured data files that require preprocessing, which can be stored on a local or shared file system. If the capture directory path on the capture system resolves to separate physical directories in each instance, you will need to merge them into a single capture directory where the preprocessing will be performed. All directories must have the same directory tree and all files contained in each of these directories must be moved into a directory that has the same relative path to the capture directory.

Typically, you will preprocess the captured workload on the replay system. If you plan to preprocess the captured workload on a system that is separate from the replay system, you will also need to move all preprocessed data files from the directory where they are stored on the preprocessing system to a directory on the replay system after preprocessing is complete.

This chapter contains the following sections:

- [Preparing a Single Database Workload Using Enterprise Manager](#)
- [Preprocessing a Database Workload Using APIs](#)

Tip: Before you can preprocess a captured workload, you must first capture the workload on the production system, as described in [Chapter 9, "Capturing a Database Workload"](#).

Preparing a Single Database Workload Using Enterprise Manager

Preparing for a single workload consists of the following tasks:

- Creating a database replay task
- Creating a replay from a replay task

- Preparing the test database
- Preprocessing the workload and deploying the replay clients

Note: Preparing the test database is only required if you have not done so already.

The following sections provide procedures for these tasks.

Creating a Database Replay Task

Before proceeding, the capture that you want to replay must have some captured user calls.

To create a database replay task:

1. From the Database Replay page, click the **Replay Tasks** tab, then click **Create** in the toolbar.

The Create Task page appears.

Database Replay

Create Task Submit Cancel

* Name:

Description:

Workloads

Select a capture to be replayed. You can select multiple captures to do a consolidated replay. By default, the replays for a consolidated replay will start at the same time.

Sel...	Name	Status	Database Name	Database Version	Workload Duration (hh:mm:ss)	Database Time (hh:mm:ss)	Workload Analyzer Report	User Calls	Concurrent Capture Name
<input checked="" type="checkbox"/>	user1_capture12	Completed	database	12.1.0.1.0	00:10:03	00:00:01	6d	740	
<input checked="" type="checkbox"/>	capture12_1	Completed	database	12.1.0.1.0	00:10:00	00:00:01	6d	757	conCaptures4
<input type="checkbox"/>	capture11_adc	Completed	repos_adc	11.2.0.2.0	00:09:41	00:00:04	6d	823	conCaptures4
<input type="checkbox"/>	capture12_adc	Completed	db_adc	12.1.0.1.0	00:10:03	00:00:17	6d	937	conCaptures4

2. Provide a **Name** for the task, select a capture to be replayed, then click **Submit**. For consolidated replays, select two or more captures.

The Database Replay page reappears, and displays your newly created replay task in the table under the Replay Task tab.

Database Replay Page Refreshed Oct 18, 2012 12:15:20 PM PDT

Confirmation
Replay Tasks 'NewEmpReplay' created successfully.

Database Replay enables you to effectively test system changes in test environments by replaying a full production workload on a test system to determine the overall impact of the change. Database Replay captures your production workload and maintains all its characteristics such as timing and concurrency.

Database Replay workload capture is performed at the database server level and therefore can be used to assess the impact of any change which might affect database performance such as parameter changes, patching, storage migrations and database upgrades.

[Show Overview](#)

Captured Workloads **Replay Tasks**

View Create... Edit... Delete Detach

Name	Owner	Replays	Consolidated Replay	Creation Date	Description
NewEmpReplay	SYSMAN	0	No	Oct 18, 2012 12:15:20 PM ...	
myTask_si	SYSMAN	2	No	Oct 17, 2012 4:20:52 PM G...	
myTask	SYSMAN	3	Yes	Oct 17, 2012 10:06:02 AM ...	

Creating a Replay from a Replay Task

To create the replay:

1. From the Database Replay page, click the **Replay Tasks** tab.
2. Click the link for the desired replay task in the table.

The Replay Task page for the capture appears.

Database Replay Page Refreshed Oct 23, 2012 1:18:39 PM PDT

Database Replay > Replay Task: NewEmpReplay

Replay Task: NewEmpReplay

Replay Task Summary

Name: NewEmpReplay
Description:
Owner: SYSMAN

Captured Workloads

Name	Database Name	Database Version	Database Time (hh:mm:ss)	Workload Duration (hh:mm:ss)	Workload Analyzer Report	Storage Host	Storage Location
capture01	database	11.1.0.7.0	00:00:03	00:10:00		slc01gbc.us.oracle.com	/scratch/viponnus/view_sto...

Replays

View Create... Delete Compare Detach

Name	Owner	Status	Database Name	Database Version	Creation Date	Description
No rows found						

3. Click **Create** in the Replays section.

The Create Replay pop-up appears.

4. Provide a required Name and optional description, then click the **Target Database** icon.

The Search and Select: Targets pop-up appears.

5. Choose the desired database, then click **Select**.
6. Click **OK** in the Create Replay pop-up.

The Database Replay page for your replay appears, which includes a Task List with links to perform the needed tasks.

Database Replay Page Refreshed Oct 22, 2012 4:35:07 PM PDT

Database Replay > Replay Task: NewReplay > Replay: NewEmpReplay

Replay: NewEmpReplay

Home

Replay Target

Target Database: Database Version: 12.1.0.0.2 Replay Host: adc2101027.us.oracle.com

Task List

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

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

Workloads

Name	Database Name	Database Version	Database Time (hh:mm:ss)	Workload Duration (hh:mm:ss)	Workload Analyzer Report	Storage Host	Storage Location
capture11	Oemrep_Database	11.2.0....	00:00:09	00:09:37		adc2101027.us.oracle.com	/scratch/tkan/db1...

You can now proceed to the first task in the Task List, described in the next section.

Preparing the Test Database

Preparing the test database consists of:

- Setting up the test database
- Isolating the test database

Note: These tasks are optional. If you have already set up your test database, skip to "[Preprocessing the Workload and Deploying the Replay Clients](#)".

The following procedures explain how to perform each of these tasks, which you can do in any order.

To set up the test database:

1. From the Replay page for your particular replay, click the link for the **Set Up Test Database** task.

The Set Up Test Database page appears.

Database Replay > Task: EmpReplay > Replay: NewEmpReplay > Set Up Test Database

Set Up Test Database

Page Refreshed Sep 18, 2012 5:10:48 PM PDT Refresh OK

View Data Real Time: Manual Refresh

Select a capture to be replayed on the test database. Select a database target type and enter a database target name. Click Go to see a list of captures for a database.

* Target Type Database Instance

* Target Name cdb Go

* Capture myCDBCapture (Sep 17, 2012 7:28:01 PM PDT)

Capture Summary

Database Name	TGC00	Start Time	Sep 17, 2012 7:28:01 PM PDT
Capture Database Version	12.1.0.0.2	Start SCN	1460958
Cluster Database	No		
DBID	669376857		

Task List

Database Upgrade ☒ No ☐ Yes

Cluster Database ☒ No ☐ Yes

Reset Status Enable All Tasks

Task	Task Name	Description	Start Time	Status	Go to Task
1	Clone Existing Database Software	Clone the existing database software to a test system. This task accesses the Database Provisioning wizard, where you can configure the steps to clone the existing database software.			
2	Create Test Database	Create a test database ready for replay. This task accesses the Clone Database page, where you can create a test database from the existing database.			

TIP The status icons in the table represent the last execution status of a task. See the Icon Key.

TIP Return to the Database Replay home page with the OK button after completing all selected tasks.

Refresh OK

- Choose whether you want to upgrade the database or not, and indicate whether this is a cluster database.
- Click the **Go to Task** icon for the Clone Existing Database Software sub-task, or click **Enable All Tasks** if you want to create the test database first.
- Follow the instructions provided in the online help for the wizards.

When the tasks are complete, a checkmark appears in the Status column for each task.

- Click **OK** to return to the Database Replay page.

To isolate the test database:

- From the Replay page for your particular replay, click the link for the **Isolate Test Database** task.

A page appears explaining that references to external systems can cause problems during the replay.

database

Oracle Database > Performance > Availability > Schema > Administration

Database Replay > Task: MyReplayTask > Replay: MyReplay > Isolate Test Database: References to External Systems

Logged in as SYSTEM

Warning

A captured workload can contain references to external systems that may only be meaningful in the capture environment. Replying a workload with unresolved references to external systems may cause unexpected problems in the production environment.

You should perform a replay in a COMPLETELY ISOLATED test environment that may include hosts, networks, e-mail servers, storage systems, and other devices. Make sure to resolve all references to external systems in the replay environment, so that replaying a workload cannot harm your production environment.

Isolate Test Database: References to External Systems

References to external systems may cause problems during the replay.

Use the links below to verify potential references to external systems and modify those that are invalid.

- Database Links - This link takes you outside of the Database Replay process.
- Directory Objects - This link takes you outside of the Database Replay process.
- Streams - This link takes you outside of the Database Replay process.

TIP There may be more references to external systems than those found via the above links.

OK

- Use the links provided to verify potential references to external systems, modify those that are invalid, then click **OK**.

The Replay Summary page reappears.

Preprocessing the Workload and Deploying the Replay Clients

The final preparation for the replay consists of:

- Preprocessing the workload

You need to preprocess each captured workload once for each version of the database against which the workload will be replayed. After you preprocess the workload once, you can use it for any subsequent replay tasks and replays without needing to preprocess again, as long as the test database is the same version as the database where the workload was preprocessed.

For instance, for a replay task that contains two replays named "MyReplay1" and "MyReplay2," after you have preprocessed "MyReplay1", you can just directly reuse the directory object to replay "MyReplay2."

The Workload Analyzer report is available after preprocessing.

- Deploying the replay clients

You do not need to deploy the replay clients to other replay client hosts if these hosts can access the Oracle home of the test database you specified in the Database Target Name field.

The following procedures explain how to accomplish each of these tasks.

To preprocess the workload:

1. From the Replay page for your particular replay, click the link for the **Preprocess Workload** task.

The Preprocess Captured Workload: Locate Workload page appears.

2. Select the desired workload location option, then click **Next**.

Note: You initially need to select the copy option.

The Preprocess Captured Workload: Copy Workload page appears.

3. Provide the required credentials and the new location to which the workloads will be copied and preprocessed, then click **Next**.

- For a consolidated replay, there are multiple source workloads, so multiple source credentials might be needed for the current location of the workload directory. For more information on consolidated replays, see ["Using Consolidated Database Replay with Enterprise Manager"](#) on page 7.

The system responds by displaying a progress bar graph during processing, then displays the Preprocess Captured Workload: Select Directory page after the copy operation concludes.

4. Specify the Directory Object, or create a new Directory Object that points to the location that contains the workload. If you chose to copy from the workload location to a new location in the previous step, make sure that the directory object points to the exact location you specified in the New Location of the Workload Directory section.

The system responds by displaying a Capture Summary. You can now expand the Capture Details section to see the workload profile and workload filters. The Capture Summary does not appear for consolidated replays.

Click **Next** to display the Preprocess Captured Workload: Schedule page.

5. Provide input to schedule the preprocess job:
 - a. Provide your own required job name or accept the system-supplied name. The job system automatically names the job in uppercase.
 - b. Indicate whether you want the job to run as soon as you submit it, or whether you want it to run at a later time.
 - c. Provide the host credentials, which are used to run the preprocess job in the operating system.

Click **Next** to display the Preprocess Captured Workload: Review page.

6. Check to make sure that the settings are what you intend, then click **Submit**.

The Database Replay page appears, and assuming that there were no errors in your input, a confirmation message at the top of the page states "Job JOBNAME to prepare the workload has been created successfully."

7. Click the *JOBNAME* link to check the status of the job. The job must succeed before you can proceed to the Replay Workload task.

Note: A message may appear in the Task List stating that you need to install an additional PL/SQL package in the test database to generate a compare period report after the trial. Click **Install PL/SQL Package** to resolve this issue before proceeding to the Replay Workload task.

Tip: After preprocessing a captured workload, you can replay it on the test system, as described in [Chapter 11, "Replaying a Database Workload"](#).

To deploy the replay clients:

1. From the Replay page for your particular replay, click the link for the **Deploy Replay Clients** task.

The Deploy Replay Clients page appears.

Database Replay > Task: EmpReplay > Replay: NewEmpReplay > Deploy Replay Clients

Deploy Replay Clients

Refresh Cancel Continue

Workload Capture
Select a capture to be replayed on the test database. Select a database target type and enter a database target name. Click Go to see a list of captures for a database.

Target Type: Database Instance
Target Name: cdb Go
Capture: myCDBCapture (Sep 17, 2012 7:28:01 PM PDT)
Replay Database Version: 12.1.0.0.2
Total Number of CPU Cores Needed: 1

TIP It is recommended that you use a total of at least 1 CPU core(s) to run the Replay Clients for the selected capture. Deploy the Replay Client to one or more hosts depending on the number of CPU cores available on each host.

Database Name: TGC00	Start Time: Sep 17, 2012 7:28:01 PM PDT
Capture Database Version: 12.1.0.0.2	Start SCN: 1460958
Cluster Database: No	
DBID: 669376857	

2. Accept the default values defined for the associated workload capture, or override these values, then click **Continue**.

The Provision Oracle Database Client wizard appears.

3. Follow the instructions provided in the online help for each step of the wizard.

After you click Submit in the Review step to run the deployment procedure according to the schedule you have set, the Replay Summary page reappears.

Preprocessing a Database Workload Using APIs

This section describes how to preprocess a captured workload using the DBMS_WORKLOAD_REPLAY package. You can also use Oracle Enterprise Manager to preprocess a captured workload, as described in ["Preparing a Single Database Workload Using Enterprise Manager"](#) on page 10-1.

To preprocess a captured workload:

- Use the PROCESS_CAPTURE procedure:

```
BEGIN
  DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE (capture_dir => 'dec06');
END;
/
```

In this example, the captured workload stored in the dec06 directory will be preprocessed.

The PROCESS_CAPTURE procedure in this example uses the capture_dir required parameter, which specifies the directory that contains the captured workload to be preprocessed.

Tip: After preprocessing a captured workload, you can replay it on the test system, as described in [Chapter 11, "Replaying a Database Workload"](#).

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about the DBMS_WORKLOAD_REPLAY package

Running the Workload Analyzer Command-Line Interface

The Workload Analyzer is a Java program that analyzes a workload capture directory and identifies parts of a captured workload that may not replay accurately due to insufficient data, errors that occurred during workload capture, or usage features that are not supported by Database Replay. The results of the workload analysis are saved to an HTML report named wcr_cap_analysis.html located in the capture directory that is being analyzed. If an error can be prevented, the workload analysis report displays available preventive actions that can be implemented before replay. If an error cannot be corrected, the workload analysis report provides a description of the error so it can be accounted for during replay. Running Workload Analyzer is the default option and is strongly recommended.

Note: If you are preprocessing a workload capture using Oracle Enterprise Manager, then you do not need to run Workload Analyzer in the command-line interface. Oracle Enterprise Manager enables you to run Workload Analyzer as part of the workload preprocessing.

Workload Analyzer is composed of two JAR files, `dbr analyzer.jar` and `dbr parser.jar`, located in the `$ORACLE_HOME/rdbms/jlib/` directory of a system running Oracle Database Enterprise Edition Release 11.2.0.2 or higher. Workload Analyzer requires Java 1.5 or higher and the `ojdbc6.jar` file located in the `$ORACLE_HOME/jdbc/lib/` directory.

To run Workload Analyzer:

1. In the command-line interface, run the following java command on a single line:

```
java -classpath
$ORACLE_HOME/jdbc/lib/ojdbc6.jar:$ORACLE_HOME/rdbms/jlib/dbrparser.jar:
$ORACLE_HOME/rdbms/jlib/dbr analyzer.jar:
oracle.dbreplay.workload.checker.CaptureChecker
<capture_directory> <connection_string>
```

For the `capture_directory` parameter, input the operating system path of the capture directory. This directory should also contain the exported AWR data for the workload capture. For the `connection_string` parameter, input the connection string of an Oracle database that is release 11.1 or higher.

An example of this command may be:

```
java -classpath
$ORACLE_HOME/jdbc/lib/ojdbc6.jar:$ORACLE_HOME/rdbms/jlib/dbrparser.jar:
$ORACLE_HOME/rdbms/jlib/dbr analyzer.jar:
oracle.dbreplay.workload.checker.CaptureChecker /scratch/capture
jdbc:oracle:thin:@myhost.mycompany.com:1521:orcl
```

2. When prompted, input the username and password of a database user with EXECUTE privileges for the `DBMS_WORKLOAD_CAPTURE` package and the `SELECT_CATALOG` role on the target database.

Replaying a Database Workload

After a captured workload is preprocessed, it can be replayed repeatedly on a replay system that is running the same version of Oracle Database.

This chapter describes how to replay a database workload on the test system and contains the following sections:

- [Setting Up the Test System](#)
- [Steps for Replaying a Database Workload](#)
- [Replaying a Database Workload Using Enterprise Manager](#)
- [Monitoring Workload Replay Using Enterprise Manager](#)
- [Replaying a Database Workload Using APIs](#)
- [Monitoring Workload Replay Using APIs](#)

Tip: Before you can replay a database workload, you must first:

- Capture the workload on the production system, as described in [Chapter 9, "Capturing a Database Workload"](#)
- Preprocess the captured workload, as described in [Chapter 10, "Preprocessing a Database Workload"](#)

Setting Up the Test System

Typically, the replay system where the preprocessed workload will be replayed should be a test system that is separate from the production system. Before a test system can be used for replay, it must be prepared properly, as described in the following sections:

- [Restoring the Database](#)
- [Resetting the System Time](#)

Restoring the Database

Before a workload can be replayed, the application data state should be logically equivalent to that of the capture system at the start time of workload capture. This minimizes replay divergence during replay. The method for restoring the database depends on the backup method that was used before capturing the workload. For example, if RMAN was used to back up the capture system, you can use RMAN `DUPLICATE` capabilities to create the test database. For more information, see ["Prerequisites for Capturing a Database Workload"](#) on page 9-1.

After the database is created with the appropriate application data on the test system, perform the system change you want to test, such as a database or operating system upgrade. The primary purpose of Database Replay is to test the effect of system changes on a captured workload. Therefore, the system changes you make should define the test you are conducting with the captured workload.

Resetting the System Time

It is recommended that the system time on the replay system host be changed to a value that approximately matches the capture start time just before replay is started. Otherwise, an invalid data set may result when replaying time-sensitive workloads. For example, a captured workload that contains SQL statements using the `SYSDATE` and `SYSTIMESTAMP` functions may cause replay divergence when replayed on a system that has a different system time. Resetting the system time will also minimize job scheduling inconsistencies between capture and replay.

Note: When resetting the system time, ensure that future Automatic Workload Repository (AWR) snapshots can still be created automatically as expected after the system time is reset.

Steps for Replaying a Database Workload

Proper planning of the workload replay ensures that the replay will be accurate. Replaying a database workload requires the following steps:

- [Setting Up the Replay Directory](#)
- [Resolving References to External Systems](#)
- [Connection Remapping](#)
- [Specifying Replay Options](#)
- [Using Filters with Workload Replay](#)
- [Setting Up Replay Clients](#)

Setting Up the Replay Directory

The captured workload must have been preprocessed and copied to the replay system. A directory object for the directory to which the preprocessed workload is copied must exist in the replay system.

Resolving References to External Systems

A captured workload may contain references to external systems, such as database links or external tables. Typically, you should reconfigure these external interactions to avoid impacting other production systems during replay. External references that need to be resolved before replaying a workload include:

- Database links
It is typically not desirable for the replay system to interact with other databases. Therefore, you should reconfigure all database links to point to an appropriate database that contains the data needed for replay.
- External tables

All external files specified using directory objects referenced by external tables need to be available to the database during replay. The content of these files should be the same as during capture, and the filenames and directory objects used to define the external tables should also be valid.

- **Directory objects**

You should reconfigure any references to directories on the production system by appropriately redefining the directory objects present in the replay system after restoring the database.

- **URLs**

URLs/URIs that are stored in the database need to be configured so that Web services accessed during the workload capture will point to the proper URLs during replay. If the workload refers to URLs that are stored in the production system, you should isolate the test system network during replay.

- **E-mails**

To avoid resending E-mail notifications during replay, any E-mail server accessible to the replay system should be configured to ignore requests for outgoing E-mails.

Tip: To avoid impacting other production systems during replay, Oracle strongly recommends running the replay within an isolated private network that does not have access to the production environment hosts.

Connection Remapping

During workload capture, connection strings used to connect to the production system are captured. In order for the replay to succeed, you need to remap these connection strings to the replay system. The replay clients can then connect to the replay system using the remapped connections.

For Oracle Real Application Clusters (Oracle RAC) databases, you can map all connection strings to a load balancing connection string. This is especially useful if the number of nodes on the replay system is different from the capture system. Alternatively, if you want to direct workload to specific instances, you can use services or explicitly specify the instance identifier in the remapped connection strings.

User Remapping

During workload capture, the username of the database user or schema used to connect to the production system is captured. You can choose to remap the captured username to that of a new user or schema.

Specifying Replay Options

After the database is restored, and connections and users are remapped, you can set the following replay options as appropriate:

- [Preserving COMMIT Order](#)
- [Controlling Session Connection Rate](#)
- [Controlling Request Rate Within a Session](#)

Preserving COMMIT Order

The `synchronization` parameter controls whether the `COMMIT` order in the captured workload will be preserved during replay.

If this parameter is set to `SCN`, the `COMMIT` order in the captured workload will be preserved during replay and all replay actions will be executed only after all dependent `COMMIT` actions have completed.

If this parameter is set to `OBJECT_ID`, all replay actions will be executed only after all relevant `COMMIT` actions have completed. Relevant `COMMIT` actions must meet the following criteria:

- Issued before the given action in the workload capture
- Modified at least one of the database objects for which the given action is referencing, either implicitly or explicitly

Setting this parameter to `OBJECT_ID` allows for more concurrency during workload replays for `COMMIT` actions that do not reference the same database objects during workload capture.

You can disable this option by setting the parameter to `OFF`, but the replay will likely yield significant replay divergence. However, this may be desirable if the workload consists primarily of independent transactions, and divergence during unsynchronized replay is acceptable.

Controlling Session Connection Rate

The `connect_time_scale` parameter enables you to scale the elapsed time between the time when the workload capture began and each session connects. You can use this option to manipulate the session connect time during replay with a given percentage value. The default value is 100, which will attempt to connect all sessions as captured. Setting this parameter to 0 will attempt to connect all sessions immediately.

Controlling Request Rate Within a Session

User think time is the elapsed time while the replayed user waits between issuing calls within a single session. To control replay speed, use the `think_time_scale` parameter to scale user think time during replay.

If user calls are being executed slower during replay than during capture, you can make the database replay attempt to catch up by setting the `think_time_auto_correct` parameter to `TRUE`. This will make the replay client shorten the think time between calls, so that the overall elapsed time of the replay will more closely match the captured elapsed time.

If user calls are being executed faster during replay than during capture, setting the `think_time_auto_correct` parameter to `TRUE` will not change the think time. The replay client will not increase the think time between calls to match the captured elapsed time.

Using Filters with Workload Replay

By default, all captured database calls are replayed during workload replay. You can use workload filters to specify which database calls to include in or exclude from the workload during workload replay.

Workload replay filters are first defined and then added to a replay filter set so they can be used in a workload replay. There are two types of workload filters: inclusion filters and exclusion filters. Inclusion filters enable you to specify database calls that

will be replayed. Exclusion filters enable you to specify database calls that will not be replayed. You can use either inclusion filters or exclusion filters in a workload replay, but not both. The workload filter is determined as an inclusion or exclusion filter when the replay filter set is created.

Setting Up Replay Clients

The replay client is a multithreaded program (an executable named `wrc` located in the `$ORACLE_HOME/bin` directory) where each thread submits a workload from a captured session. Before replay begins, the database will wait for replay clients to connect. At this point, you need to set up and start the replay clients, which will connect to the replay system and send requests based on what has been captured in the workload.

Before starting replay clients, ensure that the:

- Replay client software is installed on the hosts where it will run
- Replay clients have access to the replay directory
- Replay directory contains the preprocessed workload capture
- Replay user has the correct user ID, password, and privileges (the replay user needs the DBA role and cannot be the `SYS` user)
- Replay clients are not started on a system that is running the database
- Replay clients read the capture directory on a file system that is different from the one on which the database files reside

To do this, copy the capture directory to the system where the replay client will run. After the replay is completed, you can delete the capture directory.

After these prerequisites are met, you can proceed to set up and start the replay clients using the `wrc` executable. The `wrc` executable uses the following syntax:

```
wrc [user/password[@server]] MODE=[value] [keyword=[value]]
```

The parameters `user`, `password` and `server` specify the username, password and connection string used to connect to the replay database. The parameter `mode` specifies the mode in which to run the `wrc` executable. Possible values include `replay` (the default), `calibrate`, and `list_hosts`. The parameter `keyword` specifies the options to use for the execution and is dependent on the mode selected. To display the possible keywords and their corresponding values, run the `wrc` executable without any arguments.

The following sections describe the modes that you can select when running the `wrc` executable:

- [Calibrating Replay Clients](#)
- [Starting Replay Clients](#)
- [Displaying Host Information](#)

Calibrating Replay Clients

Since one replay client can initiate multiple sessions with the database, it is not necessary to start a replay client for each session that was captured. The number of replay clients that need to be started depends on the number of workload streams, the number of hosts, and the number of replay clients for each host.

To estimate the number of replay clients and hosts that are required to replay a particular workload, run the `wrc` executable in `calibrate` mode.

In calibrate mode, the `wrc` executable accepts the following keywords:

- `replaydir` specifies the directory that contains the preprocessed workload capture you want to replay. If unspecified, it defaults to the current directory.
- `process_per_cpu` specifies the maximum number of client processes that can run per CPU. The default value is 4.
- `threads_per_process` specifies the maximum number of threads that can run within a client process. The default value is 50.

The following example shows how to run the `wrc` executable in calibrate mode:

```
%> wrc mode=calibrate replaydir=./replay
```

In this example, the `wrc` executable is executed to estimate the number of replay clients and hosts that are required to replay the workload capture stored in a subdirectory named `replay` under the current directory. In the following sample output, the recommendation is to use at least 21 replay clients divided among 6 CPUs:

```
Workload Replay Client: Release 12.1.0.0.1 - Production on Fri Sept 30
13:06:33 2011
```

```
Copyright (c) 1982, 2011, Oracle. All rights reserved.
```

```
Report for Workload in: /oracle/replay/
-----
```

```
Recommendation:
```

```
Consider using at least 21 clients divided among 6 CPU(s).
```

```
Workload Characteristics:
```

- max concurrency: 1004 sessions
- total number of sessions: 1013

```
Assumptions:
```

- 1 client process per 50 concurrent sessions
- 4 client process per CPU
- think time scale = 100
- connect time scale = 100
- synchronization = TRUE

Starting Replay Clients

After determining the number of replay clients that are needed to replay the workload, you need to start the replay clients by running the `wrc` executable in replay mode on the hosts where they are installed. Once started, each replay client will initiate one or more sessions with the database to drive the workload replay.

In replay mode, the `wrc` executable accepts the following keywords:

- `userid` and `password` specify the user ID and password of a replay user for the replay client. If unspecified, these values default to the `system` user.
- `server` specifies the connection string that is used to connect to the replay system. If unspecified, the value defaults to an empty string.
- `replaydir` specifies the directory that contains the preprocessed workload capture you want to replay. If unspecified, it defaults to the current directory.
- `workdir` specifies the directory where the client logs will be written. This parameter is only used with the `debug` parameter for debugging purposes.

- `debug` specifies whether debug data will be created. Possible values include:
 - `on`
Debug data will be written to files in the working directory
 - `off`
No debug data will be written (the default value)

Note: Before running the `wrc` executable in debug mode, contact Oracle Support for more information.

- `connection_override` specifies whether to override the connection mappings stored in the `DBA_WORKLOAD_CONNECTION_MAP` view. If set to `TRUE`, connection remappings stored in the `DBA_WORKLOAD_CONNECTION_MAP` view will be ignored and the connection string specified using the `server` parameter will be used. If set to `FALSE`, all replay threads will connect using the connection remappings stored in the `DBA_WORKLOAD_CONNECTION_MAP` view. This is the default setting.

The following example shows how to run the `wrc` executable in replay mode:

```
%> wrc system/password@test mode=replay replaydir=./replay
```

In this example, the `wrc` executable starts the replay client to replay the workload capture stored in a subdirectory named `replay` under the current directory.

After all replay clients have connected, the database will automatically distribute workload capture streams among all available replay clients and workload replay can begin. You can monitor the status of the replay clients using the `V$WORKLOAD_REPLAY_THREAD` view. After the replay finishes, all replay clients will disconnect automatically.

Displaying Host Information

You can display the hosts that participated in a workload capture and workload replay by running the `wrc` executable in `list_hosts` mode.

In `list_hosts` mode, the `wrc` executable accepts the keyword `replaydir`, which specifies the directory that contains the preprocessed workload capture you want to replay. If unspecified, it defaults to the current directory.

The following example shows how to run the `wrc` executable in `list_hosts` mode:

```
%> wrc mode=list_hosts replaydir=./replay
```

In this example, the `wrc` executable is executed to list all hosts that participated in capturing or replaying the workload capture stored in a subdirectory named `replay` under the current directory. In the following sample output, the hosts that participated in the workload capture and three subsequent replays are shown:

```
Workload Replay Client: Release 12.1.0.0.1 - Production on Fri Sept 30
13:44:48 2011
```

```
Copyright (c) 1982, 2011, Oracle. All rights reserved.
```

```
Hosts found:
```

```
Capture:
```

```
    prod1
```

```
    prod2
```

```
Replay 1:
```

```
    test1
```

```

Replay 2:
    test1
    test2
Replay 3:
    testwin

```

Replaying a Database Workload Using Enterprise Manager

This section describes how to replay a database workload using Enterprise Manager.

The primary tool for replaying database workloads is Oracle Enterprise Manager. If Oracle Enterprise Manager is unavailable, you can also replay database workloads using APIs, as described in ["Replaying a Database Workload Using APIs"](#) on page 11-17.

Note: Before proceeding, you must already have created a replay task and created a replay from the replay task. To do this, see ["Preparing a Single Database Workload Using Enterprise Manager"](#) on page 10-1.

To replay a database workload using Enterprise Manager:

1. From the Enterprise menu of the Enterprise Manager Cloud Control console, select **Quality Management**, then **Database Replay**.

If the Database Login page appears, log in as a user with administrator privileges.

The Database Replay page appears.

Database Replay Page Refreshed Oct 18, 2012 12:31:20 PM PDT

Database Replay enables you to effectively test system changes in test environments by replaying a full production workload on a test system to determine the overall impact of the change. Database Replay captures your production workload and maintains all its characteristics such as timing and concurrency.

Database Replay workload capture is performed at the database server level and therefore can be used to assess the impact of any change which might affect database performance such as parameter changes, patching, storage migrations and database upgrades.

▼ Hide Overview

The diagram illustrates the Database Replay process:

- Production Environment:** Clients connect to the Database.
- Captured Workload:** The workload is captured from the Production Environment.
- Replay Workload:** The captured workload is processed and then replayed.
- Test Environment:** The workload is replayed on the Test Environment Database using Launch Replay Clients.
- Analysis & Reporting:** The results of the replay are analyzed and reported.

Captured Workloads

Name	Owner	Replays	Consolidated Replay	Creation Date	Description
NewEmpReplay	SYSMAN	1 No		Oct 18, 2012 12:15:20 PM ...	
myTask_si	SYSMAN	2 No		Oct 17, 2012 4:20:52 PM G...	
myTask	SYSMAN	3 Yes		Oct 17, 2012 10:06:02 AM ...	

Replay Tasks

View Create... Edit... Delete Detach

2. Select the **Replay Tasks** tab, then click the link for the desired replay task in the table.

The Replay Task page for the replay appears.

Database Replay Page Refreshed Nov 8, 2012 6:27:38 PM PST

Database Replay > Replay Task: user1_task_cons_1

Replay Task: user1_task_cons_1

Replay Task Summary

Name: user1_task_cons_1
Description:
Owner: USER1

Captured Workloads

Name	Database Name	Database Version	Database Time (hh:mm:ss)	Workload Duration (hh:mm:ss)	Workload Analyzer Report	Storage Host	Storage Location
capture12_1	database	12.1.0.1.0	00:00:01	00:10:00		slc03sct.us.oracle.com	/scratch/tkan/db12/captures
capture12_adc	db_adc	12.1.0.1.0	00:00:17	00:10:03		slc03sct.us.oracle.com	/scratch/tkan/db12/captures

Replays

Create... Delete Compare

Name	Owner	Status	Database Name	Database Version	Creation Date	Description
user1_trial_cons_4	USER1	Completed	database	12.1.0.1.0	Nov 7, 2012 9:47:12 PM G...	
user1_trial_cons_3	USER1	Completed	database	12.1.0.1.0	Nov 7, 2012 8:46:11 PM G...	
user_trial_cons_2	USER1	Completed	database	12.1.0.1.0	Nov 7, 2012 6:22:09 PM G...	
user1_trial_cons_1	USER1	Completed	database	12.1.0.1.0	Nov 6, 2012 10:57:54 AM G...	

- Click **Create** in the Replays section to create the replay.

The Create Replay pop-up appears.

- Provide a required Name and optional description, then click the **Target Database** icon.

The Search and Select: Targets pop-up appears.

- Choose the appropriate database, then click **Select**.

- Click **OK** in the Create Replay pop-up.

The Database Replay page for your replay appears, which includes a Task List with a link to perform the replay.

Database Replay Page Refreshed Nov 8, 2012 6:34:04 PM PST

Database Replay > Replay Task: user1_task_cons_1 > Replay: NewEmpReplay

Replay: NewEmpReplay

Home

Replay Target

Target Database: Database Version: 12.1.0.1.0 Replay Host: slc03sct.us.oracle.com

Task List

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

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

Workloads

Name	Database Name	Database Version	Database Time (hh:mm:ss)	Workload Duration (hh:mm:ss)	Workload Analyzer Report	Storage Host	Storage Location
capture12_1	database	12.1.0....	00:00:01	00:10:00		slc03sct.us.oracle.com	/scratch/tkan/db1...
capture12_adc	db_adc	12.1.0....	00:00:17	00:10:03		slc03sct.us.oracle.com	/scratch/tkan/db1...

- Click the link for the **Replay Workload** task.

The Replay Workload: Locate Workload page appears.

The screenshot shows the 'Locate Workload' step of the replay setup wizard. At the top, a progress bar indicates the current step. Below it, an information box states: 'Replay should be performed on a test database. If the current database target is not the intended test database, click Cancel and select the test database target before continuing the replay setup.' The main section is titled 'Replay Workload: Locate Workload' and shows the current database as 'database', replay name as 'DocReplay1', and logged in as 'system'. A note states: 'The captured workload directories must be accessible from this database.' Two radio button options are presented: 'Copy the workload directories to this host from another host.' (selected) and 'Use an existing directory with multiple workload subdirectories on this host.' Navigation buttons 'Cancel', 'Step 1 of 8', and 'Next' are visible.

8. Select the desired workload location option.

If you have not previously copied the workload from its storage location to the replay location where the replay clients can access it, select the option to copy the workload. Otherwise, select the option to use the existing replay directory that contains the workload to be replayed.

Click **Next** to display the Replay Workload: Copy Workload page.

The screenshot shows the 'Copy Workload' step of the replay setup wizard. The progress bar is updated. The title is 'Replay Workload: Copy Workload'. It shows the same database and replay name information. A note says: 'Continue to the next step after the workloads are completely copied to the current host.' The 'Copy from Workload Location' section instructs the user to enter location details and credentials. The 'Current Location of the Workload Directory' is shown with Host 'slc01hxi.us.oracle.com' and Directory '/scratch/tkan/db12/captures_cdb'. Below this, credential details for 'tkan' are shown. The 'New Location of the Workload Directory' section has Host 'adc2101027.us.oracle.com' and Directory '/scratch/tkan/newemp'. Similar credential details for 'tkan' are shown. Navigation buttons 'Cancel', 'Back', 'Step 2 of 8', and 'Next' are visible.

9. Provide the required credentials and the new location of the workload directory to which you want to copy the workload, then click **Next**.

- There are multiple source workloads for a consolidated replay, so multiple source credentials might be needed for the current location of the workload directory. For more information on consolidated replays, see ["Using Consolidated Database Replay with Enterprise Manager"](#) on page 7.

The system responds by displaying a progress bar graph during processing, then displays the Replay Workload: Select Directory page after the copy operation concludes.

10. Specify the Directory Object, or create a new Directory Object that points to the location that contains the workload. If you chose to copy from the workload location to a new location in the previous step, make sure that the directory object points to the exact location you specified in the New Location of the Workload Directory section.

The system responds by displaying a Capture Summary. You can expand the Capture Details section to see the workload profile and workload filters. You can also generate a Workload Capture Analyzer Report and Database Capture Report. The Capture Summary does not appear for consolidated replays.

Click **Next** to display the Replay Workload: Initialize Options page.

11. In the SQL Performance Analyzer section, retain or disable the Capture SQL Statements option, which is enabled by default and recommended. You can disable this option if you do not want to compare SQL tuning sets at the end of the Replay.

- The SQL Performance Analyzer can initiate an impact analysis of environmental changes on the performance of SQL statements within a SQL Tuning Set. You can create and analyze SQL Performance Analyzer tasks to test the effects of a database upgrade, initialization parameter change, Exadata simulation, or custom experiments. A task compares the effects of before-trial changes with after-trial changes.

Although Database Replay provides an analysis of how a change affects your entire system, you can use a SQL tuning set in conjunction with the SQL Performance Analyzer to gain a more SQL-centric analysis of how the change affects SQL statements and execution plans.

By capturing a SQL tuning set during workload replay, you can use SQL Performance Analyzer to compare this SQL tuning set to another SQL tuning set captured during workload capture, without having to re-execute the SQL statements. This enables you to obtain a SQL Performance Analyzer report and compare the SQL performance, before and after change, while running Database Replay.

- In the Identify Source section, initial replay options refer to the connection mappings and parameters on the Customize Options page. Connections are captured along with the workload.

Note: This section does not appear for consolidated replays or Oracle RAC.

Click **Next** to display the Replay Workload: Customize Options page.

12. Remap captured connection strings to connection strings that point to the replay system. Note that you need to remap each capture connection. For example, in the illustration above, you would need to remap the connection for both capture12_1 and capture12_adc.

(You can remap connections per workload for consolidated replay. There is a Capture Name drop-down to choose the workload.)

Click the **Connection Mappings** tab. There are several methods you can use to remap captured connection strings. You can choose to:

- **Use a single connect descriptor for all client connections** by selecting this option and entering the connect descriptor you want to use. The connect descriptor should point to the replay system.
To test the connection, click **Test Connection**. If the connect descriptor is valid, an Information message is displayed to inform you that the connection was successful.
- **Use a single TNS net service name for all client connections** by selecting this option and entering the net service name you want to use. All replay clients must be able to resolve the net service name, which can be done using a local `tnsnames.ora` file.
- **Use a separate connect descriptor or net service name for each client connect descriptor captured in the workload** by selecting this option and, for each capture system value, entering a corresponding replay system value that the replay client will be use. If you selected the "Use replay options from a previous replay" option in the Initialize Options step, the "Use a separate

connect descriptor" option is selected, and the previous Replay system values appear in the text field below.

Note: This option does not apply to consolidated replays.

For more information, see ["Connection Remapping"](#) on page 11-3.

13. Specify the replay options using the replay parameters, which control some aspects of the Replay.

To modify the replay behavior, click the **Replay Parameters** tab and enter the desired values for each replay parameter. Using the default values is recommended. For information about setting the replay parameters, see ["Specifying Replay Options"](#) on page 11-3.

After setting the replay parameters, click **Next**.

The Replay Workload: Prepare Replay Clients page appears.

Previous Customize Options **Prepare Replay Clients** Wait for Client Connections Review

Replay Workload: Prepare Replay Clients

Database: cdb
Replay Name: DocReplay1
Logged In As: system

Cancel Back Step 6 of 8 Next

Specify the list of Replay Clients below that Enterprise Manager should start automatically. You can also start more Replay Clients manually in the next step. Refer to the Oracle Real Application Testing User's Guide for information on how to set up and start the Replay Clients.

Number of Replay Clients and CPU Cores
The number of Replay Clients needed to replay the workload depends on the number of captured database sessions. Click the Estimate button to find the estimated number of Replay Clients and CPU cores needed.

Total Number of Replay Clients Needed: 2 Estimate
Total Number of CPU Cores Needed: 2

Consider starting at least 2 Replay Client(s) divided among 2 CPU core(s).

Replay Client Hosts
If the Replay Client has been installed on one or more targets, Enterprise Manager can start the Replay Clients automatically. Specify the list of Replay Clients to start automatically when you continue to the next step. You must configure each Replay Client host before proceeding.

Last Updated: Sep 19, 2012 12:02:00 PM PDT Refresh

Select	Target	Number of Replay Clients	Configured	Status	Number of CPU Cores	Memory Size (MB)	CPU Utilization %	Memory Utilization %
	(No Replay Client hosts specified)							

Add Replay Client Hosts

14. Ensure that the replay clients are prepared for replay:

Before proceeding, the replay clients must be set up. For more information, see ["Setting Up Replay Clients"](#) on page 11-5.

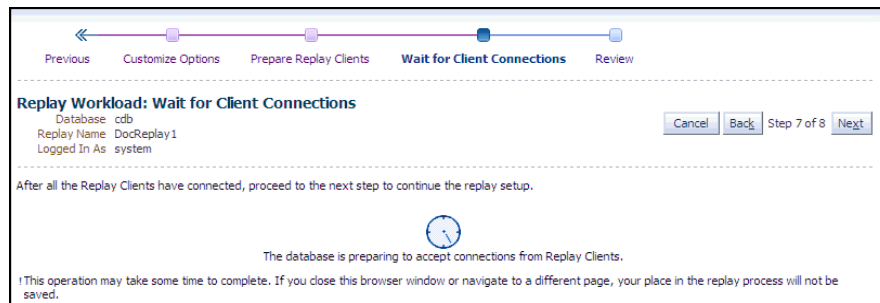
- a. Click **Estimate** to determine how many Replay clients and CPUs are required for the Replay.
- b. Click **Add Replay Client Hosts** to add a host or hosts for the Replay clients. (If you do not want to add any replay client hosts, you can still continue and start the replay clients from the command line outside of Enterprise Manager).

The Search and Select: Replay Client Host pop-up appears.

- Specify a Target Name and then click **Go**, or just click **Go** to display the entire list of available hosts.
 - Choose a host and then click **Select**.
- c. When the host name appears in the Target column of the Replay Client Hosts table, specify the number of Replay clients recommended from the estimate results, then make sure that the number of CPUs listed for the host satisfies

the minimum recommendation in the estimate results. You should start at least one replay client per captured workload.

- d. In the Configured column, click the **No** link to configure the Replay client host in the pop-up that appears.
 - e. Click **Apply** after you have finished providing input in the pop-up. The Configured column now displays Yes in the Replay Client Hosts table.
15. Click **Next** to start the replay clients and display the Replay Workload: Wait for Client Connections page.



For information about starting replay clients, see ["Setting Up Replay Clients"](#) on page 11-5.

Note: If you have reached this step in the process from the Enterprise menu, you also need to enter credentials for the replay job and for the replay result storage host.

- As replay clients are started, the replay client connections are displayed in the Client Connections table.
- The text below the clock changes if a replay client is connected.
- The Client Connections table is populated when at least one replay client is connected.

When all replay clients have connected, enter host credentials at the bottom of the page to start the replay job, then click **Next** to display the Replay Workload: Review page.

16. Review the options and parameters that have been defined for the workload replay.
 - The value for Connected Replay Clients must be at least 1 in order to successfully submit the Replay Workload job.
 - The Submit button is enabled only if at least one Replay client is connected.
 - Before starting the replay, reset the system clock to a value that is as close to the capture start time as possible. This minimizes any replay divergence that may result from replaying a time-sensitive workload. For more information, see ["Resetting the System Time"](#) on page 11-2.
17. If everything appears as you have intended, click **Submit** to submit the Replay job.

After the replay starts, the Home tab of the Database Replay page for this replay reappears with a system message that states "The workload replay has started."

For information about monitoring an active workload replay, see ["Monitoring an Active Workload Replay"](#) on page 11-15.

Monitoring Workload Replay Using Enterprise Manager

This section describes how to monitor workload replay using Enterprise Manager. The primary tool for monitoring workload replay is Oracle Enterprise Manager. Using Enterprise Manager, you can:

- Monitor or stop an active workload replay
- View a completed workload replay

If Oracle Enterprise Manager is unavailable, you can monitor workload replay using APIs and views, as described in ["Monitoring Workload Replay Using APIs"](#) on page 11-27.

This section contains the following topics:

- [Monitoring an Active Workload Replay](#)
- [Viewing a Completed Workload Replay](#)

Monitoring an Active Workload Replay

This section describes how to monitor an active workload replay using Enterprise Manager.

To monitor an active workload replay:

1. From the Database Replay page, click the **Replay Tasks** tab.
2. Click the name of the replay task that contains the replay for which you want to monitor replay progress.
3. From the Replays section of the Replay Task page, click the name of the replay you have submitted for processing in the Create Replay wizard. (The Status column for this replay should show In Progress.)

The Home tab of the Database Replay page appears, and the Replay Summary shows a Status of Running.

- The replay line in the Replay Progress chart updates dynamically while the Replay is in progress. You can update the chart by clicking the Refresh button.
- The user calls line in the Replay Progress chart indicates how much of the workload the database has serviced relative to the capture at the same time.
- Data for the Replay Divergence Summary is not available until the Replay completes.

Viewing a Completed Workload Replay

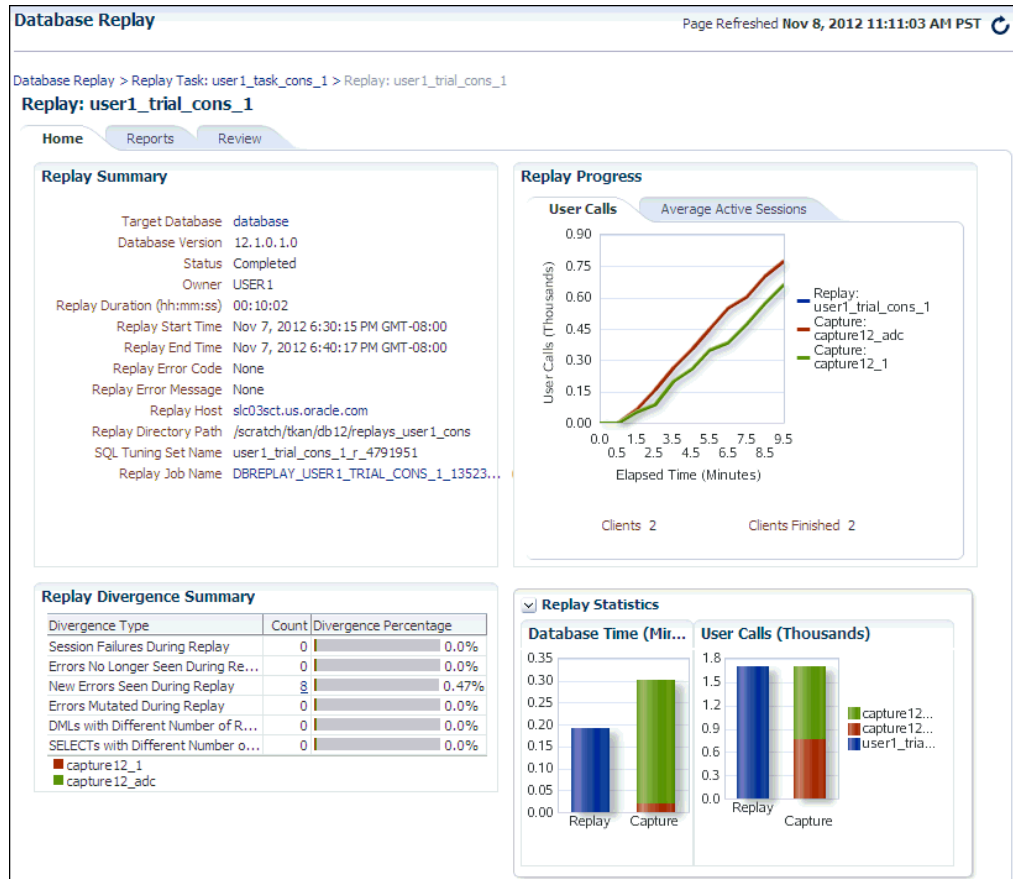
This section describes how to view a completed workload replay using Enterprise Manager.

To view a completed workload replay:

1. From the Database Replay page, click the **Replay Tasks** tab.
2. Click the name of the replay task that contains the completed replay you want to view.

- From the Replays section of the Replay Task page, click the name of the replay you have submitted for processing in the Create Replay wizard. (The Status column for this replay should show Completed.)

The Home tab of the Database Replay page appears, and the Replay Summary shows a Status of Completed.



- The replay line in the User Calls chart graphically represents the replay progress during the course of the entire replay from initiation to conclusion. The elapsed time in the Replay Progress chart equates to the number of minutes that were set up for the associated Capture.

The chart shows how much time it has taken to replay the same workload compared to the elapsed time during the workload capture in terms of user calls. If the Replay line is above or to the left of the Capture line, the replay system is processing the workload faster than the capture system.

- Under the Replay Divergence Summary, any errors and data discrepancies between the replay system and the capture system are displayed as diverged database calls during replay. You can use the percentage of total calls that diverged as a measure of the replay quality.

To view details about the diverged calls, click the link that corresponds to the type of diverged call in the Count column to access the Diverged Calls During Replay page. The Diverged Calls During Replay page shows the most relevant set of replayed calls that diverged from the workload captured by grouping them based on common attribute values and specified filter conditions. To view details about a particular diverged call—such as the call attributes, SQL

text, and bind variables—click the corresponding link in the SQL ID column to bring up the Replay Diverged Statement page.

4. To return to the Database Replay page, click the Database Replay breadcrumb.

Tip: See [Chapter 12, "Analyzing Captured and Replayed Workloads"](#) for information about accessing workload replay reports.

Replaying a Database Workload Using APIs

This section describes how to replay a database workload using the DBMS_WORKLOAD_REPLAY package. You can also use Oracle Enterprise Manager to replay a database workload, as described in ["Replaying a Database Workload Using Enterprise Manager"](#) on page 11-8.

Replaying a database workload using the DBMS_WORKLOAD_REPLAY package is a multi-step process that involves:

- [Initializing Replay Data](#)
- [Remapping Connections](#)
- [Setting Workload Replay Options](#)
- [Defining Workload Replay Filters and Replay Filter Sets](#)
- [Setting the Replay Timeout Action](#)
- [Starting a Workload Replay](#)
- [Pausing a Workload Replay](#)
- [Resuming a Workload Replay](#)
- [Cancelling a Workload Replay](#)
- [Exporting AWR Data for Workload Replay](#)
- [Importing AWR Data for Workload Replay](#)

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about the DBMS_WORKLOAD_REPLAY package

Initializing Replay Data

After the workload capture is preprocessed and the test system is properly prepared, the replay data can be initialized. Initializing replay data loads the necessary metadata into tables required by workload replay. For example, captured connection strings are loaded into a table where they can be remapped for replay.

To initialize replay data:

- Use the INITIALIZE_REPLAY procedure:

```
BEGIN
  DBMS_WORKLOAD_REPLAY.INITIALIZE_REPLAY (replay_name => 'dec06_102',
                                           replay_dir => 'dec06');
END;
```

In this example, the INITIALIZE_REPLAY procedure loads preprocessed workload data from the dec06 directory into the database.

The INITIALIZE_REPLAY procedure in this example uses the following parameters:

- The `replay_name` required parameter specifies a replay name that can be used with other APIs to retrieve settings and filters of previous replays.
- The `replay_dir` required parameter specifies the directory that contains the workload capture that will be replayed.

See Also:

- ["Preprocessing a Database Workload Using APIs"](#) on page 10-9 for information about preprocessing a workload capture
- ["Setting Up the Test System"](#) on page 11-1 for information preparing the test system

Remapping Connections

After the replay data is initialized, connection strings used in the workload capture need to be remapped so that user sessions can connect to the appropriate databases and perform external interactions as captured during replay. To view connection mappings, use the `DBA_WORKLOAD_CONNECTION_MAP` view. For information about connection remapping, see ["Connection Remapping"](#) on page 11-3.

To remap connections:

- Use the `REMAP_CONNECTION` procedure:

```
BEGIN
  DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION (connection_id => 101,
                                          replay_connection => 'dlsun244:3434/bjava21');
END;
/
```

In this example, the connection that corresponds to the connection ID 101 will use the new connection string defined by the `replay_connection` parameter.

The `REMAP_CONNECTION` procedure in this example uses the following parameters:

- The `connection_id` required parameter is generated when initializing replay data and corresponds to a connection from the workload capture.
- The `replay_connection` optional parameter specifies the new connection string that will be used during workload replay.

Remapping Users

Asides from remapping connection strings, you can also use a new schema or user instead of the user captured in the workload capture. To view captured users, use the `DBA_WORKLOAD_USER_MAP` view. For information about user remapping, see ["User Remapping"](#) on page 11-3.

To remap users:

- Use the `SET_USER_MAPPING` procedure:

```
BEGIN
  DBMS_WORKLOAD_REPLAY.SET_USER_MAPPING (capture_user => 'PROD',
                                          replay_user => 'TEST');
END;
/
```

In this example, the PROD user used during capture is remapped to the TEST user during replay.

The SET_USER_MAPPING procedure in this example uses the following parameters:

- The capture_user required parameter specifies the username captured during the time of the workload capture.
- The replay_user required parameter specifies the username to which the captured user is remapped during replay. If this parameter is set to NULL, then the mapping is disabled.

Setting Workload Replay Options

After the replay data is initialized, and connections and users are remapped, you need to prepare the database for workload replay. For information about workload replay preparation, see ["Steps for Replaying a Database Workload"](#) on page 11-2.

To prepare workload replay on the replay system:

- Use the PREPARE_REPLAY procedure:

```
BEGIN
    DBMS_WORKLOAD_REPLAY.PREPARE_REPLAY (synchronization => TRUE,
                                         capture_sts => TRUE,
                                         sts_cap_interval => 300);
END;
/
```

In this example, the PREPARE_REPLAY procedure prepares a replay that has been previously initialized. The COMMIT order in the workload capture will be preserved. A SQL tuning set will also be captured in parallel with the workload replay.

The PREPARE_REPLAY procedure uses the following parameters:

- The synchronization required parameter determines if synchronization will be used during workload replay.

If this parameter is set to SCN, the COMMIT order in the captured workload will be preserved during replay and all replay actions will be executed only after all dependent COMMIT actions have completed. The default value is SCN.

If this parameter is set to OBJECT_ID, all replay actions will be executed only after all relevant COMMIT actions have completed. Relevant COMMIT actions must meet the following criteria:

- * Issued before the given action in the workload capture
- * Modified at least one of the database objects for which the given action is referencing, either implicitly or explicitly

Setting this parameter to OBJECT_ID allows for more concurrency during workload replays for COMMIT actions that do not reference the same database objects during workload capture.

You can disable this option by setting the parameter to OFF, but the replay will likely yield significant replay divergence. However, this may be desirable if the workload consists primarily of independent transactions, and divergence during unsynchronized replay is acceptable.

- The connect_time_scale parameter scales the elapsed time from when the workload capture started to when the session connects with the specified

value and is interpreted as a % value. Use this parameter to increase or decrease the number of concurrent users during replay. The default value is 100.

- The `think_time_scale` parameter scales the elapsed time between two successive user calls from the same session and is interpreted as a % value. Setting this parameter to 0 will send user calls to the database as fast as possible during replay. The default value is 100.
- The `think_time_auto_correct` parameter corrects the think time (based on the `think_time_scale` parameter) between calls when user calls take longer to complete during replay than during capture. This parameter can be set to either `TRUE` or `FALSE`. Setting this parameter to `TRUE` reduces the think time if the workload replay is taking longer than the workload capture. The default value is `TRUE`.
- The `scale_up_multiplier` parameter defines the number of times the workload is scaled up during replay. Each captured session will be replayed concurrently for as many times as specified by this parameter. However, only one session in each set of identical replay sessions will execute both queries and updates. The rest of the sessions will only execute queries.
- The `capture_sts` parameter specifies whether to capture a SQL tuning set in parallel with the workload replay. If this parameter is set to `TRUE`, you can capture a SQL tuning set during workload replay and use SQL Performance Analyzer to compare it to another SQL tuning set without having to re-execute the SQL statements. This enables you to obtain a SQL Performance Analyzer report and compare the SQL performance—before and after the change—while running Database Replay. You can also export the resulting SQL tuning set with its AWR data using the `EXPORT_AWR` procedure, as described in ["Exporting AWR Data for Workload Replay"](#) on page 11-25.

This feature is not supported for Oracle RAC. Workload replay filters that are defined using `DBMS_WORKLOAD_REPLAY` do not apply to the SQL tuning set capture. The default value for this parameter is `FALSE`.
- The `sts_cap_interval` parameter specifies the duration of the SQL tuning set capture from the cursor cache in seconds. The default value is 300. Setting the value of this parameter below the default value may cause additional overhead with some workloads and is not recommended.

For more information about setting these parameters, see ["Specifying Replay Options"](#) on page 11-3.

Defining Workload Replay Filters and Replay Filter Sets

This section describes how to add and remove workload replay filters, and how to create and use replay filter sets. For information about using workload filters and replay filter sets with workload replay, see ["Using Filters with Workload Replay"](#) on page 11-4.

This section contains the following topics:

- [Adding Workload Replay Filters](#)
- [Deleting Workload Replay Filters](#)
- [Creating a Replay Filter Set](#)
- [Using a Replay Filter Set](#)

Adding Workload Replay Filters

This section describes how to add a new filter to be used in a replay filter set.

To add a new filter:

- Use the `ADD_FILTER` procedure:

```
BEGIN
  DBMS_WORKLOAD_REPLAY.ADD_FILTER (
    fname => 'user_ichan',
    fattribute => 'USER',
    fvalue => 'ICHAN');
END;
/
```

In this example, the `ADD_FILTER` procedure adds a filter named `user_ichan`, which can be used to filter out all sessions belonging to the user name `ICHAN`.

The `ADD_FILTER` procedure in this example uses the following parameters:

- The `fname` required parameter specifies the name of the filter that will be added.
- The `fattribute` required parameter specifies the attribute on which the filter will be applied. Valid values include `PROGRAM`, `MODULE`, `ACTION`, `SERVICE`, `USER`, and `CONNECTION_STRING`. You must specify a valid captured connection string that will be used during replay as the `CONNECTION_STRING` attribute.
- The `fvalue` required parameter specifies the value for the corresponding attribute on which the filter will be applied. It is possible to use wildcards such as `%` with some of the attributes, such as modules and actions.

Once all workload replay filters are added, you can create a replay filter set that can be used when replaying the workload.

Deleting Workload Replay Filters

This section describes how to delete workload replay filters.

To delete workload replay filters:

- Use the `DELETE_FILTER` procedure:

```
BEGIN
  DBMS_WORKLOAD_REPLAY.DELETE_FILTER (fname => 'user_ichan');
END;
/
```

In this example, the `DELETE_FILTER` procedure removes the filter named `user_ichan`.

The `DELETE_FILTER` procedure in this example uses the `fname` required parameter, which specifies the name of the filter to be removed.

Creating a Replay Filter Set

After the workload replay filters are added, you can create a set of replay filters to use with workload replay. When creating a replay filter set, all workload replay filters that were added since the previous replay filter set was created will be used.

To create a replay filter set:

- Use the `CREATE_FILTER_SET` procedure:

```
BEGIN
  DBMS_WORKLOAD_REPLAY.CREATE_FILTER_SET (
    replay_dir => 'apr09',
    filter_set => 'replayfilters',
    default_action => 'INCLUDE');
END;
/
```

In this example, the `CREATE_FILTER_SET` procedure creates a replay filter set named `replayfilters`, which will replay all captured calls for the replay stored in the `apr09` directory, except for the part of the workload defined by the replay filters.

The `CREATE_FILTER_SET` procedure in this example uses the following parameters:

- The `replay_dir` parameter specifies the directory where the replay to be filtered is stored
- The `filter_set` parameter specifies the name of the filter set to create
- The `default_action` parameter determines if every captured database call should be replayed and whether the workload replay filters should be considered as inclusion or exclusion filters.

If this parameter is set to `INCLUDE`, all captured database calls will be replayed, except for the part of the workload defined by the replay filters. In this case, all replay filters will be treated as exclusion filters, since they will define the part of the workload that will not be replayed. This is the default behavior.

If this parameter is set to `EXCLUDE`, none of the captured database calls will be replayed, except for the part of the workload defined by the replay filters. In this case, all replay filters will be treated as inclusion filters, since they will define the part of the workload that will be replayed.

Using a Replay Filter Set

Once the replay filter set is created and the replay is initialized, you can use the replay filter set to filter the replay in the `replay_dir` directory.

To use a replay filter set:

- Use the `USE_FILTER_SET` procedure:

```
BEGIN
  DBMS_WORKLOAD_REPLAY.USE_FILTER_SET (filter_set => 'replayfilters');
END;
/
```

In this example, the `USE_FILTER_SET` procedure uses the filter set named `replayfilters`.

The `USE_FILTER_SET` procedure in this example uses the `filter_set` required parameter, which specifies the name of the filter set to be used in the replay.

Setting the Replay Timeout Action

This section describes how to set a timeout action for the workload replay. You can set a replay timeout action to abort user calls that are significantly slower during replay or cause a replay to hang. For example, you may want to set a replay timeout action to abort runaway queries caused by sub-optimal execution plans following a database upgrade.

When a replay timeout action is enabled, a user call will exit with an ORA-15569 error if it is delayed beyond the conditions specified by the replay timeout action. The aborted call and its error are reported as error divergence.

To set a replay timeout:

- Use the SET_REPLAY_TIMEOUT procedure:

```
BEGIN
  DBMS_WORKLOAD_REPLAY.SET_REPLAY_TIMEOUT (
    enabled => TRUE,
    min_delay => 20,
    max_delay => 60,
    delay_factor => 10);
END;
/
```

In this example, the SET_REPLAY_TIMEOUT procedure defines a replay timeout action that will abort a user call if the delay during replay is more than 60 minutes, or if the delay during replay is over 20 minutes and the elapsed time is 10 times greater than the capture elapsed time.

The SET_REPLAY_TIMEOUT procedure in this example uses the following parameters:

- The enabled parameter specifies if the replay timeout action is enabled or disabled. The default value is TRUE.
- The min_delay parameter defines the lower bound value of call delay in minutes. The replay timeout action is only activated when the delay is over this value. The default value is 10.
- The max_delay parameter defines the upper bound value of call delay in minutes. The replay timeout action is activated and issues an error when the delay is over this value. The default value is 120.
- The delay_factor parameter defines a factor for the call delays that are between the values of min_delay and max_delay. The replay timeout action issues an error when the current replay elapsed time is higher than the multiplication of the capture elapsed time and this value. The default value is 8.

To retrieve the replay timeout action setting:

- Use the GET_REPLAY_TIMEOUT procedure:

```
DECLARE
  enabled          BOOLEAN;
  min_delay        NUMBER;
  max_delay        NUMBER;
  delay_factor     NUMBER;
BEGIN
  DBMS_WORKLOAD_REPLAY.GET_REPLAY_TIMEOUT(enabled, min_delay, max_delay,
    delay_factor);
END;
/
```

The GET_REPLAY_TIMEOUT procedure in this example returns the following parameters:

- The enabled parameter returns whether the replay timeout action is enabled or disabled.

- The `min_delay` parameter returns the lower bound value of call delay in minutes.
- The `max_delay` parameter returns the upper bound value of call delay in minutes.
- The `delay_factor` parameter returns the delay factor.

Starting a Workload Replay

Before starting a workload replay, you must first:

- Preprocess the captured workload, as described in ["Preprocessing a Database Workload Using APIs"](#) on page 10-9
- Initialize the replay data, as described in ["Initializing Replay Data"](#) on page 11-17
- Specify the replay options, as described in ["Setting Workload Replay Options"](#) on page 11-19
- Start the replay clients, as described in ["Starting Replay Clients"](#) on page 11-6

Note: Once a workload replay is started, new replay clients will not be able to connect to the database. Only replay clients that were started before the `START_REPLAY` procedure is executed will be used to replay the captured workload.

To start a workload replay:

- Use the `START_REPLAY` procedure:

```
BEGIN
  DBMS_WORKLOAD_REPLAY.START_REPLAY ();
END;
/
```

Pausing a Workload Replay

Pausing a workload replay will halt all subsequent user calls issued by the replay clients until the workload replay is either resumed or cancelled. User calls that are already in progress will be allowed to complete. This option enables you to temporarily stop the replay to perform a change and observe its impact for the remainder of the replay.

To pause a workload replay:

- Use the `PAUSE_REPLAY` procedure:

```
BEGIN
  DBMS_WORKLOAD_REPLAY.PAUSE_REPLAY ();
END;
/
```

Resuming a Workload Replay

This section describes how to resume a workload replay that is paused.

To resume a workload replay:

- Use the `RESUME_REPLAY` procedure:

```
BEGIN
```

```

        DBMS_WORKLOAD_REPLAY.RESUME_REPLAY ();
    END;
/

```

Cancelling a Workload Replay

This section describes how to cancel a workload replay.

To cancel a workload replay:

- Use the CANCEL_REPLAY procedure:

```

BEGIN
    DBMS_WORKLOAD_REPLAY.CANCEL_REPLAY ();
END;
/

```

Exporting AWR Data for Workload Replay

Exporting AWR data enables detailed analysis of the workload. This data is also required if you plan to run the AWR Compare Period report on a pair of workload captures or replays.

To export AWR data:

- Use the EXPORT_AWR procedure:

```

BEGIN
    DBMS_WORKLOAD_REPLAY.EXPORT_AWR (replay_id => 1);
END;
/

```

In this example, the AWR snapshots that correspond to the workload replay with a replay ID of 1 are exported, along with any SQL tuning set that may have been captured during workload replay.

The EXPORT_AWR procedure uses the `replay_id` required parameter, which specifies the ID of the replay whose AWR snapshots will be exported.

Note: This procedure works only if the corresponding workload replay was performed in the current database and the AWR snapshots that correspond to the original replay time period are still available.

Importing AWR Data for Workload Replay

After AWR data is exported from the replay system, you can import the AWR data into another system. Importing AWR data enables detailed analysis of the workload. This data is also required if you plan to run the AWR Compare Period report on a pair of workload captures or replays.

To import AWR data:

- Use the IMPORT_AWR function:

```

CREATE USER capture_awr
SELECT DBMS_WORKLOAD_REPLAY.IMPORT_AWR (replay_id => 1,
                                         staging_schema => 'capture_awr')
FROM DUAL;

```

In this example, the AWR snapshots that correspond to the workload replay with a capture ID of 1 are imported using a staging schema named `capture_awr`.

The `IMPORT_AWR` procedure in this example uses the following parameters:

- The `replay_id` required parameter specifies the ID of the replay whose AWR snapshots will be import.
- The `staging_schema` required parameter specifies the name of a valid schema in the current database which can be used as a staging area while importing the AWR snapshots from the replay directory to the `SYS AWR` schema.

Note: This function fails if the schema specified by the `staging_schema` parameter contains any tables with the same name as any of the AWR tables.

Monitoring Workload Replay Using APIs

This section describes how to monitor workload replay using APIs and views. You can also use Oracle Enterprise Manager to monitor workload replay, as described in ["Monitoring Workload Replay Using Enterprise Manager"](#) on page 11-15.

This section contains the following topics:

- [Retrieving Information About Diverged Calls](#)
- [Monitoring Workload Replay Using Views](#)

Retrieving Information About Diverged Calls

During replay, any error and data discrepancies between the replay system and the capture system are recorded as diverged calls.

To retrieve information about a diverged call—including its SQL identifier, SQL text, and bind values—call the `GET_DIVERGING_STATEMENT` function using the following parameters:

- Set the `replay_id` parameter to the ID of the replay in which the call diverged
- Set the `stream_id` parameter to the stream ID of the diverged call
- Set the `call_counter` parameter to the call counter of the diverged call

To view these information about a diverged call, use the `DBA_WORKLOAD_REPLAY_DIVERGENCE` view. The following example illustrates a function call:

```
DECLARE
  r                                CLOB;
  ls_stream_id                    NUMBER;
  ls_call_counter                 NUMBER;
  ls_sql_cd                      VARCHAR2(20);
  ls_sql_err                     VARCHAR2(512);
  CURSOR c IS
    SELECT stream_id, call_counter
    FROM DBA_WORKLOAD_REPLAY_DIVERGENCE
    WHERE replay_id = 72;
BEGIN
  OPEN c;
  LOOP
    FETCH c INTO ls_stream_id, ls_call_counter;
    EXIT when c%notfound;
    DBMS_OUTPUT.PUT_LINE (ls_stream_id||' '||ls_call_counter);
    r:=DBMS_WORKLOAD_REPLAY.GET_DIVERGING_STATEMENT(replay_id => 72,
    stream_id => ls_stream_id, call_counter => ls_call_counter);
```

```
DBMS_OUTPUT.PUT_LINE (r);  
END LOOP;  
END;  
/
```

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about the DBMS_WORKLOAD_REPLAY package

Monitoring Workload Replay Using Views

This section summarizes the views that you can display to monitor workload replay. You need DBA privileges to access these views.

- The DBA_WORKLOAD_CAPTURES view lists all the workload captures that have been captured in the current database.
- The DBA_WORKLOAD_FILTERS view lists all workload filters for workload captures defined in the current database.
- The DBA_WORKLOAD_REPLAYS view lists all the workload replays that have been replayed in the current database.
- The DBA_WORKLOAD_REPLAY_DIVERGENCE view enables you to view information about diverged calls, such as the replay identifier, stream identifier, and call counter.
- The DBA_WORKLOAD_REPLAY_FILTER_SET view lists all workload filters for workload replays defined in the current database.
- The DBA_WORKLOAD_CONNECTION_MAP view lists the connection mapping information for workload replay.
- The V\$WORKLOAD_REPLAY_THREAD view lists information about all sessions from the replay clients.

See Also:

- *Oracle Database Reference* for information about these views

Analyzing Captured and Replayed Workloads

This chapter describes how to analyze captured and replayed workloads using various Database Replay reports:

- Workload capture reports
Workload capture reports display information about captured workloads. See ["Using Workload Capture Reports"](#) on page 12-1.
- Workload replay reports
Workload replay reports measure performance differences between the capture system and the replay system. See ["Using Workload Replay Reports"](#) on page 12-4.
- Replay compare period reports
Replay compare period reports compare a workload replay to its capture, or to another replay of the same capture. If you are using Consolidated Database Replay, this report also compares multiple workload captures to a consolidated replay. See ["Using Replay Compare Period Reports"](#) on page 12-9.
- SQL Performance Analyzer reports
SQL Performance Analyzer reports compare a SQL tuning set from a workload replay to one from its workload capture, or two SQL tuning sets from two workload replays. See ["Using SQL Performance Analyzer Reports"](#) on page 12-14.

Note: After the replay analysis is complete, you can restore the database to its original state at the time of workload capture and repeat workload replay to test other changes to the system once the workload directory object is backed up to another physical location.

Using Workload Capture Reports

Workload capture reports contain captured workload statistics, information about the top session activities that were captured, and any workload filters used during the capture process.

The following sections describe how to generate and utilize workload capture reports:

- [Accessing Workload Capture Reports Using Enterprise Manager](#)
- [Generating Workload Capture Reports Using APIs](#)
- [Reviewing Workload Capture Reports](#)

Accessing Workload Capture Reports Using Enterprise Manager

This section describes how to generate a workload capture report using Oracle Enterprise Manager.

The primary tool for generating workload capture reports is Oracle Enterprise Manager. If for some reason Oracle Enterprise Manager is unavailable, you can generate workload capture reports using APIs, as described in ["Generating Workload Capture Reports Using APIs"](#) on page 12-2.

To access workload capture reports using Enterprise Manager:

1. From the Enterprise menu of the Enterprise Manager Cloud Control console, select **Quality Management**, then **Database Replay**.

If the Database Login page appears, log in as a user with administrator privileges.

The Database Replay page appears.

2. From the Captured Workloads tab of the Database Replay page for a capture that has a Status other than Completed, click the name of the desired capture from the Capture table.

The Summary tab of the Database Replay page appears.

3. Click the **Reports** tab for access to controls for the Workload Capture report and Workload Capture ASH Analytics report.

- The Workload Capture report contains detailed output showing the type of data components that were captured and not captured.

For information about how to interpret the workload capture report, see ["Reviewing Workload Capture Reports"](#) on page 12-3.

- The Capture ASH Analytics report shows which sessions are consuming the most database time. This report provides a stacked chart to help you visualize the active session activity for several dimensions, such as Event, Activity Class, Module/Action, Session, Instance ID, and PL/SQL function.

The "Other Activity" list choice for the report means that the activity has not been captured.

4. After you access a report, you can save it by clicking **Save**.

See Also:

- *Oracle Database Performance Tuning Guide* for information about ASH (Active Session History).

Generating Workload Capture Reports Using APIs

This section describes how to generate a workload capture report using the DBMS_WORKLOAD_CAPTURE package. You can also use Oracle Enterprise Manager to generate a workload capture report, as described in ["Accessing Workload Capture Reports Using Enterprise Manager"](#) on page 12-2.

To generate a report on the latest workload capture, use the DBMS_WORKLOAD_CAPTURE.GET_CAPTURE_INFO procedure and the DBMS_WORKLOAD_CAPTURE.REPORT function:

```
DECLARE
    cap_id          NUMBER;
    cap_rpt         CLOB;
BEGIN
```



```

cap_id := DBMS_WORKLOAD_CAPTURE.GET_CAPTURE_INFO(dir => 'dec06');
cap_rpt := DBMS_WORKLOAD_CAPTURE.REPORT(capture_id => cap_id,
                                         format => DBMS_WORKLOAD_CAPTURE.TYPE_TEXT);
END;
/

```

In this example, the `GET_CAPTURE_INFO` procedure retrieves all information regarding the workload capture in the `dec06` directory and returns the appropriate `cap_id` for the workload capture. The `REPORT` function generates a text report using the `cap_id` that was returned by the `GET_CAPTURE_INFO` procedure.

The `GET_CAPTURE_INFO` procedure uses the `dir` required parameter, which specifies the name of the workload capture directory object.

The `REPORT` function uses the following parameters:

- The `capture_id` required parameter relates to the directory that contains the workload capture for which the report will be generated. The directory should be a valid directory in the host system containing the workload capture. The value of this parameter should match the `cap_id` returned by the `GET_CAPTURE_INFO` procedure.
- The `format` required parameter specifies the report format. Valid values include `DBMS_WORKLOAD_CAPTURE.TYPE_TEXT` and `DBMS_WORKLOAD_REPLAY.TYPE_HTML`.

For information about how to interpret the workload capture report, see ["Reviewing Workload Capture Reports"](#) on page 12-3.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_WORKLOAD_CAPTURE` package

Reviewing Workload Capture Reports

The workload capture report contains various types of information that can be used to assess the validity of the workload capture. Using the information provided in this report, you can determine if the captured workload:

- Represents the actual workload you want to replay
- Does not contain any workload you want to exclude
- Can be replayed

The information contained in the workload capture report is divided into the following categories:

- Details about the workload capture (such as the name of the workload capture, defined filters, date, time, and SCN of capture)
- Overall statistics about the workload capture (such as the total DB time captured, and the number of logins and transactions captured) and the corresponding percentages with respect to total system activity
- Profile of the captured workload
- Profile of the workload that was not captured due to version limitations
- Profile of the uncaptured workload that were excluded using defined filters
- Profile of the uncaptured workload that consists of background process or scheduled jobs

Using Workload Replay Reports

Workload replay reports contain information that can be used to measure performance differences between the capture system and the replay system.

The following sections describe how to generate and review workload replay reports:

- [Accessing Workload Replay Reports Using Enterprise Manager](#)
- [Generating Workload Replay Reports Using APIs](#)
- [Reviewing Workload Replay Reports](#)

Accessing Workload Replay Reports Using Enterprise Manager

This section describes how to generate a workload replay report using Oracle Enterprise Manager.

The primary tool for generating workload replay reports is Oracle Enterprise Manager. If for some reason Oracle Enterprise Manager is unavailable, you can generate workload replay reports using APIs, as described in "[Generating Workload Replay Reports Using APIs](#)" on page 12-8

To access workload replay reports using Enterprise Manager:

1. From the Enterprise menu of the Enterprise Manager Cloud Control console, select **Quality Management**, then **Database Replay**.

If the Database Login page appears, log in as a user with administrator privileges.

The Database Replay page appears.

2. Click the **Replay Tasks** tab, then select a replay for which you want to access reports.
3. Click the **Reports** tab to gain access to individual reports.

There are several types of reports you can view for a completed workload replay:

- Database Replay

Use this report to view complete replay statistics in a tabular format, including replay divergence and the workload profile.

DB Replay Report for task1

DB Name	DB Id	Release	RAC	Replay Name	Replay Status
SIDB	1079228280	12.1.0.1.0	NO	task1	COMPLETED

Replay Information

Information	Replay	Capture
Name	task1	sidb_cap1
Status	COMPLETED	COMPLETED
Database Name	SIDB	SIDB
Database Version	12.1.0.1.0	12.1.0.1.0
Start Time	28-11-12 06:37:11	28-11-12 05:40:47
End Time	28-11-12 06:47:11	28-11-12 05:50:47
Duration	10 minutes 0 seconds	10 minutes 0 seconds
Directory Object	SIDB_REP1	SIDB_REP1
Directory Path	/scratch/wload/sidb_rep1	/scratch/wload/sidb_rep1
AWR DB Id	1079228280	
AWR Begin Snap Id	22	
AWR End Snap Id	23	
Replay Schedule Name		

Replay Options

Option Name	Value
Synchronization	SCN
Connect Time	100%
Think Time	100%
Think Time Auto Correct	TRUE
Number of WRC Clients	1 (1 Completed, 0 Running)

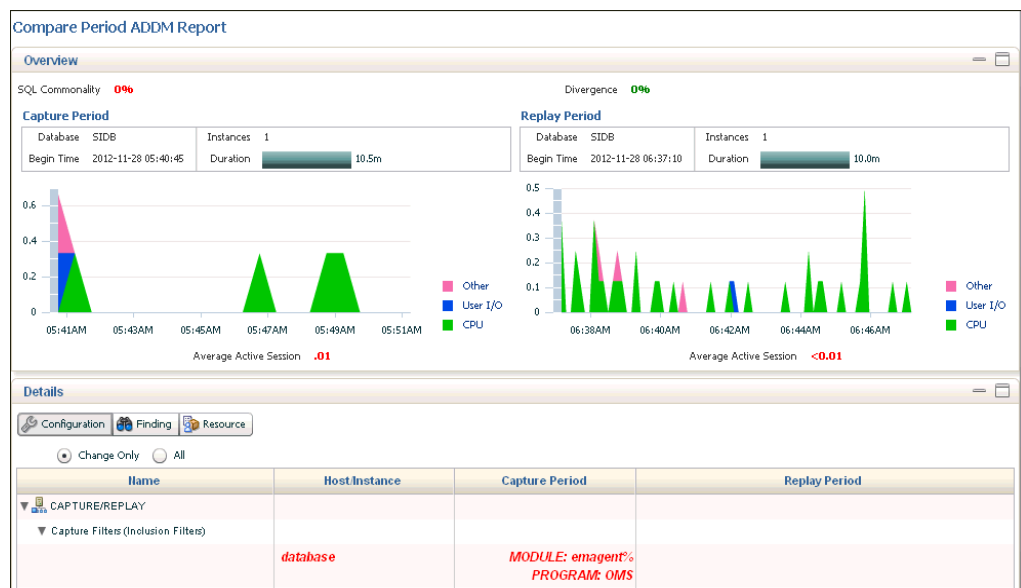
Replay Statistics

Statistic	Replay	Capture
DB Time	3.314 seconds	1.481 seconds
Average Active Sessions	.01	0
User calls	788	788

Replay Divergence Summary

■ Compare Period ADDM

Use this report to perform a high-level comparison of one workload replay to its capture or to another replay of the same capture. Only workload replays that contain at least 5 minutes of database time can be compared using this report.



Examine the following sections of the report to understand the performance change between the two periods and the cause of the change:

– Overview

This portion of the report shows SQL commonality, which is the comparability between the base and comparison periods based on the average resource consumption of the SQL statements common to both periods.

A commonality value of 100% means that the workload "signature" in both time periods is identical. A commonality of 100% is expected for this use case, because the workload being replayed is the same (assuming that you are not using replay filters). A value of 0% means that the two time periods have no items in common for the specific workload dimension.

Commonality is based on the type of input (that is, which SQL is executing) as well as the load of the executing SQL statements. Consequently, SQL statements running in only one time period, but not consuming significant time, do not affect commonality. Therefore, two workloads could have a commonality of 100% even if some SQL statements are running only in one of the two periods, provided that these SQL statements do not consume significant resources.

- Configuration

The information displayed shows base period and comparison period values for various parameters categorized by instance, host, and database.

- Findings

The findings can show performance improvements and identify the major performance differences caused by system changes. For negative outcomes, if you understand and remove the cause, the negative outcome can be eliminated.

The values shown for the Base Period and Comparison Period represent performance with regard to database time.

The Change Impact value represents a measurement of the scale of a change in performance from one time period to another. It is applicable to issues or items measured by the total database time they consumed in each time period. The absolute values are sorted in descending order.

If the value is positive, an improvement has occurred, and if the value is negative, a regression has occurred. For instance, a change impact of -200% means that period 2 is three times as slow as period 1.

You can run performance tuning tools, such as ADDM and the SQL Tuning Advisor, to fix issues in the comparison period to improve general system performance.

- Resources

The information shown provides a summary of the division of database time for both time periods, and shows the resource usage for CPU, memory, I/O, and interconnect (Oracle RAC only).

- SQL Performance Analyzer

Use this report to compare a SQL tuning set from a workload capture to another SQL tuning set from a workload replay, or two SQL tuning sets from two workload replays. Comparing SQL tuning sets with Database Replay provides more information than SQL Performance Analyzer test-execute, because it considers and shows all execution plans for each SQL statement, while SQL Performance Analyzer test-execute generates only one execution plan per SQL statement for each SQL trial.

- Replay Compare Period

Use this report to compare the AWR data from one workload replay to its capture or to another replay of the same capture. Before running this report, AWR data for the captured or replayed workload must have been previously exported.

Compare Period Report: Consolidated Replay

[Collapse all sections](#)

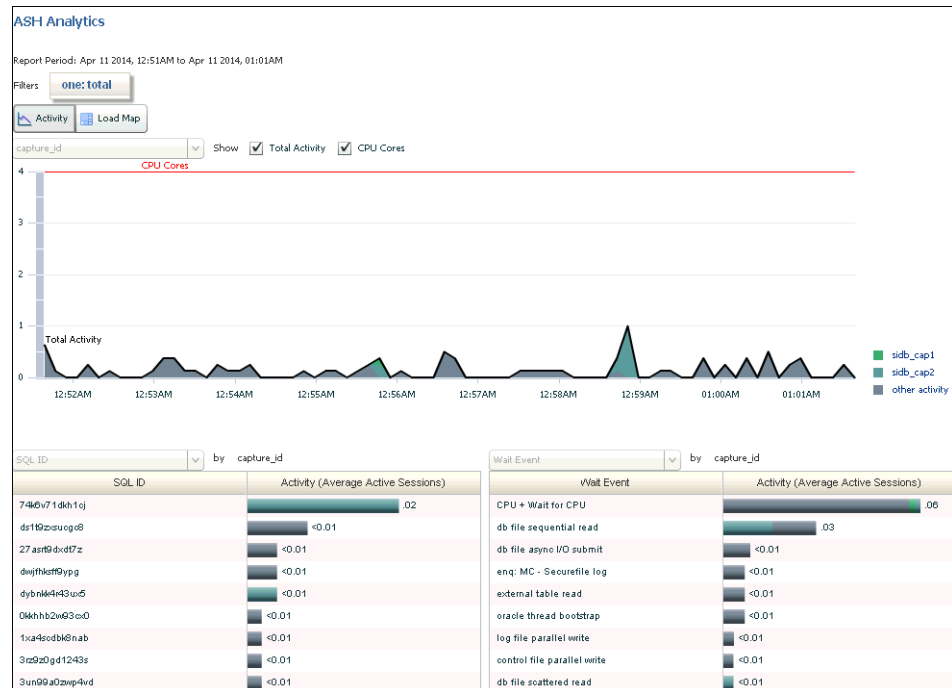
Comparison Metric	HR	HR_CRM_SALES	CRM	HR_CRM_SALES	SALES	HR_CRM_SALES
Total Sample Count	10	30	150	140	150	120
Total DB Time	30	50	188.03749	160	170	140
Total CPU Time	10	30	110	140	150	120
Total WAIT Time	20	20	78.03749	20	20	20
Total IO Time	20	20	20	20	20	20
IO Request Count	220	140	70	90	60	80
Avg Time Per IO Request	0.09090	0.14285	0.28571	0.22223	0.33334	0.25000

This report compares the performance of multiple captures with the consolidated replay. However, the comparison is made with individual ASH capture data vs ASH replay data filtered for the specific captured workload. Time Duration in this report is always represented in seconds

For information about using this report, see ["Reviewing Replay Compare Period Reports"](#) on page 12-11. For information about how to interpret replay compare period reports, see *Oracle Database 2 Day + Performance Tuning Guide*.

■ Replay ASH Analytics

The Replay ASH Analytics report contains active session history (ASH) information for a specified duration of a workload that was replayed for the category you selected in the drop-down menu. Before running this report, AWR data must have been previously exported from the captured or replayed workload.



The chart shows workload activity breakdown values for wait classes, and provides detailed statistics for the top activity sessions that are adversely affecting the system.

You can optionally use the Load Map for a graphical view of system activity. The Load Map is useful for viewing activity in a single- or multi-dimensional layout when you are not interested in seeing how activity has changed over time within the selected period.

For more information about this report, see the *Oracle Database 2 Day + Performance Tuning Guide*.

Generating Workload Replay Reports Using APIs

This section describes how to generate a workload replay report using the DBMS_WORKLOAD_REPLAY package. You can also use Oracle Enterprise Manager to generate a workload replay report, as described in ["Accessing Workload Replay Reports Using Enterprise Manager"](#) on page 12-4.

To generate a report on the latest workload replay for a workload capture, use the DBMS_WORKLOAD_REPLAY.GET_REPLAY_INFO procedure and the DBMS_WORKLOAD_REPLAY.REPORT function:

```
DECLARE
    cap_id      NUMBER;
    rep_id      NUMBER;
    rep_rpt     CLOB;
BEGIN
    cap_id := DBMS_WORKLOAD_REPLAY.GET_REPLAY_INFO(dir => 'dec06');
    /* Get the latest replay for that capture */
    SELECT max(id)
    INTO    rep_id
    FROM    dba_workload_replays
    WHERE   capture_id = cap_id;

    rep_rpt := DBMS_WORKLOAD_REPLAY.REPORT(replay_id => rep_id,
                                           format => DBMS_WORKLOAD_REPLAY.TYPE_TEXT);
END;
/
```

In this example, the GET_REPLAY_INFO procedure retrieves all information regarding the workload capture in the dec06 directory and the history of all the workload replay attempts from this directory. The procedure first imports a row into DBA_WORKLOAD_CAPTURES, which contains information about the workload capture. It then imports a row for every replay attempt retrieved from the replay directory into the DBA_WORKLOAD_REPLAYS view. The SELECT statement returns the appropriate rep_id for the latest replay of the workload. The REPORT function generates a text report using the rep_id that was returned by the SELECT statement.

The GET_CAPTURE_INFO procedure uses the dir required parameter, which specifies the name of the workload replay directory object.

The REPORT function uses the following parameters:

- The replay_id required parameter relates to the directory that contains the workload replay for which the report will be generated. The directory should be a valid directory in the host system containing the workload replay. The value of this parameter should match the rep_id returned by the GET_CAPTURE_INFO procedure.
- The format parameter required parameter specifies the report format. Valid values include DBMS_WORKLOAD_REPLAY.TYPE_TEXT, DBMS_WORKLOAD_REPLAY.TYPE_HTML, and DBMS_WORKLOAD_REPLAY.TYPE_XML.

For information about how to interpret the workload replay report, see ["Reviewing Workload Replay Reports"](#) on page 12-9.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_WORKLOAD_REPLAY` package

Reviewing Workload Replay Reports

After the workload is replayed on a test system, there may be some divergence in what is replayed compared to what was captured. There are numerous factors that can cause replay divergence, which can be analyzed using the workload replay report. The information contained in the workload replay report consists of performance and replay divergence.

Performance divergence may result when new algorithms are introduced in the replay system that affect the overall performance of the database. For example, if the workload is replayed on a newer version of Oracle Database, a new algorithm may cause specific requests to run faster, and the divergence will appear as a faster execution. In this case, this is a desirable divergence.

Data divergence occurs when the results of DML or SQL queries do not match results that were originally captured in the workload. For example, a SQL statement may return fewer rows during replay than those returned during capture.

Error divergence occurs when a replayed database call:

- Encounters a new error that was not captured
- Does not encounter an error that was captured
- Encounters a different error from what was captured

The information contained in the workload replay report are divided into the following categories:

- Details about the workload replay and the workload capture, such as job name, status, database information, duration and time of each process, and the directory object and path
- Replay options selected for the workload replay and the number of replay clients that were started
- Overall statistics about the workload replay and the workload capture (such as the total DB time captured and replayed, and the number of logins and transactions captured and replay) and the corresponding percentages with respect to total system activity
- Profile of the replayed workload
- Replay divergence
- Error divergence
- DML and SQL query data divergence

Using Replay Compare Period Reports

Use replay compare period reports to compare the performance of:

- A workload replay to its workload capture
- A workload replay to another replay of the same workload capture

- Multiple workload captures to a consolidated replay

The following sections describe how to generate and review replay compare period reports:

- [Generating Replay Compare Period Reports Using APIs](#)
- [Reviewing Replay Compare Period Reports](#)

See Also:

- [Chapter 14, "Using Consolidated Database Replay"](#) for information about Consolidated Database Replay

Generating Replay Compare Period Reports Using APIs

This section describes how to generate replay compare period reports using the DBMS_WORKLOAD_REPLAY package. This report only compares workload replays that contain at least 5 minutes of database time.

To generate replay compare period reports, use the DBMS_WORKLOAD_REPLAY.COMPARE_PERIOD_REPORT procedure:

```
BEGIN
  DBMS_WORKLOAD_REPLAY.COMPARE_PERIOD_REPORT (
    replay_id1 => 12,
    replay_id2 => 17,
    format => DBMS_WORKLOAD_CAPTURE.TYPE_HTML,
    result => :report_bind);
END;
/
```

In this example, the COMPARE_PERIOD_REPORT procedure generates a replay compare period report in HTML format that compares a workload replay with a replay ID of 12 with another replay with an ID of 17.

The COMPARE_PERIOD_REPORT procedure in this example uses the following parameters:

- The `replay_id1` parameter specifies the numerical identifier of the workload replay after change for which the reported will be generated. This parameter is required.
- The `replay_id2` parameter specifies the numerical identifier of the workload replay before change for which the reported will be generated. If unspecified, the comparison will be performed with the workload capture.
- The `format` parameter specifies the report format. Valid values include DBMS_WORKLOAD_CAPTURE.TYPE_HTML for HTML and DBMS_WORKLOAD_CAPTURE.TYPE_XML for XML. This parameter is required.
- The `result` parameter specifies the output of the report.

For information about how to interpret the replay compare period report, see ["Reviewing Replay Compare Period Reports"](#) on page 12-11.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about the DBMS_WORKLOAD_REPLAY package

Reviewing Replay Compare Period Reports

Reviewing replay compare period reports enables you to determine if any replay divergence occurred and whether there were any significant performance changes.

Depending on the type of comparison that is being made, one of three types of replay compare period reports is generated:

- **Capture vs. Replay**

This report type compares the performance of a workload replay to its workload capture.

- **Replay vs. Replay**

This report type compares the performance of two workload replays of the same workload capture.

- **Consolidated Replay**

This report type compares the performance of multiple workload captures to a consolidated replay. Only the ASH Data Comparison section is available for this report type. For more information about this report type, see ["Reporting and Analysis for Consolidated Database Replay"](#) on page 14-7.

All replay compare period report types contain information about the most important changes between the two runs that are being compared. Use this information to determine the appropriate action to take. For example, if a new concurrency issue is found, review Automatic Database Diagnostic Monitor (ADDM) reports to diagnose the issue. If a new SQL performance problem is found, run SQL Tuning Advisor to fix the problem.

The information in the replay compare period report is divided into the following sections:

- [General Information](#)
- [Replay Divergence](#)
- [Main Performance Statistics](#)
- [Top SQL/Call](#)
- [Hardware Usage Comparison](#)
- [ADDM Comparison](#)
- [ASH Data Comparison](#)

General Information

This section contains metadata about the two runs being compared in the report. Any `init.ora` parameter changes between the two runs are also shown here. Review this section to verify if the system change being tested was performed.

Replay Divergence

This section contains the divergence analysis of the second run relative to the first. If the analysis shows significant divergence, review the full divergence report.

Main Performance Statistics

This section contains a high-level performance statistic comparison across the two runs (such as change in database time). If the comparison shows an insignificant change in database time, then the performance between the two runs are generally

similar. If there is a significant change in database time, review the statistics to determine the component (CPU or user I/O) that is causing the greatest change.

Top SQL/Call

This section compares the performance change of individual SQL statements from one run to the next. The SQL statements are ordered by the total change in database time. Top SQL statements that regressed by the most database time are best candidates for SQL tuning.

Hardware Usage Comparison

This section compares CPU and I/O usage across the two runs. The number of CPUs is summed for all instances and CPU usage is averaged over instances.

I/O statistics are shown for data and temp files. A high value for the single block read time (much higher than 10 milliseconds) suggests that the system is I/O bound. If this is the case, then review the total read and write times to determine if the latency is caused by excessive I/O requests or poor I/O throughput.

ADDM Comparison

This section contains a comparison of ADDM analyses across the two runs ordered by the absolute difference in impact. Compare the ADDM results for the two runs to identify possible problems that only existed in one run. If the system change being tested is intended to improve database performance, then verify if the expected improvement is reflected in the ADDM findings.

ASH Data Comparison

This section compares the ASH data across the two runs. The begin time and end time of the comparison period are displayed in a table. These times may not match the capture and replay times of the two runs being compared. Instead, these times represent the times when the ASH samples were taken.

The ASH Data Comparison section contains the following subsections:

- [Compare Summary](#)
- [Top SQL](#)
- [Long Running SQL](#)
- [Common SQL](#)
- [Top Objects](#)

Compare Summary This section summarizes the activity during the two runs based on database time and wait time distribution:

- DB Time Distribution indicates how the total database time is distributed across CPU usage, wait times, and I/O requests.

[Figure 12–1](#) shows the DB Time Distribution subsection of a sample report.

Figure 12–1 DB Time Distribution

(-) DB Time Distribution		
Compare Attr	Time Period1	Time Period2
Total Sample Count	2050	7490
Total DB Time	2519.88801	10164.33483
Total CPU Time	2260	9130
Total WAIT Time	259.88801	1034.33483
Total IO Time	10	770
IO Request Count	2924	329819
Avg Time Per IO Request	00000000.00342	00000000.00233

- Wait Time Distribution indicates how the total wait time is distributed across wait events. The top wait event class, event name, and event count are listed for both runs.

Figure 12–2 shows the Wait Time Distribution subsection of a sample report.

Figure 12–2 Wait Time Distribution

(-) Wait Time Distribution		
Time Period (1)		
Event(1)	Wait time(1)	Event Count(1)
log file switch (checkpoint incomplete),Configuration	239.88801	12
direct path write temp,User I/O	10	2924
log buffer space,Configuration	10	6
Time Period (2)		
Event(2)	Wait time(2)	Event Count(2)
direct path read,User I/O	590	185443
log file switch (checkpoint incomplete),Configuration	114.33483	20
direct path read temp,User I/O	80	132686
free buffer waits,Configuration	70	435
direct path write temp,User I/O	50	73
db file sequential read,User I/O	50	11617
log file switch completion,Configuration	30	44
log buffer space,Configuration	30	34
latch: row cache objects,Concurrency	10	28
undo segment extension,Configuration	10	4

Top SQL This section displays the top SQL statements for both runs by total database time, CPU time, and wait time.

Long Running SQL This section displays the top long-running SQL statements for both runs. Each long-running SQL statement contains details about the query, such as the maximum, minimum, and average response times.

Common SQL This section extracts the SQL statements that are common in both runs and displays the top common SQL statements by variance in average response time and total database time.

Top Objects This section contains details about the top objects for both runs by total wait time.

Figure 12–3 shows the Top Objects section of a sample report.

Figure 12-3 Top Objects

(-) Top Objects By Total Wait time			
Time Period (1)			
Object Id(1)	Object Name(1)	Wait Time(1)	Owner(1)
89843	CR2	249.88801	SYS
Time Period (2)			
Object Id(2)	Object Name(2)	Wait Time(2)	Owner(2)
89843	CR2	700	SYS
89844	CR3	100	SYS
89842	CR1	50	SYS
89842	CR1	50	SYS

Using SQL Performance Analyzer Reports

Use the SQL Performance Analyzer report to compare a SQL tuning set from a workload replay to another SQL tuning set from a workload capture, or two SQL tuning sets from two workload replays.

Comparing SQL tuning sets with Database Replay provides more information than SQL Performance Analyzer test-execute because it considers and shows all execution plans for each SQL statement, while SQL Performance Analyzer test-execute generates only one execution plan per SQL statement for each SQL trial.

Generating SQL Performance Analyzer Reports Using APIs

This section describes how to generate a SQL Performance Analyzer report using the DBMS_WORKLOAD_REPLAY package.

To generate a SQL Performance Analyzer report, use the DBMS_WORKLOAD_REPLAY.COMPARE_SQLSET_REPORT procedure:

```
BEGIN
  DBMS_WORKLOAD_REPLAY.COMPARE_SQLSET_REPORT (
    replay_id1 => 12,
    format => DBMS_WORKLOAD_CAPTURE.TYPE_HTML,
    result => :report_bind);
END;
/
```

In this example, the COMPARE_SQLSET_REPORT procedure generates a SQL Performance Analyzer report in HTML format that compares a SQL tuning set captured during the workload replay with a replay ID of 12 to a SQL tuning set captured during the workload capture.

The COMPARE_SQLSET_REPORT procedure in this example uses the following parameters:

- The `replay_id1` parameter specifies the numerical identifier of the workload replay after change for which the reported will be generated. This parameter is required.
- The `replay_id2` parameter specifies the numerical identifier of the workload replay after change for which the reported will be generated. If unspecified, the comparison will be performed with the workload capture.
- The `format` parameter specifies the report format. Valid values include `DBMS_WORKLOAD_CAPTURE.TYPE_HTML` for HTML, `DBMS_WORKLOAD_CAPTURE.TYPE_XML` for XML, and `DBMS_WORKLOAD_CAPTURE.TYPE_TEXT` for text. This parameter is required.

- The result parameter specifies the output of the report.

For information about how to interpret the SQL Performance Analyzer report, see ["Reviewing the SQL Performance Analyzer Report in Command-Line"](#) on page 6-12.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_WORKLOAD_REPLAY` package

Using Workload Intelligence

Workload Intelligence analyzes the data stored in a captured workload and identifies significant patterns in a workload. This chapter describes how to use Workload Intelligence and contains the following sections:

- [Overview of Workload Intelligence](#)
- [Analyzing Captured Workloads Using Workload Intelligence](#)
- [Example: Workload Intelligence Results](#)

Overview of Workload Intelligence

Workload capture generates a number of binary files—called capture files—that contain relevant information about the captured workload. The information stored in the capture files enables workload replay to realistically reproduce the captured workload at a later time. For each client request that is recorded by workload capture, the captured information includes SQL text, bind values, transaction information, timing data, identifiers of accessed objects, and other information about the request.

Workload Intelligence enables you to use the information stored in capture files in additional ways, including the following:

- Analyze and model the workload
- Discover significant patterns and trends within the workload
- Visualize what was running on the production system during workload capture

This section describes Workload Intelligence and contains the following topics:

- [About Workload Intelligence](#)
- [Use Case for Workload Intelligence](#)
- [Requirements for Using Workload Intelligence](#)

See Also:

- ["Workload Capture"](#) on page 8-2 for information about workload capture
- ["Workload Replay"](#) on page 8-3 for information about workload replay

About Workload Intelligence

Workload Intelligence comprises a suite of Java programs that enable you to analyze the data stored in a captured workload. These Java programs operate on data recorded

during a workload capture to create a model that describes the workload. This model can help you identify significant patterns of templates that are executed as part of the workload.

A template represents a read-only SQL statement, or an entire transaction that consists of one or more SQL statements. If two SQL statements (or transactions) exhibit significant similarity, then they are represented by the same template.

Workload Intelligence enables you to better visualize what a captured workload looks like by exploring template patterns and the corresponding SQL statements. For each pattern, you can view important statistics, such as the number of executions of a given pattern and the database time consumed by the pattern during its execution.

Use Case for Workload Intelligence

SQL statements that are executed in a production system are typically not manually inputted by the users, but instead come from one or more applications running on an application server that is connected to the database server. There is usually a finite number of such SQL statements in an application. Even if different bind values are used in every execution of a particular statement, its SQL text essentially remains the same.

Depending on the user input to the application, a code path is executed that includes one or more SQL statements submitted to the database in a given order as defined by the application code. Frequent user actions correspond to application code paths that are regularly executed. Such frequently executed code paths generate a frequent pattern of SQL statements that are executed by the database in a given order. By analyzing a captured workload, Workload Intelligence discovers such patterns and associates them with related execution statistics. In other words, Workload Intelligence uses the information stored in capture files to discover patterns that are generated by significant code paths of applications running in the production system during workload capture. Workload Intelligence does this without requiring any information about the applications.

Using Workload Intelligence to discover significant patterns:

- Enables you to better visualize what was running in the database during workload capture.
- Provides more information that can be used for optimizations.
- Offers a better context because SQL statements are not isolated, but are combined.

Requirements for Using Workload Intelligence

Workload Intelligence uses the information that is stored in the capture files, and does not require the execution of the workload using workload replay. Furthermore, Workload Intelligence does not require any user schema, user data, or connection to the production system. To avoid any overhead in the production system, it is recommended that Workload Intelligence be used on a test system where the capture files have been copied, especially for large captured workloads because running Workload Intelligence may be resource intensive.

The necessary Java classes for invoking the Java programs that comprise Workload Intelligence are packed in `$ORACLE_HOME/rdbms/jlib/dbrintelligence.jar`. Two other jar files must be included in the classpath: `$ORACLE_HOME/rdbms/jlib/dbrparser.jar` and `$ORACLE_HOME/jdbc/lib/ojdbc6.jar`.

Workload Intelligence also uses some SYS tables and views internally.

Analyzing Captured Workloads Using Workload Intelligence

This section describes the steps in using Workload Intelligence.

To analyze captured workloads using Workload Intelligence:

1. Create a database user with the appropriate privileges to use Workload Intelligence, as described in ["Creating a Database User for Workload Intelligence"](#) on page 13-3.
2. Create a new Workload Intelligence job by running the LoadInfo Java program, as described in ["Creating a Workload Intelligence Job"](#) on page 13-3.
3. Generate a model that describes the workload by running the BuildModel Java program, as described in ["Generating a Workload Model"](#) on page 13-4.
4. Identify patterns in templates that occur in the workload by running the FindPatterns Java program, as described in ["Identifying Patterns in a Workload"](#) on page 13-5.
5. Generate a report to display the results by running the GenerateReport Java program, as described in ["Generating a Workload Intelligence Report"](#) on page 13-6.

Creating a Database User for Workload Intelligence

Before using Workload Intelligence, first create a database user with the appropriate privileges.

[Example 13-1](#) shows how to create a database user that can use Workload Intelligence.

Example 13-1 Creating a Database User for Workload Intelligence

```
create user workintusr identified by password;
grant create session to workintusr;
grant select,insert,alter on WI$_JOB to workintusr;
grant insert,alter on WI$_TEMPLATE to workintusr;
grant insert,alter on WI$_STATEMENT to workintusr;
grant insert,alter on WI$_OBJECT to workintusr;
grant insert,alter on WI$_CAPTURE_FILE to workintusr;
grant select,insert,alter on WI$_EXECUTION_ORDER to workintusr;
grant insert,update,delete,alter on WI$_FREQUENT_PATTERN to workintusr;
grant insert,delete,alter on WI$_FREQUENT_PATTERN_ITEM to workintusr;
grant insert,delete,alter on WI$_FREQUENT_PATTERN_METADATA to workintusr;
grant select on WI$_JOB_ID to workintusr;
grant execute on DBMS_WORKLOAD_REPLAY to workintusr;
```

Creating a Workload Intelligence Job

To create a Workload Intelligence job, use the LoadInfo program. LoadInfo is a Java program that creates a new task to apply the algorithms of Workload Intelligence. The program parses the data contained in a capture directory and stores the relevant information required for running Workload Intelligence in internal tables.

The LoadInfo program uses the following syntax:

```
java oracle.dbreplay.workload.intelligence.LoadInfo -cstr connection_string -user
username -job job_name -cdir capture_directory
```

```
java oracle.dbreplay.workload.intelligence.LoadInfo -version
```

```
java oracle.dbreplay.workload.intelligence.LoadInfo -usage
```

The LoadInfo program supports the following options:

- `-cstr`
Specifies the JDBC connection string to the database where Workload Intelligence stores the information and intermediate results required for execution (for example, `jdbc:oracle:thin@hostname:portnum:ORACLE_SID`)
- `-user`
Specifies the database username. The user must have certain privileges for using Workload Intelligence.

For information about creating a database user with the appropriate privileges, see ["Creating a Database User for Workload Intelligence"](#) on page 13-3.
- `-job`
Specifies a name that uniquely identifies the Workload Intelligence job.
- `-cdir`
Specifies the operating system path of the capture directory to be analyzed by Workload Intelligence.
- `-version`
Displays the version information for the LoadInfo program.
- `-usage`
Displays the command-line options for the LoadInfo program.

[Example 13-2](#) shows how to create a workload intelligence job named `wijobsales` using the LoadInfo program.

Example 13-2 Creating a Workload Intelligence Job

```
java -classpath $ORACLE_HOME/rdbms/jlib/dbrintelligence.jar:  
$ORACLE_HOME/rdbms/jlib/dbrparser.jar:  
$ORACLE_HOME/jdbc/lib/ojdbc6.jar:  
oracle.dbreplay.workload.intelligence.LoadInfo -job wijobsales -cdir  
/test/captures/sales -cstr jdbc:oracle:thin:@myhost:1521:orcl -user workintusr
```

Generating a Workload Model

To generate a workload model, use the BuildModel program. BuildModel is a Java program that reads data from a captured workload (this data must be generated by the LoadInfo program) and generates a model that describes the workload. This model can then be used to identify frequent template patterns that occur in the workload.

The BuildModel program uses the following syntax:

```
java oracle.dbreplay.workload.intelligence.BuildModel -cstr connection_string  
-user username -job job_name
```

```
java oracle.dbreplay.workload.intelligence.BuildModel -version
```

```
java oracle.dbreplay.workload.intelligence.BuildModel -usage
```

The BuildModel program supports the following options:

- `-cstr`

Specifies the JDBC connection string to the database where Workload Intelligence stores the information and intermediate results required for execution (for example, `jdbc:oracle:thin@hostname:portnum:ORACLE_SID`)

- `-user`

Specifies the database username. The user must have certain privileges for using Workload Intelligence.

For information about creating a database user with the appropriate privileges, see ["Creating a Database User for Workload Intelligence"](#) on page 13-3.

- `-job`

Specifies a name that uniquely identifies the Workload Intelligence job.

- `-version`

Displays the version information for the BuildModel program.

- `-usage`

Displays the command-line options for the BuildModel program.

[Example 13-3](#) shows how to generate a workload model using the BuildModel program.

Example 13-3 Generating a Workload Model

```
java -classpath $ORACLE_HOME/rdbms/jlib/dbrintelligence.jar:
$ORACLE_HOME/rdbms/jlib/dbrparser.jar:
$ORACLE_HOME/jdbc/lib/ojdbc6.jar:
oracle.dbreplay.workload.intelligence.BuildModel -job wijobsales -cstr
jdbc:oracle:thin:@myhost:1521:orcl -user workintusr
```

Identifying Patterns in a Workload

To identify patterns in a workload, use the FindPatterns program. FindPatterns is a Java program that reads data from a captured workload (this data must be generated by the LoadInfo program) and its corresponding workload model (the workload model must be generated by the BuildModel program), and identifies frequent template patterns that occur in the workload.

The FindPatterns program uses the following syntax:

```
java oracle.dbreplay.workload.intelligence.FindPatterns -cstr connection_string
-user username -job job_name -t threshold
```

```
java oracle.dbreplay.workload.intelligence.FindPatterns -version
```

```
java oracle.dbreplay.workload.intelligence.FindPatterns -usage
```

The FindPatterns program supports the following options:

- `-cstr`

Specifies the JDBC connection string to the database where Workload Intelligence stores the information and intermediate results required for execution (for example, `jdbc:oracle:thin@hostname:portnum:ORACLE_SID`)

- `-user`

Specifies the database username. The user must have certain privileges for using Workload Intelligence.

For information about creating a database user with the appropriate privileges, see ["Creating a Database User for Workload Intelligence"](#) on page 13-3.

- `-job`
Specifies a name that uniquely identifies the Workload Intelligence job.
- `-t`
Specifies a threshold probability that defines when a transition from one template to the next is part of the same pattern or the border between two patterns. Valid values include real numbers in the range [0.0, 1.0]. Setting this value is optional; its default value is 0.5.
- `-version`
Displays the version information for the FindPatterns program.
- `-usage`
Displays the command-line options for the FindPatterns program.

[Example 13-4](#) shows how to identify frequent template patterns in a workload using the FindPatterns program.

Example 13-4 Identifying Patterns in a Workload

```
java -classpath $ORACLE_HOME/rdbms/jlib/dbrintelligence.jar:  
$ORACLE_HOME/rdbms/jlib/dbrparser.jar:  
$ORACLE_HOME/jdbc/lib/ojdbc6.jar:  
oracle.dbreplay.workload.intelligence.FindPatterns -job wijobsales -cstr  
jdbc:oracle:thin:@myhost:1521:orcl -user workintusr -t 0.2
```

Generating a Workload Intelligence Report

To generate a Workload Intelligence report, use the GenerateReport program. GenerateReport is a Java program that generates a report to display the results of Workload Intelligence. The Workload Intelligence report is an HTML page that displays the patterns identified in the workload.

The GenerateReport program uses the following syntax:

```
java oracle.dbreplay.workload.intelligence.GenerateReport -cstr connection_string  
-user username -job job_name -top top_patterns -out filename
```

```
java oracle.dbreplay.workload.intelligence.GenerateReport -version
```

```
java oracle.dbreplay.workload.intelligence.GenerateReport -usage
```

The GenerateReport program supports the following options:

- `-cstr`
Specifies the JDBC connection string to the database where Workload Intelligence stores the information and intermediate results required for execution (for example, `jdbc:oracle:thin@hostname:portnum:ORACLE_SID`)
- `-user`
Specifies the database username. The user must have certain privileges for using Workload Intelligence.

For information about creating a database user with the appropriate privileges, see ["Creating a Database User for Workload Intelligence"](#) on page 13-3.

- `-job`
Specifies a name that uniquely identifies the Workload Intelligence job.
- `-top`
Specifies a number that indicates how many patterns to display in the report. The patterns are ordered by different criteria (number of executions, DB time, and length) and only the defined number of top results are displayed. Setting this value is optional; its default value is 10.
- `-out`
Specifies the name of the file (in HTML format) where the report is stored. Setting this value is optional; its default value is based on the job name specified in the `-job` option.
- `-version`
Displays the version information for the `GenerateReport` program.
- `-usage`
Displays the command-line options for the `GenerateReport` program.

[Example 13–5](#) shows how to generate a Workload Intelligence report using the `GenerateReport` program.

Example 13–5 Generating a Workload Intelligence Report

```
java -classpath $ORACLE_HOME/rdbms/jlib/dbrintelligence.jar:
$ORACLE_HOME/rdbms/jlib/dbrparser.jar:
$ORACLE_HOME/jdbc/lib/ojdbc6.jar:
oracle.dbreplay.workload.intelligence.GenerateReport -job wijobsales -cstr
jdbc:oracle:thin:@myhost:1521:orcl -user workintusr -top 5 -out wijobsales.html
```

Example: Workload Intelligence Results

This section assumes a scenario where Workload Intelligence is used on a captured workload generated by Swingbench, a benchmark used for stress testing Oracle Database.

The most significant pattern discovered by Workload Intelligence consists of the following 6 templates:

```
SELECT product_id, product_name, product_description, category_id, weight_class,
       supplier_id, product_status, list_price, min_price, catalog_url
FROM product_information
WHERE product_id = :1;
```

```
SELECT p.product_id, product_name, product_description, category_id, weight_class,
       supplier_id, product_status, list_price, min_price, catalog_url,
       quantity_on_hand, warehouse_id
FROM product_information p, inventories i
WHERE i.product_id = :1 and i.product_id = p.product_id;
```

```
INSERT INTO order_items (order_id, line_item_id, product_id, unit_price, quantity)
VALUES (:1, :2, :3, :4, :5);
```

```
UPDATE orders
SET order_mode = :1, order_status = :2, order_total = :3
WHERE order_id = :4;
```

```
SELECT /*+ use_nl */ o.order_id, line_item_id, product_id, unit_price, quantity,
      order_mode, order_status, order_total, sales_rep_id, promotion_id,
      c.customer_id, cust_first_name, cust_last_name, credit_limit, cust_email
FROM orders o, order_items oi, customers c
WHERE o.order_id = oi.order_id
      AND o.customer_id = c.customer_id
      AND o.order_id = :1;
```

```
UPDATE inventories
      SET quantity_on_hand = quantity_on_hand - :1
WHERE product_id = :2
      AND warehouse_id = :3;
```

This pattern corresponds to a common user action for ordering a product. In this example, the identified pattern was executed 222,261 times (or approximately 8.21% of the total number of executions) and consumed 58,533.70 seconds of DB time (or approximately 11.21% of total DB time).

Another significant pattern discovered by Workload Intelligence in this example consists of the following 4 templates:

```
SELECT customer_seq.nextval
FROM dual;
```

```
INSERT INTO customers (customer_id, cust_first_name, cust_last_name, nls_language,
                      nls_territory, credit_limit, cust_email, account_mgr_id)
VALUES (:1, :2, :3, :4, :5, :6, :7, :8);
```

```
INSERT INTO logon
VALUES (:1, :2);
```

```
SELECT customer_id, cust_first_name, cust_last_name, nls_language, nls_territory,
      credit_limit, cust_email, account_mgr_id
FROM customers
WHERE customer_id = :1;
```

This pattern corresponds to the creation of a new customer account followed by a login in the system. In this example, the identified pattern was executed 90,699 times (or approximately 3.35% of the total number of executions) and consumed 17,484.97 seconds of DB time (or approximately 3.35% of total DB time).

Using Consolidated Database Replay

Database Replay enables you to capture a workload on the production system and replay it on a test system. This can be very useful when evaluating or adopting new database technologies because these changes can be tested on a test system without affecting the production system. However, if the new system being tested offers significantly better performance than the existing system, then Database Replay may not accurately predict how much additional workload can be handled by the new system.

For example, if you are consolidating multiple production systems into a single Oracle Exadata Machine, replaying a workload captured from one of the existing systems on Oracle Exadata Machine may result in much lower resource usage (such as host CPU and I/O) during replay because the new system is much more powerful. In this case, it is more useful to assess how the new system will handle the combined workloads from all existing systems, rather than that of a single workload from one system.

Consolidated Database Replay enables you to consolidate multiple workloads captured from one or multiple systems and replay them concurrently on a single test system. In this example, using Consolidated Database Replay will help you to assess how the database consolidation will affect the production system and if a single Oracle Exadata Machine can handle the combined workloads from the consolidated databases.

This chapter describes how to use Consolidated Database Replay and contains the following sections:

- [Use Cases for Consolidated Database Replay](#)
- [Steps for Using Consolidated Database Replay](#)
- [Using Consolidated Database Replay with Enterprise Manager](#)
- [Using Consolidated Database Replay with APIs](#)
- [Example: Replying a Consolidated Workload with APIs](#)

Use Cases for Consolidated Database Replay

Consolidated Database Replay enables you to replay multiple workloads captured from one or multiple systems concurrently. During the replay, every workload capture that is consolidated will start to replay when the consolidated replay begins.

Some typical use cases for Consolidated Database Replay include:

- [Database Consolidation Using Pluggable Databases](#)
- [Stress Testing](#)

- [Scale-Up Testing](#)

Each of these use cases can be performed using the procedures described in this chapter. In addition, you can employ various workload scale-up techniques when using Consolidated Database Replay, as described in [Chapter 15, "Using Workload Scale-Up"](#).

Database Consolidation Using Pluggable Databases

One use for Consolidated Database Replay is to assess if the system can handle the combined workload from a database consolidation. For example, assume that you want to consolidate the databases for the CRM, ERP, and SCM applications by migrating them to pluggable databases (PDBs). You can use Consolidated Database Replay to combine the captured workloads from the three applications and replay them concurrently on PDBs. For an example of this use case, see ["Example: Replaying a Consolidated Workload with APIs"](#) on page 14-18.

Stress Testing

Another use for Consolidated Database Replay is for stress testing or capacity planning. For example, assume that you are expecting the workload for the Sales application to double during the holiday season. You can use Consolidated Database Replay to test the added stress on the system by doubling the workload and replaying the combined workload. For an example of this use case, see ["Using Time Shifting"](#) on page 15-2.

Scale-Up Testing

A third use for Consolidated Database Replay is for scale-up testing. For example, assume that you want to test if your system can handle captured workloads from the Financials application and the Orders application concurrently. You can use Consolidated Database Replay to test the effects of the scaled-up workload on your system by combining the workloads and replaying them simultaneously. For examples of this use case, see ["Using Workload Folding"](#) on page 15-5 and ["Using Schema Remapping"](#) on page 15-7.

Steps for Using Consolidated Database Replay

This section describes the steps involved when using Consolidated Workload Replay and contains the following topics:

- [Capturing Database Workloads for Consolidated Database Replay](#)
- [Setting Up the Test System for Consolidated Database Replay](#)
- [Preprocessing Database Workloads for Consolidated Database Replay](#)
- [Replaying Database Workloads for Consolidated Database Replay](#)
- [Reporting and Analysis for Consolidated Database Replay](#)

Capturing Database Workloads for Consolidated Database Replay

Consolidated Database Replay does not require any special steps for capturing database workloads. The steps for capturing database workloads are exactly the same as for capturing a single workload for Database Replay, as described in [Chapter 9, "Capturing a Database Workload"](#).

This section contains the following topics for workload captures that are specific to Consolidated Database Replay:

- [Supported Types of Workload Captures](#)
- [Capture Subsets](#)

Supported Types of Workload Captures

Consolidated Database Replay supports multiple workloads captured from one or multiple systems running Oracle Database 9i Release 2 (release 9.2.0.8.0) or higher on one or multiple operating systems. For example, you can use workloads captured from one system running Oracle Database 9i Release 2 (release 9.2.0.8.0) on HP-UX and another system running Oracle Database 10g Release 2 (release 10.2.0.4.0) on AIX.

Note: Consolidated Database Replay is only available on Oracle Database 11g Release 2 (release 11.2.0.2.0) and higher.

Capture Subsets

Consolidated Database Replay enables you to transform existing workload captures into new, smaller pieces of capture subsets. You can then generate new workload captures from the capture subsets that can be used in different use cases, as described in "[Use Cases for Consolidated Database Replay](#)" on page 14-1.

A **capture subset** is a piece of a workload capture that is defined from an existing workload capture by applying a time range. The time range is specified as an offset from the start of the workload capture. All user workload captured within the specified time range are included in the defined capture subset.

For example, assume that a workload was captured from 2 a.m. to 8 p.m. and the peak workload is identified to be from 10 a.m. to 4 p.m. You can define a capture subset to represent the peak workload by applying a time range that starts at 8 hours after the start of the workload (or 10 a.m.) and ends at 14 hours after the start of the workload (or 4 p.m.).

However, if a capture subset only contains recorded user workload that satisfy the specified time range, user logins that occurred before the specified time range are not recorded. If these user logins are required for replay, then the capture subset may not be replayable. For example, if a user session starts at 9:30 a.m. and ends at 10:30 a.m. and the specified time range for the capture subset is 10:00 a.m. to 4:00 p.m., the replay may fail if the user login at 9:30 a.m. is not included in the workload. Similarly, the specified time range may also include incomplete user calls that are only partially recorded if a user sessions starts at 3:30 p.m. but does not complete until 4:30 p.m.

Consolidated Database Replay addresses this problem by including only incomplete user calls caused by the start time of the specified time range. To avoid including the same incomplete user calls twice if the workload capture is folded, incomplete user calls caused by the end time are *not* included by default. Therefore, a capture subset is essentially the minimal number of recorded user calls during a specified time range that are required for proper replay, including the necessary user logins, alter session statements, and incomplete user calls caused by the start time.

See Also: "[Generating Capture Subsets Using APIs](#)" on page 14-9

Setting Up the Test System for Consolidated Database Replay

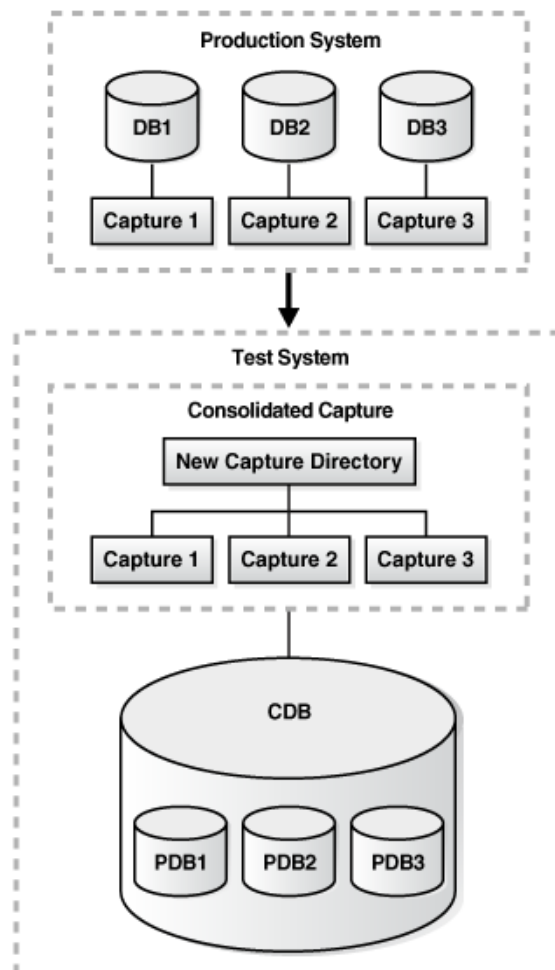
Setting up the test system for Consolidated Database Replay is similar to setting up a test system for Database Replay, as described in ["Setting Up the Test System"](#) on page 11-1. However, there are some additional considerations when setting up a replay database for Consolidated Database Replay.

To minimize divergence during the replay, the test system should contain the same application data and the state of the application data should be logically equivalent to that of the capture system at the start time of each workload capture. However, because a consolidated capture may contain multiple workload captures from different production systems, the test system needs to be set up for all the captures. In this case, it is recommended that the multitenant architecture be used to consolidate multiple databases, so that each database will have equivalent data to its capture system at the capture start time.

For Consolidated Database Replay, all participating workload captures must be placed under a new capture directory on the test system. You can copy all the workload captures into the new capture directory, or create symbolic links pointing to the original workload captures. Before consolidating the workload captures, ensure that the new capture directory has enough disk space to store all participating captures.

[Figure 14-1](#) illustrates how to set up the test system and new capture directory to consolidate three workload captures.

Figure 14-1 *Setting Up the Test System for Consolidated Database Replay*



See Also:

- ["Setting the Replay Directory Using APIs"](#) on page 14-9
- *Oracle Database Concepts* for information about the multitenant architecture

Preprocessing Database Workloads for Consolidated Database Replay

Preprocessing a database workload for Consolidated Database Replay is similar to preprocessing a database workload for Database Replay, as described in [Chapter 10, "Preprocessing a Database Workload"](#).

For Consolidated Database Replay, preprocess each captured workload into its own directory. Do not combine different workload captures into one directory for preprocessing. Preprocessing of captured workloads must be performed using a database running the same version of Oracle Database as that of the test system where the workloads will be replayed.

Replaying Database Workloads for Consolidated Database Replay

Replaying consolidated workloads using Consolidated Database Replay is quite different from replaying a single database workload using Database Replay.

This section contains the following topics for replaying workloads that are specific to Consolidated Database Replay:

- [Defining Replay Schedules](#)
- [Remapping Connections for Consolidated Database Replay](#)
- [Remapping Users for Consolidated Database Replay](#)
- [Preparing for Consolidated Database Replay](#)
- [Replaying Individual Workloads](#)

Defining Replay Schedules

A **replay schedule** adds one or multiple workload captures to a consolidated replay and specifies the order in which the captures will start during replay. A replay schedule must be created before a consolidated replay can be initialized. Multiple replay schedules can be defined for a consolidated replay. During replay initialization, you can select from any of the existing replay schedules.

Replay schedules perform two types of operation:

- [Adding Workload Captures](#)
- [Adding Schedule Orders](#)

See Also: ["Defining Replay Schedules Using APIs"](#) on page 14-10

Adding Workload Captures The first type of operation performed by a replay schedule is to add the participating workload captures to a replay.

When a workload capture is added to a replay schedule, an unique number is returned to identify the workload capture. A workload capture can be added to a replay schedule more than once, as it will be assigned a different capture number each time it is added. The replay schedule will point to the same capture directory each time to avoid a waste of disk space by copying the capture each time it is added.

See Also: ["Adding Workload Captures to Replay Schedules Using APIs"](#) on page 14-11

Adding Schedule Orders The second type of operation performed by a replay schedule is to add schedule orders that specify the order in which the participating workload captures will start during replay.

A **schedule order** defines an order between the start of two workload captures that have been added to the replay schedule. Multiple schedule orders can be added to a replay schedule. For example, assume that a replay schedule has three workload captures added. One schedule order can be added to specify that Capture 2 must wait for Capture 1 to complete before starting. Another schedule order can be added to specify that Capture 3 must wait for Capture 1 to complete before starting. In this case, both Capture 2 and Capture 3 must wait for Capture 1 to complete before starting.

It is possible for a replay schedule to not contain any schedule orders. In this case, all participating workload captures in the replay schedule will start to replay simultaneously when the consolidated replay begins.

See Also: ["Adding Schedule Orders to Replay Schedules Using APIs"](#) on page 14-12

Remapping Connections for Consolidated Database Replay

As in the case with replaying a single database workload using Database Replay, captured connection strings used to connect to the production system need to be remapped to the replay system, as described in ["Connection Remapping"](#) on page 11-3.

For Consolidated Database Replay, you need to remap captured connection strings from multiple workload captures to different connection strings during replay.

See Also: ["Remapping Connection Using APIs"](#) on page 14-16

Remapping Users for Consolidated Database Replay

As in the case with replaying a single database workload using Database Replay, usernames of database users and schemas used to connect to the production system can be remapped during replay, as described in ["User Remapping"](#) on page 11-3.

For Consolidated Database Replay, you can choose to remap the captured users from multiple workload captures to different users or schemas during replay.

See Also: ["Remapping Users Using APIs"](#) on page 14-16

Preparing for Consolidated Database Replay

As is the case with replaying a single database workload using Database Replay, replay options are defined during preparation of a replay, as described in ["Specifying Replay Options"](#) on page 11-3.

For Consolidated Database Replay, all participating workload captures in a consolidated replay use the same replay options during replay that are defined during replay preparation.

See Also: ["Preparing for Consolidated Database Replay Using APIs"](#) on page 14-17

Replaying Individual Workloads

It is recommended that each of the participating workloads be replayed individually before replaying the consolidated workload, as described in [Chapter 11, "Replaying a Database Workload"](#).

The individual replays can establish a baseline performance for each workload capture and be used to analyze the performance of the consolidated replay.

Reporting and Analysis for Consolidated Database Replay

Reporting and analysis for Consolidated Database Replay is performed using the replay compare period report, as described in ["Using Replay Compare Period Reports"](#) on page 12-9.

The replay compare period report for Consolidated Database Replay identifies the Active Session History (ASH) data for each individual workload capture and compares the ASH data from the workload capture to the filtered ASH data from the consolidated replay. Use this report to compare replays of the same consolidated workload capture.

The replay compare period report for Consolidated Database Replay treats the consolidated replay as multiple Capture vs. Replay comparisons. The summary section of the report contains a table that summarizes all individual Capture vs. Replay comparisons. Review the information in this section to gain a general understanding of how the consolidated replay ran.

[Figure 14-2](#) shows the summary section of a sample replay compare period report for Consolidated Database Replay.

Figure 14-2 Compare Period Report: Consolidated Replay

Compare Period Report: Consolidated Replay						
Collapse all sections						
Compare Attr	Capture "wrr-20120923-154838" with ID 61	Replay "cons_replay" with ID 85	Capture "wrr-20120924-000237" with ID 67	Replay "cons_replay" with ID 85	Capture "wrr-20120924-002300" with ID 70	Replay "cons_replay" with ID 85
Total Sample Count	2050	7490	1130	1600	1360	3280
Total DB Time	2519.88801	10164.33483	1784.11442	2212.95289	2368.00657	3720
Total CPU Time	2260	9130	1600	2050	2310	3580
Total WAIT Time	259.88801	1034.33483	184.11442	162.95289	58.00657	140
Total IO Time	10	770	40	10	40	100
IO Request Count	2924	329819	4473	3390	82366	42543
Avg Time Per IO Request	00000000.00342	00000000.00233	00000000.00894	00000000.00295	00000000.00049	00000000.00235

The rest of the sections in the report resemble the ASH Data Comparison section of the replay compare period report and are formed by joining all Capture vs. Replay reports in the consolidated replay. For a description of this section, see ["ASH Data Comparison"](#) on page 12-12.

Using Consolidated Database Replay with Enterprise Manager

This section describes how to use Consolidated Database Replay with Enterprise Manager.

The primary tool for replaying consolidated database workloads is Oracle Enterprise Manager. If Oracle Enterprise Manager is unavailable, you can also replay consolidated database workloads using APIs, as described in ["Using Consolidated Database Replay with APIs"](#) on page 14-8.

The process for replaying a consolidated database workload is nearly identical to that of replaying a single database workload. The differences are documented in the procedures for single replays in the following sections:

- ["Creating a Database Replay Task"](#) in [Chapter 10, "Preprocessing a Database Workload"](#)
- ["Preprocessing the Workload and Deploying the Replay Clients"](#) in [Chapter 10, "Preprocessing a Database Workload"](#)
- ["Replaying a Database Workload Using Enterprise Manager"](#) in [Chapter 11, "Replaying a Database Workload"](#)

The following list provides a summary of the differences between replaying a consolidated database workload versus replaying a single database workload:

- When creating a replay task, you need to select two or more captured workloads from the Select Captures table in the Create Task page.
- The Preprocess Captured Workload: Copy Workload step of the wizard has more than one choice for the Capture Name drop-down, so you may need to enter multiple credentials for the current location of the workload directory.
- The Preprocess Captured Workload: Select Directory step of the wizard does not display a Capture Summary as it does for single replays.
- The Replay Workload: Copy Workload step of the wizard has more than one choice for the Capture Name drop-down, so you may need to enter multiple credentials for the current location of the workload directory.
- The Replay Workload: Select Directory step of the wizard does not display a Capture Summary as it does for single replays.
- The Replay Workload: Initialize Options step of the wizard does not display the Identify Source section.
- The Replay Workload: Customize Options step of the wizard has more than one choice for the Capture Name drop-down in the Connection Mappings tab, so you can remap connections for each captured workload. The option to use a single connect descriptor or net service name is not available.

Using Consolidated Database Replay with APIs

This section describes how to create and replay consolidated workloads using the `DBMS_WORKLOAD_REPLAY` package. You can also create and replay consolidated workloads using Oracle Enterprise Manager, as described in ["Using Consolidated Database Replay with Enterprise Manager"](#) on page 14-7.

Creating and replay a consolidated workload using APIs is a multi-step process that involves:

- [Generating Capture Subsets Using APIs](#)
- [Setting the Replay Directory Using APIs](#)
- [Defining Replay Schedules Using APIs](#)
- [Running Consolidated Database Replay Using APIs](#)

See Also: *Oracle Database PL/SQL Packages and Types Reference* for information about the DBMS_WORKLOAD_REPLAY package

Generating Capture Subsets Using APIs

This section describes how to generate capture subsets from existing workload captures using the DBMS_WORKLOAD_REPLAY package. For information about capture subsets, see ["Capture Subsets"](#) on page 14-3.

To generate a capture subset from existing workload captures:

1. Use the GENERATE_CAPTURE_SUBSET procedure:

```
DBMS_WORKLOAD_REPLAY.GENERATE_CAPTURE_SUBSET (
    input_capture_dir      IN VARCHAR2,
    output_capture_dir     IN VARCHAR2,
    new_capture_name       IN VARCHAR2,
    begin_time             IN NUMBER,
    begin_include_incomplete IN BOOLEAN   DEFAULT TRUE,
    end_time               IN NUMBER,
    end_include_incomplete IN BOOLEAN   DEFAULT FALSE,
    parallel_level         IN NUMBER     DEFAULT NULL);
```

2. Set the input_capture_dir parameter to the name of the directory object that points to an existing workload capture.
3. Set the output_capture_dir parameter to the name of the directory object that points to an empty directory where the new workload capture will be stored.
4. Set the new_capture_name parameter to the name of the new workload capture that is to be generated.
5. Set the other parameters, which are optional, as appropriate.

For information about these parameters, see *Oracle Database PL/SQL Packages and Types Reference*.

[Example 14-1](#) shows how to create a capture subset named peak_wkld at directory object peak_capdir from an existing workload capture at directory object rec_dir. The capture subset includes workload from 2 hours after the start of the workload capture (or 7,200 seconds) to 3 hours after the start of the workload capture (or 10,800 seconds).

Example 14-1 Generating a Capture Subset

```
EXEC DBMS_WORKLOAD_REPLAY.GENERATE_CAPTURE_SUBSET ('rec_dir', 'peak_capdir',
    'peak_wkld', 7200, TRUE, 10800, FALSE, 1);
```

See Also: *Oracle Database PL/SQL Packages and Types Reference* for information about the GENERATE_CAPTURE_SUBSET procedure

Setting the Replay Directory Using APIs

This section describes how to set the replay directory on the test system using the DBMS_WORKLOAD_REPLAY package. Set the replay directory to a directory on the test system that contains the workload captures to be replayed. For information about setting up the test system, see ["Setting Up the Test System for Consolidated Database Replay"](#) on page 14-4.

To set the replay directory:

1. Use the SET_REPLAY_DIRECTORY procedure:


```
DBMS_WORKLOAD_REPLAY.SET_REPLAY_DIRECTORY (  
    replay_dir IN VARCHAR2);
```

2. Set the `replay_dir` parameter to the name of the directory object that points to the operating system directory containing the workload captures to be used for workload consolidation.

[Example 14-2](#) shows how to set the replay directory to a directory object named `rep_dir`.

Example 14-2 Setting the Replay Directory

```
EXEC DBMS_WORKLOAD_REPLAY.SET_REPLAY_DIRECTORY ('rep_dir');
```

You can also use the `DBMS_WORKLOAD_REPLAY` package to view the current replay directory that has been set by the `SET_REPLAY_DIRECTORY` procedure.

To view the current replay directory that has been set:

- Use the `GET_REPLAY_DIRECTORY` function:

```
DBMS_WORKLOAD_REPLAY.GET_REPLAY_DIRECTORY RETURN VARCHAR2;
```

If no replay directory has been set, then the function returns `NULL`.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about the `SET_REPLAY_DIRECTORY` procedure
- *Oracle Database PL/SQL Packages and Types Reference* for information about the `GET_REPLAY_DIRECTORY` function

Defining Replay Schedules Using APIs

This section describes how to define replay schedules using the `DBMS_WORKLOAD_REPLAY` package. For information about replay schedules, see ["Defining Replay Schedules"](#) on page 14-5.

Before defining replay schedules, ensure that the following prerequisites are met:

- All workload captures are preprocessed using the `PROCESS_CAPTURE` procedure on a system running the same database version as the replay system, as described in [Chapter 10, "Preprocessing a Database Workload"](#).
- All capture directories are copied to the replay directory on the replay system
- Replay directory is set using the `SET_REPLAY_DIRECTORY` procedure, as described in ["Setting the Replay Directory Using APIs"](#) on page 14-9.
- Database state is not in replay mode

To define replay schedules:

1. Create a new replay schedule, as described in ["Creating Replay Schedules Using APIs"](#) on page 14-11.
2. Add workload captures to the replay schedule, as described in ["Adding Workload Captures to Replay Schedules Using APIs"](#) on page 14-11.
3. Add schedule orders to the replay schedule, as described in ["Adding Schedule Orders to Replay Schedules Using APIs"](#) on page 14-12.
4. Save the replay schedule, as described in ["Saving Replay Schedules Using APIs"](#) on page 14-14.

Creating Replay Schedules Using APIs

This section describes how to create replay schedules using the DBMS_WORKLOAD_REPLAY package. For information about replay schedules, see ["Defining Replay Schedules"](#) on page 14-5.

To create a replay schedule:

1. Use the BEGIN_REPLAY_SCHEDULE procedure:

```
DBMS_WORKLOAD_REPLAY.BEGIN_REPLAY_SCHEDULE (
    schedule_name IN VARCHAR2);
```

2. Set the schedule_name parameter to the name of this replay schedule.

Note: The BEGIN_REPLAY_SCHEDULE procedure initiates the creation of a reusable replay schedule. Only one replay schedule can be defined at a time. Calling this procedure again while a replay schedule is being defined will result in an error.

[Example 14-3](#) shows how to create a replay schedule named peak_schedule.

Example 14-3 Creating a Replay Schedule

```
EXEC DBMS_WORKLOAD_REPLAY.BEGIN_REPLAY_SCHEDULE ('peak_schedule');
```

See Also: *Oracle Database PL/SQL Packages and Types Reference* for information about the BEGIN_REPLAY_SCHEDULE procedure

Adding Workload Captures to Replay Schedules Using APIs

This section describes how to add workload captures to and remove workload captures from replay schedules using the DBMS_WORKLOAD_REPLAY package. For information about adding workload captures to replay schedules, see ["Adding Workload Captures"](#) on page 14-5.

Before adding workload captures to a replay schedule, ensure that the following prerequisite is met:

- A replay schedule to which the workload captures are to be added is created.

For information about creating a replay schedule, see ["Creating Replay Schedules Using APIs"](#) on page 14-11.

To add workload captures to a replay schedule:

1. Use the ADD_CAPTURE function:

```
DBMS_WORKLOAD_REPLAY.ADD_CAPTURE (
    capture_dir_name IN VARCHAR2,
    start_delay_seconds IN NUMBER DEFAULT 0,
    stop_replay IN BOOLEAN DEFAULT FALSE,
    take_begin_snapshot IN BOOLEAN DEFAULT FALSE,
    take_end_snapshot IN BOOLEAN DEFAULT FALSE,
    query_only IN BOOLEAN DEFAULT FALSE)
RETURN NUMBER;
```

```
DBMS_WORKLOAD_REPLAY.ADD_CAPTURE (
    capture_dir_name IN VARCHAR2,
    start_delay_seconds IN NUMBER,
    stop_replay IN VARCHAR2,
```

```
take_begin_snapshot IN VARCHAR2 DEFAULT 'N',
take_end_snapshot   IN VARCHAR2 DEFAULT 'N',
query_only          IN VARCHAR2 DEFAULT 'N')
RETURN NUMBER;
```

This function returns an unique identifier that identifies the workload capture in this replay schedule.

2. Set the `capture_dir_name` parameter to the name of the directory object that points to the workload capture under the top-level replay directory.

The directory must contain a valid workload capture that is preprocessed on a system running the same database version as the replay system.

3. Set the other parameters, which are optional, as appropriate.

For information about these parameters, see *Oracle Database PL/SQL Packages and Types Reference*.

[Example 14-4](#) shows how to add a workload capture named `peak_wkld` to a replay schedule by using the `ADD_CAPTURE` function in a `SELECT` statement.

Example 14-4 Adding a Workload Capture to a Replay Schedule

```
SELECT DBMS_WORKLOAD_REPLAY.ADD_CAPTURE ('peak_wkld')
FROM dual;
```

You can also use the `DBMS_WORKLOAD_REPLAY` package to remove workload captures from a replay schedule.

To remove workload captures from a replay schedule:

1. Use the `REMOVE_CAPTURE` procedure:

```
DBMS_WORKLOAD_REPLAY.REMOVE_CAPTURE (
    schedule_capture_number IN NUMBER);
```

2. Set the `schedule_capture_number` parameter to the unique identifier that identifies the workload capture in this replay schedule.

The unique identifier is the same identifier that was returned by the `ADD_CAPTURE` function when the workload capture was added to the replay schedule.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about the `ADD_CAPTURE` function
- *Oracle Database PL/SQL Packages and Types Reference* for information about the `REMOVE_CAPTURE` procedure

Adding Schedule Orders to Replay Schedules Using APIs

This section describes how to add schedule orders to and remove schedule orders from replay schedules using the `DBMS_WORKLOAD_REPLAY` package. For information about adding schedule orders to replay schedules, see ["Adding Schedule Orders"](#) on page 14-6.

Before adding schedule orders to a replay schedule, ensure that the following prerequisites are met:

- A replay schedule to which the schedule orders are to be added is created.

For information about creating a replay schedule, see ["Creating Replay Schedules Using APIs"](#) on page 14-11.

- All workload captures participating in the schedule order are added to the replay schedule.

For information about adding workload captures to a replay schedule, see ["Adding Workload Captures to Replay Schedules Using APIs"](#) on page 14-11.

Note: Adding schedule orders to a replay schedule is optional. If you do not add a schedule order to a replay schedule, then all workload captures added to the replay schedule will start to replay simultaneously when the consolidated replay begins.

To add schedule orders to a replay schedule:

1. Use the `ADD_SCHEDULE_ORDERING` function:

```
DBMS_WORKLOAD_REPLAY.ADD_SCHEDULE_ORDERING (
    schedule_capture_id IN VARCHAR2,
    wait_for_capture_id IN VARCHAR2)
RETURN NUMBER;
```

This function add a schedule order between two workload captures that have been added to the replay schedule. If a schedule order cannot be added, it returns a non-zero error code.

2. Set the `schedule_capture_id` parameter to the workload capture that you want to wait in this schedule order.
3. Set the `wait_for_capture_id` parameter to the workload capture that you want to be completed before the other workload capture can start in this schedule order.

You can also use the `DBMS_WORKLOAD_REPLAY` package to remove schedule orders from a replay schedule.

To remove schedule orders from a replay schedule:

1. Use the `REMOVE_SCHEDULE_ORDERING` procedure:

```
DBMS_WORKLOAD_REPLAY.REMOVE_SCHEDULE_ORDERING (
    schedule_capture_id IN VARCHAR2,
    wait_for_capture_id IN VARCHAR2);
```

2. Set the `schedule_capture_id` parameter to the workload capture waiting in this schedule order.
3. Set the `wait_for_capture_id` parameter to the workload capture that needs to be completed before the other workload capture can start in this schedule order.

To view schedule orders:

- Use the `DBA_WORKLOAD_SCHEDULE_ORDERING` view.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about the `ADD_SCHEDULE_ORDERING` function
- *Oracle Database PL/SQL Packages and Types Reference* for information about the `REMOVE_SCHEDULE_ORDERING` procedure
- *Oracle Database Reference* for information about the `DBA_WORKLOAD_SCHEDULE_ORDERING` view

Saving Replay Schedules Using APIs

This section describes how to save replay schedules that been defined using the `DBMS_WORKLOAD_REPLAY` package.

Before saving a replay schedule, ensure that the following prerequisites are met:

- A replay schedule that will be saved is created.
For information about creating a replay schedule, see ["Creating Replay Schedules Using APIs"](#) on page 14-11.
- All workload captures participating in the schedule order are added to the replay schedule.
For information about adding workload captures to a replay schedule, see ["Adding Workload Captures to Replay Schedules Using APIs"](#) on page 14-11.
- Any schedule orders that you want to use are added to the replay schedule (this step is optional).
For information about adding schedule orders to a replay schedule, see ["Adding Schedule Orders to Replay Schedules Using APIs"](#) on page 14-12.

To save a replay schedule:

- Use the `END_REPLAY_SCHEDULE` procedure:
`DBMS_WORKLOAD_REPLAY.END_REPLAY_SCHEDULE;`

This procedure completes the creation of a replay schedule. The replay schedule is saved and associated with the replay directory. Once a replay schedule is saved, you can use it for a consolidated replay.

To view replay schedules:

- Use the `DBA_WORKLOAD_REPLAY_SCHEDULES` view.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about the `END_REPLAY_SCHEDULE` procedure
- *Oracle Database Reference* for information about the `DBA_WORKLOAD_REPLAY_SCHEDULES` view

Running Consolidated Database Replay Using APIs

This section describes how to run Consolidated Database Replay using the `DBMS_WORKLOAD_REPLAY` package. For information about consolidated replay, see ["Replaying Database Workloads for Consolidated Database Replay"](#) on page 14-5.

Before running Consolidated Database Replay, ensure that the following prerequisites are met:

- All workload captures are preprocessed using the `PROCESS_CAPTURE` procedure on a system running the same database version as the replay system, as described in [Chapter 10, "Preprocessing a Database Workload"](#).
- All capture directories are copied to the replay directory on the replay system
- Replay directory is set using the `SET_REPLAY_DIRECTORY` procedure, as described in ["Setting the Replay Directory Using APIs"](#) on page 14-9.
- Database is logically restored to the same application state as that of all the capture systems at the start time of all workload captures.

To run Consolidated Database Replay:

1. Initialize the replay data, as described in ["Initializing Consolidated Database Replay Using APIs"](#) on page 14-15.
2. Remap connections strings, as described in ["Remapping Connection Using APIs"](#) on page 14-16.
3. Remap users, as described in ["Remapping Users Using APIs"](#) on page 14-16.
Remapping users is optional.
4. Prepare the consolidated replay, as described in ["Preparing for Consolidated Database Replay Using APIs"](#) on page 14-17.
5. Set up and start the replay clients, as described in ["Setting Up Replay Clients"](#) on page 11-5.
6. Start the consolidated replay, as described in ["Starting Consolidated Database Replay Using APIs"](#) on page 14-18.
7. Generate reports and perform analysis, as described in ["Reporting and Analysis for Consolidated Database Replay"](#) on page 14-7.

Initializing Consolidated Database Replay Using APIs

This section describes how to initialize the replay data for a consolidated replay using the `DBMS_WORKLOAD_REPLAY` package.

Initializing the replay data performs the following operations:

- Puts the database state in initialized mode for the replay of a consolidated workload.
- Points to the replay directory that contains all workload captures participating in the replay schedule.
- Loads the necessary metadata into tables required for replay.
For example, captured connection strings are loaded into a table where they can be remapped for replay.

To initialize Consolidated Database Replay:

1. Use the `INITIALIZED_CONSOLIDATED_REPLAY` procedure:

```
DBMS_WORKLOAD_REPLAY.INITIALIZE_CONSOLIDATED_REPLAY (
    replay_name      IN VARCHAR2,
    schedule_name    IN VARCHAR2);
```

2. Set the `replay_name` parameter to the name of the consolidated replay.
3. Set the `schedule_name` parameter to the name of the replay schedule to use.

The `schedule_name` parameter is the name of the replay schedule used during its creation, as described in ["Creating Replay Schedules Using APIs"](#) on page 14-11.

[Example 14-5](#) shows how to initialize a consolidated replay named `peak_replay` using the replay schedule named `peak_schedule`.

Example 14-5 Initializing Consolidated Database Replay

```
EXEC DBMS_WORKLOAD_REPLAY.INITIALIZE_CONSOLIDATED_REPLAY ('peak_replay',  
    'peak_schedule');
```

See Also: *Oracle Database PL/SQL Packages and Types Reference* for information about the `INITIALIZE_CONSOLIDATED_REPLAY` procedure

Remapping Connection Using APIs

This section describes how to remap connection strings for a consolidated replay using the `DBMS_WORKLOAD_REPLAY` package. For information about connection remapping, see ["Remapping Connections for Consolidated Database Replay"](#) on page 14-6.

To remap connection strings:

1. Use the `REMAP_CONNECTION` procedure:

```
DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION (  
    schedule_cap_id    IN NUMBER,  
    connection_id      IN NUMBER,  
    replay_connection  IN VARCHAR2);
```

This procedure remaps the captured connection to a new connection string for all participating workload captures in the replay schedule.

2. Set the `schedule_capture_id` parameter to a participating workload capture in the current replay schedule.

The `schedule_capture_id` parameter is the unique identifier returned when adding the workload capture to the replay schedule, as described in ["Adding Workload Captures to Replay Schedules Using APIs"](#) on page 14-11.

3. Set the `connection_id` parameter to the connection to be remapped.

The `connection_id` parameter is generated when replay data is initialized and corresponds to a connection from the workload capture.

4. Set the `replay_connection` parameter to the new connection string that will be used during replay.

See Also: *Oracle Database PL/SQL Packages and Types Reference* for information about the `REMAP_CONNECTION` procedure

Remapping Users Using APIs

This section describes how to remap users for a consolidated replay using the `DBMS_WORKLOAD_REPLAY` package. For information about remapping users, see ["Remapping Users for Consolidated Database Replay"](#) on page 14-6.

Before remapping users, ensure that the following prerequisites are met:

- Replay data is initialized, as described in ["Initializing Consolidated Database Replay Using APIs"](#) on page 14-15.
- The database state is not in replay mode.

To remap users:

1. Use the SET_USER_MAPPING procedure:

```
DBMS_WORKLOAD_REPLAY.SET_USER_MAPPING (
    schedule_cap_id IN NUMBER,
    capture_user    IN VARCHAR2,
    replay_user     IN VARCHAR2);
```

2. Set the schedule_capture_id parameter to a participating workload capture in the current replay schedule.

The schedule_capture_id parameter is the unique identifier returned when adding the workload capture to the replay schedule, as described in ["Adding Workload Captures to Replay Schedules Using APIs"](#) on page 14-11.

3. Set the capture_user parameter to the username of the user or schema captured during the time of the workload capture.
4. Set the replay_user parameter to the username of a new user or schema to which the captured user is remapped during replay.

If this parameter is set to NULL, then the mapping is disabled.

[Example 14-6](#) shows how to remap the PROD user used during capture to the TEST user during replay for the workload capture identified as 1001.

Example 14-6 Remapping an User

```
EXEC DBMS_WORKLOAD_REPLAY.SET_USER_MAPPING (1001, 'PROD', 'TEST');
```

See Also: *Oracle Database PL/SQL Packages and Types Reference* for information about the SET_USER_MAPPING procedure

Preparing for Consolidated Database Replay Using APIs

This section describes how to prepare a consolidated replay using the DBMS_WORKLOAD_REPLAY package. For information about preparing consolidated replays, see ["Preparing for Consolidated Database Replay"](#) on page 14-6.

Before preparing a consolidated replay, ensure that the following prerequisites are met:

- Replay data is initialized, as described in ["Initializing Consolidated Database Replay Using APIs"](#) on page 14-15.
 - Captured connections are remapped, as described in ["Remapping Connection Using APIs"](#) on page 14-16.
 - Users are mapped, as described in ["Remapping Users Using APIs"](#) on page 14-16.
- Remapping users is optional. However, if you are planning to remap users during replay, then it must be completed before preparing the consolidated replay.

Preparing a consolidated replay performs the following operations:

- Specifies the replay options, such as synchronization mode (or COMMIT order), session connection rate, and session request rate.
- Puts the database state in replay mode.
- Enables the start of replay clients.

Note: Consolidated Database Replay only supports unsynchronized mode and OBJECT_ID-based synchronization. SCN-based synchronization is currently not supported.

To prepare a consolidated replay:

- Use the `PREPARE_CONSOLIDATED_REPLAY` procedure:

```
DBMS_WORKLOAD_REPLAY.PREPARE_CONSOLIDATED_REPLAY (  
    synchronization          IN VARCHAR2 DEFAULT 'OBJECT_ID',  
    connect_time_scale       IN NUMBER   DEFAULT 100,  
    think_time_scale         IN NUMBER   DEFAULT 100,  
    think_time_auto_correct  IN BOOLEAN  DEFAULT TRUE,  
    capture_sts              IN BOOLEAN  DEFAULT FALSE,  
    sts_cap_interval         IN NUMBER   DEFAULT 300);
```

For information about these parameters and how to set them, see ["Specifying Replay Options"](#) on page 11-3.

See Also: *Oracle Database PL/SQL Packages and Types Reference* for information about the `PREPARE_CONSOLIDATED_REPLAY` procedure

Starting Consolidated Database Replay Using APIs

This section describes how to start a consolidated replay using the `DBMS_WORKLOAD_REPLAY` package.

Before starting a consolidated replay, ensure that the following prerequisites are met:

- The consolidated replay is prepared, as described in ["Preparing for Consolidated Database Replay Using APIs"](#) on page 14-17.
- An adequate number of replay clients are started.

For information about setting up and starting replay clients, see ["Setting Up Replay Clients"](#) on page 11-5.

To start a consolidated replay:

- Use the `START_CONSOLIDATED_REPLAY` procedure:

```
DBMS_WORKLOAD_REPLAY.START_CONSOLIDATED_REPLAY;
```

See Also: *Oracle Database PL/SQL Packages and Types Reference* for information about the `START_CONSOLIDATED_REPLAY` procedure

Example: Replaying a Consolidated Workload with APIs

This section assumes a scenario where workloads from three separate production systems running different versions of Oracle Database on various operating systems are being consolidated.

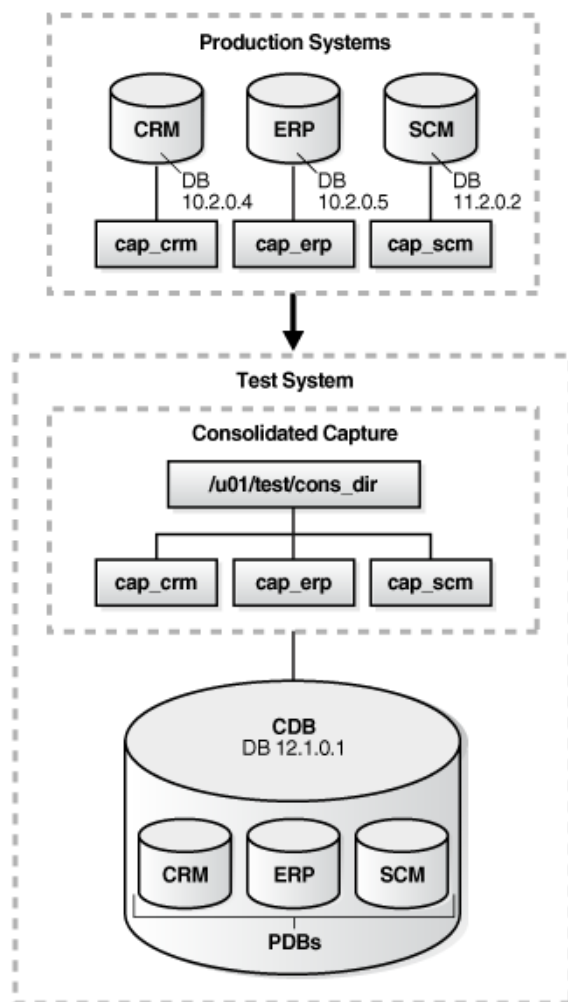
This scenario uses the following assumptions:

- The first workload to be consolidated is captured from the CRM system, which is running Oracle Database 10g Release 2 (release 10.2.0.4) on a Solaris server.
- The second workload to be consolidated is captured from the ERP system, which is running Oracle Database 10g Release 2 (release 10.2.0.5) on a Linux server.
- The third workload to be consolidated is captured from the SCM system, which is running Oracle Database 11g Release 2 (release 11.2.0.2) on a Solaris server.

- The test system is set up as a multitenant container database (CDB) running Oracle Database 12c Release 1 (release 12.1.0.1).
- The CDB contains three PDBs created from the CRM, ERP, and SCM systems.
- Each PDB contained within the CDB is restored to the same application data state as the CRM, ERP, and SCM systems at the capture start time.

Figure 14–3 illustrates this scenario.

Figure 14–3 Scenario for Consolidating Three Workloads



To consolidate the workloads and replay the consolidated workload in this scenario:

1. On the test system, preprocess the individual workload captures into separate directories:
 - For the CRM workload:
 - a. Create a directory object:


```
CREATE [OR REPLACE] DIRECTORY crm AS '/u01/test/cap_crm';
```
 - b. Ensure that the captured workload from the CRM system is stored in this directory.

c. Preprocess the workload:

```
EXEC DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE ('CRM');
```

■ **For the ERP workload:**

a. Create a directory object:

```
CREATE [OR REPLACE] DIRECTORY erp AS '/u01/test/cap_erp';
```

b. Ensure that the captured workload from the ERP system is stored in this directory.

c. Preprocess the workload:

```
EXEC DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE ('ERP');
```

■ **For the SCM workload:**

a. Create a directory object:

```
CREATE [OR REPLACE] DIRECTORY scm AS '/u01/test/cap_scm';
```

b. Ensure that the captured workload from the SCM system is stored in this directory.

c. Preprocess the workload:

```
EXEC DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE ('SCM');
```

2. Create a root directory to store the preprocessed workloads:

```
mkdir '/u01/test/cons_dir';
CREATE [OR REPLACE] DIRECTORY cons_workload AS '/u01/test/cons_dir';
```

3. Copy each preprocessed workload directory into the root directory:

```
cp -r /u01/test/cap_crm /u01/test/cons_dir
cp -r /u01/test/cap_erp /u01/test/cons_dir
cp -r /u01/test/cap_scm /u01/test/cons_dir
```

4. For each workload, create a directory object using the new operating system directory path:

```
CREATE [OR REPLACE] DIRECTORY crm AS '/u01/test/cons_dir/cap_crm';
CREATE [OR REPLACE] DIRECTORY erp AS '/u01/test/cons_dir/cap_erp';
CREATE [OR REPLACE] DIRECTORY scm AS '/u01/test/cons_dir/cap_scm';
```

5. Set the replay directory to the root directory previously created in Step 2:

```
EXEC DBMS_WORKLOAD_REPLAY.SET_REPLAY_DIRECTORY ('CONS_WORKLOAD');
```

6. Create a replay schedule and add the workload captures:

```
EXEC DBMS_WORKLOAD_REPLAY.BEGIN_REPLAY_SCHEDULE ('CONS_SCHEDULE');
SELECT DBMS_WORKLOAD_REPLAY.ADD_CAPTURE ('CRM') FROM dual;
SELECT DBMS_WORKLOAD_REPLAY.ADD_CAPTURE ('ERP') FROM dual;
SELECT DBMS_WORKLOAD_REPLAY.ADD_CAPTURE ('SCM') FROM dual;
EXEC DBMS_WORKLOAD_REPLAY.END_REPLAY_SCHEDULE;
```

7. Initialize the consolidated replay:

```
EXEC DBMS_WORKLOAD_REPLAY.INITIALIZE_CONSOLIDATED_REPLAY ('CONS_REPLAY',
'CONS_SCHEDULE');
```

8. Remap connections:

- a. Query the DBA_WORKLOAD_CONNECTION_MAP view for the connection mapping information:

```
SELECT schedule_cap_id, conn_id, capture_conn, replay_conn
FROM dba_workload_connection_map;
```

- b. Remap the connections:

```
EXEC DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION (schedule_cap_id => 1,
conn_id => 1, replay_connection => 'CRM');
EXEC DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION (schedule_cap_id => 2,
conn_id => 1, replay_connection => 'ERP');
EXEC DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION (schedule_cap_id => 3,
conn_id => 1, replay_connection => 'SCM');
```

The `replay_connection` parameter represents the services that are defined on the test system.

- c. Verify the connection remappings:

```
SELECT schedule_cap_id, conn_id, capture_conn, replay_conn
FROM dba_workload_connection_map;
```

9. Prepare the consolidated replay:

```
EXEC DBMS_WORKLOAD_REPLAY.PREPARE_CONSOLIDATED_REPLAY (
synchronization => 'OBJECT_ID');
```

10. Start replay clients:

- a. Estimate the number of replay clients that are required:

```
wrc mode=calibrate replaydir=/u01/test/cons_dir/cap_crm
wrc mode=calibrate replaydir=/u01/test/cons_dir/cap_erp
wrc mode=calibrate replaydir=/u01/test/cons_dir/cap_scm
```

- b. Add the output to determine the number of replay clients required.

You will need to start at least one replay client per workload capture contained in the consolidated workload.

- c. Start the required number of replay clients by repeating this command:

```
wrc username/password mode=replay replaydir=/u01/test/cons_dir
```

The `replaydir` parameter is set to the root directory in which the workload captures are stored.

11. Start the consolidated replay:

```
EXEC DBMS_WORKLOAD_REPLAY.START_CONSOLIDATED_REPLAY;
```

See Also:

- *Oracle Database Administrator's Guide* for information about configuring a CDB
- *Oracle Database Administrator's Guide* for information about creating PDBs

Using Workload Scale-Up

This chapter describes using various workload scale-up techniques with Consolidated Database Replay and contains the following sections:

- [Overview of Workload Scale-Up](#)
- [Using Time Shifting](#)
- [Using Workload Folding](#)
- [Using Schema Remapping](#)

Overview of Workload Scale-Up

Consolidated Database Replay enables you to replay multiple workloads captured from one or multiple systems concurrently. During the replay, every workload capture that is consolidated will start to replay when the consolidated replay begins.

Depending on the use case, you can employ various workload scale-up techniques when using Consolidated Database Replay. For information about typical use cases for Consolidated Database Replay, see "[Use Cases for Consolidated Database Replay](#)" on page 14-1.

This section describes the following workload scale-up techniques:

- [About Time Shifting](#)
- [About Workload Folding](#)
- [About Schema Remapping](#)

About Time Shifting

Database Replay enables you to perform time shifting when replaying captured workloads. This technique is useful in cases where you want to conduct stress testing on a system by adding workloads to an existing workload capture and replaying them together.

For example, assume that there are three workloads captured from three applications: Sales, CRM, and DW. In order to perform stress testing, you can align the peaks of these workload captures and replay them together using Consolidated Database Replay.

See Also:

- ["Using Time Shifting"](#) on page 15-2 for information about using time shifting
- ["Stress Testing"](#) on page 14-2 for information about using Consolidated Database Replay for stress testing

About Workload Folding

Database Replay enables you to perform scale-up testing by folding an existing workload capture. For example, assume that a workload was captured from 2 a.m. to 8 p.m. You can use Database Replay to fold the original workload into three capture subsets: one from 2 a.m. to 8 a.m., a second from 8 a.m. to 2 p.m., and a third from 2 p.m. to 8 p.m. By replaying the three capture subsets together, you can fold the original capture and triple the workload during replay to perform scale-up testing.

See Also:

- ["Using Workload Folding"](#) on page 15-5 for information about using workload folding
- ["Capture Subsets"](#) on page 14-3 for information about capture subsets
- ["Scale-Up Testing"](#) on page 14-2 for information about using Consolidated Database Replay for scale-up testing

About Schema Remapping

Database Replay enables you to perform scale-up testing by remapping database schemas. This technique is useful in cases when you are deploying multiple instances of the same application—such as a multi-tenant application—or adding a new geographical area to an existing application.

For example, assume that a single workload exists for a Sales application. To perform scale-up testing and identify possible host bottlenecks, set up the test system with multiple schemas from the Sales schema.

See Also:

- ["Using Schema Remapping"](#) on page 15-7 for information about using schema remapping
- ["Scale-Up Testing"](#) on page 14-2 for information about using Consolidated Database Replay for scale-up testing

Using Time Shifting

This section describes how to use time shifting with Consolidated Database Replay, and assumes a scenario where you want to use time shifting to align the peaks of workloads captured from three applications and replay them simultaneously. The scenario demonstrates how to use time shifting for stress testing. For more information about time shifting, see ["About Time Shifting"](#) on page 15-1.

This scenario uses the following assumptions:

- The first workload is captured from the Sales application.
- The second workload is captured from the CRM application and its peak time occurs 1 hour before that of the Sales workload.

- The third workload is captured from the DW application and its peak time occurs 30 minutes before that of the Sales workload.
- To align the peaks of these workloads, time shifting is performed by adding a delay of one hour to the CRM workload and a delay of 30 minutes to the DW workload during replay.

To perform time shifting in this scenario:

1. On the replay system which will undergo stress testing, create a directory object for the root directory where the captured workloads are stored:

```
CREATE [OR REPLACE] DIRECTORY cons_dir AS '/u01/test/cons_dir';
```

2. Preprocess the individual workload captures into separate directories:

- For the Sales workload:

- a. Create a directory object:

```
CREATE [OR REPLACE] DIRECTORY sales AS '/u01/test/cons_dir/cap_sales';
```

- b. Ensure that the captured workload from the Sales application is stored in this directory.

- c. Preprocess the workload:

```
EXEC DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE ('SALES');
```

- For the CRM workload:

- a. Create a directory object:

```
CREATE [OR REPLACE] DIRECTORY crm AS '/u01/test/cons_dir/cap_crm';
```

- b. Ensure that the captured workload from the CRM application is stored in this directory.

- c. Preprocess the workload:

```
EXEC DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE ('CRM');
```

- For the DW workload:

- a. Create a directory object:

```
CREATE [OR REPLACE] DIRECTORY DW AS '/u01/test/cons_dir/cap_dw';
```

- b. Ensure that the captured workload from the DW application is stored in this directory.

- c. Preprocess the workload:

```
EXEC DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE ('DW');
```

3. Set the replay directory to the root directory:

```
EXEC DBMS_WORKLOAD_REPLAY.SET_REPLAY_DIRECTORY ('CONS_DIR');
```

4. Create a replay schedule and add the workload captures:

```
EXEC DBMS_WORKLOAD_REPLAY.BEGIN_REPLAY_SCHEDULE ('align_peaks_schedule');
SELECT DBMS_WORKLOAD_REPLAY.ADD_CAPTURE ('SALES') FROM dual;
SELECT DBMS_WORKLOAD_REPLAY.ADD_CAPTURE ('CRM', 3600) FROM dual;
SELECT DBMS_WORKLOAD_REPLAY.ADD_CAPTURE ('DW', 1800) FROM dual;
EXEC DBMS_WORKLOAD_REPLAY.END_REPLAY_SCHEDULE;
```

Note that a delay of 3,600 seconds (or 1 hour) is added to the CRM workload, and a delay of 1,800 seconds (or 30 minutes) is added to the DW workload.

5. Initialize the consolidated replay:

```
EXEC DBMS_WORKLOAD_REPLAY.INITIALIZE_CONSOLIDATED_REPLAY ('align_peaks_replay',  
                                                         'align_peaks_schedule');
```

6. Remap connections:

a. Query the DBA_WORKLOAD_CONNECTION_MAP view for the connection mapping information:

```
SELECT schedule_cap_id, conn_id, capture_conn, replay_conn  
       FROM dba_workload_connection_map;
```

b. Remap the connections:

```
EXEC DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION (schedule_cap_id => 1,  
                                             conn_id => 1, replay_connection => 'inst1');  
EXEC DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION (schedule_cap_id => 1,  
                                             conn_id => 2, replay_connection => 'inst1');  
EXEC DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION (schedule_cap_id => 2,  
                                             conn_id => 1, replay_connection => 'inst2');  
EXEC DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION (schedule_cap_id => 2,  
                                             conn_id => 2, replay_connection => 'inst2');  
EXEC DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION (schedule_cap_id => 3,  
                                             conn_id => 1, replay_connection => 'inst3');  
EXEC DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION (schedule_cap_id => 3,  
                                             conn_id => 2, replay_connection => 'inst3');
```

The `replay_connection` parameter represents the services that are defined on the test system.

c. Verify the connection remappings:

```
SELECT schedule_cap_id, conn_id, capture_conn, replay_conn  
       FROM dba_workload_connection_map;
```

7. Prepare the consolidated replay:

```
EXEC DBMS_WORKLOAD_REPLAY.PREPARE_CONSOLIDATED_REPLAY;
```

8. Start replay clients:

a. Estimate the number of replay clients that are required:

```
wrc mode=calibrate replaydir=/u01/test/cons_dir/cap_sales  
wrc mode=calibrate replaydir=/u01/test/cons_dir/cap_crm  
wrc mode=calibrate replaydir=/u01/test/cons_dir/cap_dw
```

b. Add the output to determine the number of replay clients required.

You will need to start at least one replay client per workload capture contained in the consolidated workload.

c. Start the required number of replay clients by repeating this command:

```
wrc username/password mode=replay replaydir=/u01/test/cons_dir
```

The `replaydir` parameter is set to the root directory in which the workload captures are stored.

9. Start the consolidated replay:


```
EXEC DBMS_WORKLOAD_REPLAY.START_CONSOLIDATED_REPLAY;
```

Using Workload Folding

This section describes how to use workload folding with Consolidated Database Replay, and assumes a scenario where you want to use workload folding to triple a captured workload. The scenario demonstrates how to use workload folding for scale-up testing. For more information about workload folding, see ["About Workload Folding"](#) on page 15-2.

This scenario uses the following assumptions:

- The original workload was captured from 2 a.m. to 8 p.m. and folded into three capture subsets.
- The first capture subset contains part of the original workload from 2 a.m. to 8 a.m.
- The second capture subset contains part of the original workload from 8 a.m. to 2 p.m.
- The third capture subset contains part of the original workload from 2 p.m. to 8 p.m.
- To triple the workload during replay, workload folding is performed by replaying the three capture subsets simultaneously.

To perform workload folding in this scenario:

1. On the replay system where you plan to perform scale-up testing, create a directory object for the root directory where the captured workloads are stored:

```
CREATE [OR REPLACE] DIRECTORY cons_dir AS '/u01/test/cons_dir';
```

2. Create a directory object for the directory where the original workload is stored:

```
CREATE [OR REPLACE] DIRECTORY cap_monday AS '/u01/test/cons_dir/cap_monday';
```

3. Create directory objects for the directories where you are planning to store the capture subsets:

- a. Create a directory object for the first capture subset:

```
CREATE [OR REPLACE] DIRECTORY cap_mon_2am_8am
AS '/u01/test/cons_dir/cap_monday_2am_8am';
```

- b. Create a directory object for the second capture subset:

```
CREATE [OR REPLACE] DIRECTORY cap_mon_8am_2pm
AS '/u01/test/cons_dir/cap_monday_8am_2pm';
```

- c. Create a directory object for the third capture subset:

```
CREATE [OR REPLACE] DIRECTORY cap_mon_2pm_8pm
AS '/u01/test/cons_dir/cap_monday_2pm_8pm';
```

4. Create the capture subsets:

- a. Generate the first capture subset for the time period from 2 a.m. to 8 a.m.:

```
EXEC DBMS_WORKLOAD_REPLAY.GENERATE_CAPTURE_SUBSET ('CAP_MONDAY',
'CAP_MON_2AM_8AM', 'mon_2am_8am_wkld',
0, TRUE, 21600, FALSE, 1);
```

- b.** Generate the second capture subset for the time period from 8 a.m. to 2 p.m.:

```
EXEC DBMS_WORKLOAD_REPLAY.GENERATE_CAPTURE_SUBSET ('CAP_MONDAY',
                                                    'CAP_MON_8AM_2PM', 'mon_8am_2pm_wkld',
                                                    21600, TRUE, 43200, FALSE, 1);
```

- c.** Generate the third capture subset for the time period from 2 p.m. to 8 p.m.:

```
EXEC DBMS_WORKLOAD_REPLAY.GENERATE_CAPTURE_SUBSET ('CAP_MONDAY',
                                                    'CAP_MON_2PM_8PM', 'mon_2pm_8pm_wkld',
                                                    43200, TRUE, 0, FALSE, 1);
```

- 5.** Preprocess the capture subsets:

```
EXEC DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE ('CAP_MON_2AM_8AM');
EXEC DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE ('CAP_MON_8AM_2PM');
EXEC DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE ('CAP_MON_2PM_8PM');
```

- 6.** Set the replay directory to the root directory:

```
EXEC DBMS_WORKLOAD_REPLAY.SET_REPLAY_DIRECTORY ('CONS_DIR');
```

- 7.** Create a replay schedule and add the capture subsets:

```
EXEC DBMS_WORKLOAD_REPLAY.BEGIN_REPLAY_SCHEDULE ('monday_folded_schedule');
SELECT DBMS_WORKLOAD_REPLAY.ADD_CAPTURE ('CAP_MON_2AM_8AM') FROM dual;
SELECT DBMS_WORKLOAD_REPLAY.ADD_CAPTURE ('CAP_MON_8AM_2PM') FROM dual;
SELECT DBMS_WORKLOAD_REPLAY.ADD_CAPTURE ('CAP_MON_2PM_8PM') FROM dual;
EXEC DBMS_WORKLOAD_REPLAY.END_REPLAY_SCHEDULE;
```

- 8.** Initialize the consolidated replay:

```
EXEC DBMS_WORKLOAD_REPLAY.INITIALIZE_CONSOLIDATED_REPLAY (
    'monday_folded_replay', 'monday_folded_schedule');
```

- 9.** Remap connections:

- a.** Query the DBA_WORKLOAD_CONNECTION_MAP view for the connection mapping information:

```
SELECT schedule_cap_id, conn_id, capture_conn, replay_conn
FROM dba_workload_connection_map;
```

- b.** Remap the connections:

```
EXEC DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION (schedule_cap_id => 1,
                                             conn_id => 1, replay_connection => 'inst1');
EXEC DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION (schedule_cap_id => 1,
                                             conn_id => 2, replay_connection => 'inst1');
EXEC DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION (schedule_cap_id => 2,
                                             conn_id => 1, replay_connection => 'inst2');
EXEC DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION (schedule_cap_id => 2,
                                             conn_id => 2, replay_connection => 'inst2');
EXEC DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION (schedule_cap_id => 3,
                                             conn_id => 1, replay_connection => 'inst3');
EXEC DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION (schedule_cap_id => 3,
                                             conn_id => 2, replay_connection => 'inst3');
```

The `replay_connection` parameter represents the services that are defined on the test system.

- c.** Verify the connection remappings:

```
SELECT schedule_cap_id, conn_id, capture_conn, replay_conn
FROM dba_workload_connection_map;
```

10. Prepare the consolidated replay:

```
EXEC DBMS_WORKLOAD_REPLAY.PREPARE_CONSOLIDATED_REPLAY;
```

11. Start replay clients:

a. Estimate the number of replay clients that are required:

```
wrc mode=calibrate replaydir=/u01/test/cons_dir/cap_monday_2am_8am
wrc mode=calibrate replaydir=/u01/test/cons_dir/cap_monday_8am_2pm
wrc mode=calibrate replaydir=/u01/test/cons_dir/cap_monday_2pm_8pm
```

b. Add the output to determine the number of replay clients required.

You will need to start at least one replay client per workload capture contained in the consolidated workload.

c. Start the required number of replay clients by repeating this command:

```
wrc username/password mode=replay replaydir=/u01/test/cons_dir
```

The `replaydir` parameter is set to the root directory in which the workload captures are stored.

12. Start the consolidated replay:

```
EXEC DBMS_WORKLOAD_REPLAY.START_CONSOLIDATED_REPLAY;
```

Using Schema Remapping

This section describes how to use schema remapping with Consolidated Database Replay, and assumes a scenario where you want to use schema remapping to identify possible host bottlenecks when deploying multiple instances of an application. The scenario demonstrates how to use schema remapping for scale-up testing. For more information about schema remapping, see ["About Schema Remapping"](#) on page 15-2.

This scenario uses the following assumptions:

- A single workload exists that is captured from the Sales application.
- To set up the replay system with multiple schemas from the Sales schema, schema remapping is performed by adding the captured workload multiples times into a replay schedule and remapping the users to different schemas.

To perform schema remapping in this scenario:

1. On the replay system where you plan to perform scale-up testing, create a directory object for the root directory where the captured workloads are stored:

```
CREATE [OR REPLACE] DIRECTORY cons_dir AS '/u01/test/cons_dir';
```

2. Create a directory object for the directory where the captured workload is stored:

```
CREATE [OR REPLACE] DIRECTORY cap_sales AS '/u01/test/cons_dir/cap_sales';
```

Ensure that the captured workload from the Sales application is stored in this directory.

3. Preprocess the captured workload:

```
EXEC DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE ('SALES');
```

4. Set the replay directory to the root directory:

```
EXEC DBMS_WORKLOAD_REPLAY.SET_REPLAY_DIRECTORY ('CONS_DIR');
```

5. Create a replay schedule and add the captured workload multiple times:

```
EXEC DBMS_WORKLOAD_REPLAY.BEGIN_REPLAY_SCHEDULE ('double_sales_schedule');  
SELECT DBMS_WORKLOAD_REPLAY.ADD_CAPTURE ('CAP_SALES') FROM dual;  
SELECT DBMS_WORKLOAD_REPLAY.ADD_CAPTURE ('CAP_SALES') FROM dual;  
EXEC DBMS_WORKLOAD_REPLAY.END_REPLAY_SCHEDULE;
```

6. Initialize the consolidated replay:

```
EXEC DBMS_WORKLOAD_REPLAY.INITIALIZE_CONSOLIDATED_REPLAY (  
    'double_sales_replay', 'double_sales_schedule');
```

7. Remap the users:

```
EXEC DBMS_WORKLOAD_REPLAY.SET_USER_MAPPING (2, 'sales_usr', 'sales_usr_2');
```

8. Prepare the consolidated replay:

```
EXEC DBMS_WORKLOAD_REPLAY.PREPARE_CONSOLIDATED_REPLAY;
```

9. Start replay clients:

- a. Estimate the number of replay clients that are required:

```
wrc mode=calibrate replaydir=/u01/test/cons_dir/cap_sales
```

- b. Add the output to determine the number of replay clients required.

You will need to start at least one replay client per workload capture contained in the consolidated workload.

- c. Start the required number of replay clients by repeating this command:

```
wrc username/password mode=replay replaydir=/u01/test/cons_dir
```

The `replaydir` parameter is set to the root directory in which the workload captures are stored.

10. Start the consolidated replay:

```
EXEC DBMS_WORKLOAD_REPLAY.START_CONSOLIDATED_REPLAY;
```

Part III

Test Data Management

Oracle Database offers test data management features that enable you to:

- Store the list of applications, tables, and relationships between table columns using Application Data Modeling
- Replicate information that pertains only to a particular site using data subsetting
- Replace sensitive data from your production system with fictitious data that can be used during testing using Oracle Data Masking

Part III covers test data management and contains the following chapters:

- [Chapter 16, "Data Discovery and Modeling"](#)
- [Chapter 17, "Data Subsetting"](#)
- [Chapter 18, "Masking Sensitive Data"](#)

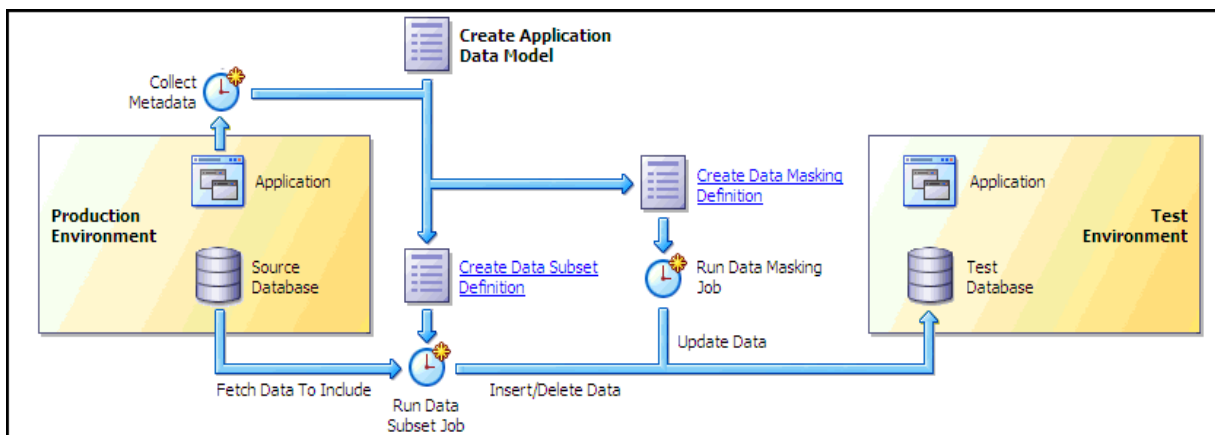
Data Discovery and Modeling

Oracle Data Masking Pack and Oracle Test Data Management Pack use Enterprise Manager's Data Discovery and Modeling capability to enable operations, such as sensitive data discovery, data subsetting, and data masking. Data Discovery and Modeling enables scanning and tagging of sensitive data and modeling of data relationships incorporated within an Application Data Model (ADM).

The ADM stores the list of applications, tables, and relationships between table columns that are either declared in the data dictionary, imported from application metadata, or user-specified. The ADM maintains sensitive data types and their associated columns, and is used by test data operations, such as data subsetting and data masking, to securely produce test data. Creating an ADM is a prerequisite for data subsetting and data masking operations.

Figure 16–1 shows the Application Data Model's relationship to other Data Discovery and Modeling components as well as the production and test environments.

Figure 16–1 Data Discovery and Modeling Architecture



You can perform several tasks related to Application Data Modeling, including the following tasks discussed in this chapter:

- [Creating an Application Data Model](#)
- [Managing Sensitive Column Types](#)
- [Associating a Database to an Application Data Model](#)
- [Importing and Exporting an Application Data Model](#)
- [Verifying or Upgrading a Source Database](#)

- [Using Self Update to Download the Latest Data Masking and Test Data Management Templates](#)

Note: The procedures in this chapter are applicable to Oracle Enterprise Manager 12.1 or higher Cloud Control only.

See Also:

- [Chapter 17, "Data Subsetting,"](#) for information about data subsetting
- [Chapter 18, "Masking Sensitive Data,"](#) for information about data masking

Creating an Application Data Model

The following procedure enables you to:

- Initiate creation of an Application Data Model (ADM)
- View and edit application tables
- View referential relationships
- Manually add a referential relationship
- Discover sensitive columns
- Set the type for sensitive columns

Before proceeding, ensure that you have the following privileges:

- EM_ALL_OPERATOR for Enterprise Manager Cloud Control users
- SELECT_CATALOG_ROLE for database users
- Select Any Dictionary privilege for database users

Note: When you create an ADM, the PL/SQL metadata collection packages are automatically deployed on the target database. The Database user must have DBA privileges to auto-deploy the packages.

To create an Application Data Model:

1. From the Data Discovery and Modeling page, view the diagram that shows how you can create a database for a test environment.

As the diagram shows, the first step is to create an ADM.

2. Create an ADM:

- a. Click **Create**.

A pop-up window requesting general properties information appears.

- b. Provide the required Name and Source Database.

The Source Database is the source from which the metadata is to be extracted.

- c. Select an Application Suite:

If you select **Custom Application Suite**:

- By default, metadata collection is enabled for the ADM creation process.
- If you uncheck "Create One Application For Each Schema," you create a shell ADM and will need to edit the ADM later to add applications and tables. Also, no metadata collection job is submitted, unlike the default choice.

If you select **Oracle Application Suite**:

- **Oracle E-Business Suite**—You provide database credentials for APPS user (or equivalent) and submit a job to create the ADM.
- **Oracle Fusion Applications**—You provide database credentials for FUSION user (or equivalent) and submit a job to create the ADM.

Note the following points about metadata collections:

- The metadata collection for the selected application suite populates the ADM with the applications and tables in the suite.
- The ADM can collect metadata for one or more schemas. An ADM application typically represents a schema. Each schema you select becomes an ADM application, and the ADM becomes populated with the tables in the schema, particularly in the case of custom applications. Note, however, that multiple applications can also map to a single schema, as in the case of Fusion Applications. The actual mapping depends on the application metadata discovered by the metadata collection job.

d. Click *Continue*.

Assuming that you selected Custom Application Suite, a Schemas pop-up appears in which you select schemas to include from the Available list.

e. Click *Continue*, provide the schedule parameters, then click *Submit* to submit the metadata collection job.

The ADM you created appears in the Application Data Models page. The Most Recent Job Status table column indicates that the metadata collection job is running. The model is locked, and you cannot edit it during this period until the status indicates that the job is complete.

3. View and edit application tables:

a. Select the model you created, then select *Edit*.

The Applications and Tables subpage appears, displaying the applications found during metadata collection.

To see the tables for an application, click the expand (>) icon.

b. To edit an application, select the application, open the *Actions* menu, then select *Add Table to Application*.

The Add Table to Application pop-up window appears.

c. Click the *Table* search icon.

The Search and Select pop-up appears, showing all of the tables from the selected schema that are not assigned to an application.

d. Select an unassigned table, then click *OK*.

The table name now appears in the Add Table to Application pop-up.

e. After selecting a Table Type, click *OK*.

The table now appears in the Applications and Tables view.

4. View referential relationships:

a. Click the **Referential Relationships** tab.

There are three types of referential relationships:

– Dictionary-defined

Upon opening this tab, this view shows the referential relationships that the metadata collection extracted, resulting from primary key and foreign key relationships. You can remove relationships from the ADM if desired.

– Imported from template

If there are application templates available from the vendor of the enterprise application, for example, Oracle Fusion Applications or Oracle E-Business Suite, then the ADM can be created from the application vendor-supplied template by using the Import action on the ADM home page.

– User-defined

See the step below about manually adding a referential relationship for more information.

b. Open an application view by selecting it, then using the chevron icon (>) to reveal parent and dependent key relationships, or by selecting **Expand All** from the **View** menu to view all relationships.

5. Manually add a referential relationship:

a. From the Referential Relationships tab, open the **Actions** menu, then select **Add Referential Relationship**.

The Add Referential Relationship pop-up window appears.

b. Select the requisite Parent Key and Dependent Key information.

c. In the Columns Name list, select a dependent key column to associate with a parent key column.

d. Click **OK** to add the referential relationship to the ADM.

The new dependent column now appears in the referential relationships list.

6. Discover sensitive columns automatically or add them manually:

To automatically discover sensitive columns:

a. From the Sensitive Columns tab, open the **Actions** menu, then select **Create Sensitive Column Discovery Job**.

The Parameters pop-up appears.

b. Select one or more applications and one or more sensitive column types.

Each type you select is processed for each application to search for columns that match the type.

c. Click **Continue**.

The schedule pop-up window appears.

d. Provide the required information, schedule the job, then click **Submit** when you have finished.

The Sensitive Columns subpage reappears.

e. Click **Save and Return** to return to the Application Data Models home page.

- f. When the Most Recent Job Status column indicates that the job is Successful, select the ADM, then click **Edit**.
- g. Select the **Sensitive Columns** tab, then click **Discovery Results** to view the job results.
- h. To set the sensitive status of any column, select the row for the column you want to define, open the **Set Status** menu, then select either **Sensitive** or **Not Sensitive**.
- i. Click **OK** to save and return to the Sensitive Columns tab.
The sensitive columns you defined in the previous step now appear in the list.
- j. Click **Save and Return** to return to the Application Data Models page.

To manually add sensitive columns:

- a. From the Application Data Models page, select an ADM, then click **Edit**.
- b. Select the **Sensitive Columns** tab, then click **Add**.
The Add Sensitive Column pop-up appears.
- c. Provide the required information and an optional Sensitive Column Type, then click **OK**.

The sensitive column now appears in the table for the Sensitive Columns tab.

- 7. Change the type for sensitive columns:
 - a. Click the **Sensitive Columns** tab.
This view shows the sensitive columns that have already been identified.
 - b. Select the sensitive column for which you want to change the type.
 - c. Open the **Actions** menu, then select **Set Sensitive Column Type**.
The Set Sensitive Column Type pop-up window appears.
 - d. Select the new type and click **OK**.

Managing Sensitive Column Types

After you have successfully created an ADM, the next task is to create either a new sensitive column type or one based on an existing type.

To create a sensitive column type:

- 1. From the Actions menu of the Application Data Models page, select **Sensitive Column Types**.
The Sensitive Column Types page appears.
- 2. Click **Create**.
The Create Sensitive Column Type pop-up appears.
- 3. Specify a required name and regular expressions for the Column Name, Column Comment, and Column Data search patterns.
 - The Or Search Type means that any of the patterns can match for a candidate sensitive column.
 - The And Search Type means that all of the patterns must match for a candidate sensitive column.

If you do not provide expressions for any of these parameters, the system does not search for the entity.

4. Click **OK**.

The sensitive column appears in the table in the Sensitive Column Types page.

To create a sensitive column type based on an existing type:

1. From the Actions menu of the Application Data Models page, select **Sensitive Column Types**.

The Sensitive Column Types page appears.

2. Select either a sensitive column type you have already defined, or select one from the out-of-box types that the product provides.

3. Click **Create Like**.

The Create Sensitive Column Type pop-up appears.

4. Specify a required name and alter the existing expressions for the Column Name, Column Comment, and Column Data search patterns to suit your needs.

5. Click **OK**.

The sensitive column appears in the table in the Sensitive Column Types page.

Associating a Database to an Application Data Model

After you have created an Application Data Model (ADM), you can select additional databases to be associated databases of an ADM, as explained in the following procedure. See ["Creating an Application Data Model"](#) for instructions on creating an ADM.

To associate a database to an ADM:

1. From the Application Data Models page, select an ADM, select **Actions**, then **Associated Databases**.

This dialog lists all of the databases associated with this ADM and the schemas assigned to each application per database. You can add more databases that give you a choice of data sources when subsetting and databases to mask during masking.

2. Click **Add**, then select a database from the pop-up.

The selected database now appears in the Database section of the Associated Databases dialog.

3. To change a schema, select the associated database on the left, select the application on the right for which the schema is to be changed, then click **Select Schema**.

4. Select the missing schema from the list in the pop-up, then click **Select**.

Importing and Exporting an Application Data Model

You can share Application Data Models (ADM) with other Enterprise Manager environments that use a different repository by exporting an ADM, which can subsequently be imported into the new repository.

An exported ADM is by definition in the XML file format required for import. You can edit an exported ADM XML file prior to import. When exporting an ADM for

subsequent import, it is best to have one that uses most or all of the features—applications, tables, table types, referential relationships, sensitive columns. This way, if you are going to edit the exported file prior to import, it is clear which XML tags are required and where they belong in the file.

- [Importing an ADM](#)
- [Exporting an ADM](#)

Importing an ADM

There are two methods of import:

- Import an ADM XML file from the desktop
- Import an ADM XML file from the Software Library

To import an ADM XML file from your desktop:

1. From the **Actions** menu, select **Import**, then select **File from Desktop**.
2. In the pop-up that appears, specify a name for the ADM, the source database you want to assign to the ADM, and location on your desktop from which you want to import the ADM.
3. Click **OK**.

The ADM now appears on the Data Discovery and Modeling page.

To import an ADM XML file from the Software Library:

1. From the **Actions** menu, select **Import**, then select **File from Software Library**.
2. In the Export File from Software Library pop-up that appears, select the desired ADM XML file on the left, then specify a name and the source database you want to assign to the ADM on the right.
3. Click **Import**.

The ADM now appears on the Data Discovery and Modeling page.

After importing an ADM, you may want to discover sensitive columns or run a verification job. In the process of performing these tasks, the PL/SQL metadata collection packages are automatically deployed on the target database. The Database user must have DBA privileges to auto-deploy the packages.

Exporting an ADM

There are three methods of export:

- Export a selected ADM to the desktop
- Export an ADM from the Software Library
- Export an ADM to a TSDP Catalog

To export an ADM as an XML file to your desktop:

1. From the Data Discovery and Modeling page, select the ADM you want to export.
2. From the **Actions** menu, select **Export**, then select **Selected Application Data Model**.
3. In the File Download pop-up that appears, click **Save**.
4. In the Save As pop-up that appears, navigate to a file location and click **Save**.

The system converts the ADM into an XML file that now appears at the specified location on your desktop.

To export an ADM from the Software Library:

1. From the **Actions** menu, select **Export**, then select **File from Software Library**.
2. In the Export File from Software Library pop-up that appears, select the desired ADM and click **Export**.
3. In the File Download pop-up that appears, click **Save**.
4. In the Save As pop-up that appears, navigate to a file location and click **Save**.

The system converts the ADM into an XML file that now appears at the specified location on your desktop.

To export an ADM to a Transparent Sensitive Data Protection (TSDP) Catalog:

1. From the Data Discovery and Modeling page, select the ADM you want to export.
2. From the **Actions** menu, select **Export**, then select **Export to TSDP Catalog**.
3. The Data Discovery and Modeling page displays a table of associated databases. Select a database and click the **Export Sensitive Data** button.
4. In the Export Sensitive Data pop-up that appears, provide credentials for the selected database and click **OK**.

A message appears on the Data Discovery and Modeling page confirming that the sensitive data was copied to the database.

For detailed information on TSDP, see *Oracle Database Security Guide*.

Verifying or Upgrading a Source Database

After you have created an Application Data Model (ADM), the Source Database Status column can indicate Valid, Invalid, Needs Verification, or Needs Upgrade.

- **Invalid status**—Verify the source database to update the referential relationships in the application data model with those found in the data dictionary, and to also determine if each item in the application data model has a corresponding object in the database.
- **Needs Verification status**—You have imported an Oracle supplied template and you must verify the ADM before you can use it. This is to ensure that necessary referential relationships from data dictionary are pulled into the ADM.
- **Needs Upgrade status**—You have imported a pre-12c masking definition, so you now need to upgrade the ADM.

To verify a source database:

1. Select the ADM to be verified, indicated with an Invalid status.
2. From the **Actions** menu, select **Verify**.
3. Select the source database with the Invalid status, then click **Create Verification Job**.
4. Specify job parameters in the Create Verification Job pop-up, then click **Submit**.
5. After the job completes successfully, click the source database and note the object problems listed.

6. Fix the object problems, rerun the Verification Job, then check that the Source Database Status is now Valid.

To upgrade an ADM:

1. Select the ADM to be upgraded, indicated with a Needs Upgrade status.
2. From the Actions menu, select **Upgrade**.
3. Specify job parameters in the Create Upgrade Job pop-up, then click **Submit**.
4. After the job completes successfully, check that the Source Database Status column now indicates Valid. If the column indicates Invalid, see the previous procedure.

Using Self Update to Download the Latest Data Masking and Test Data Management Templates

Use the Self Update feature to get the latest data masking and data subsetting templates available from Oracle, out of band of the next major release cycle. With the auto-download feature enabled, new templates appear in the Software Library as they become available. Otherwise, you can access them manually as follows:

1. From the **Setup** menu, select **Extensibility**, then select **Self Update**.
2. On the Self Update page, scroll down and select Test Data Management templates.
3. In the available updates table, select the templates you want to download and select **Download** from the **Actions** menu.

This action downloads the templates to the Software Library from where you can import them into your Data Masking and Test Data Management environment or save them locally for editing.

4. To save a downloaded template:
 - a. Navigate to the appropriate home page (ADM, masking, or subset).
 - b. From the **Actions** menu, select **Export from the Software Library**.
 - c. Select a template in the pop-up window and click **Save**.
 - d. Specify a location where to save the XML file.

The template file is available for editing prior to being imported into Data Discovery and Modeling.

5. To import a downloaded template:
 - a. Navigate to the appropriate home page (ADM, masking, or subset).
 - b. From the **Actions** menu, select **Import from the Software Library**.
 - c. Select a template in the pop-up window and specify appropriate values for the input parameters.
 - d. Click **Import**. Template definitions are imported into the respective tables.

Data Subsetting

This chapter introduces the concept of inline masking and subsetting ([About Inline Masking and Subsetting](#)), and outlines a number of [Inline Masking and Subsetting Scenarios](#). The chapter also provides procedures for the following tasks:

- [Creating a Data Subset Definition](#)
- [Importing and Exporting Subset Templates and Dumps](#)
- [Creating a Subset Version of a Target Database](#)

Note: Data subsetting is supported only in Oracle Database versions 10.1 or higher. The procedures in this chapter are applicable only to Oracle Enterprise Manager Cloud Control 12.1 or higher.

About Inline Masking and Subsetting

You can reduce the size of the database simultaneous with masking sensitive data. This serves the dual purpose of obscuring exported production data while greatly reducing hardware costs related to storing large masked production databases for testing.

Note: Inline masking is available only with Oracle Database 11g or higher releases.

The benefits of integrating data masking with subsetting include the following:

- Prepare the test system in a single flow
- Avoid the necessity of maintaining large-size masked databases for test purposes
- Exported data in the form of a dump file can be imported into multiple databases without exposing sensitive data
- Subsetting is enhanced by ability to discard columns containing chunks of large data

You can select one or more data masking definitions during subset creation. The masking definitions must be based on the same ADM as the current subset definition. At the same time, you can significantly reduce the subset size by defining column rules to set CLOB and BLOB columns to null (or another supported format such as Fixed String, Fixed Number).

You generate a subset in two ways:

- **Export Dump**—if masking definitions are part of the subset model, mapping tables are created during generation, and the resulting dump contains masked values
- **In Place Delete**—subsetting is performed on a cloned copy of the production database; if data masking is part of the subset model, pregenerated masking scripts are executed on the target sequentially

Advantages of inline masking include the following:

- Sensitive data never leaves the production environment and thus is not exposed (Export Dump option).
- There is no need to temporarily store data in a staging area.
- Exported data can subsequently be imported into multiple environments.
- You can define table rules to export only a subset of data, and can further trim the volume by using column rules to eliminate large vertical columns.
- You can mask the same data in different ways and import into different test databases.
- You can use the provisioning framework to create multiple copies of trimmed down, referentially intact databases containing no sensitive data (in place delete), or import a dump file into multiple databases (export dump).

The section "[Creating a Data Subset Definition](#)" includes instructions for combining data subsetting and data masking within the process of creating a subset definition. See [Chapter 18, "Masking Sensitive Data,"](#) for information on data masking and creating a data masking definition.

Creating a Data Subset Definition

The procedure described in this section enables you to create a subset database, after which you can perform other tasks, such as editing the properties of the subset definition or exporting a subset definition.

The interface also allows you to perform inline, or at the source, masking while creating the subset definition. To use this feature to the fullest, you must have licenses for both of the following packs:

- Test Data Management Pack
- Data Masking Pack

If you have one or the other license, certain restrictions apply, as indicated within the procedure. For information on separately licensed Oracle Enterprise Manager management packs, management plug-ins, and other products, see *Oracle Enterprise Manager Licensing Information*.

Before proceeding, ensure that you have the following privileges:

- EM_ALL_OPERATOR for Enterprise Manager Cloud Control users
- SELECT_CATALOG_ROLE for database users
- SELECT_ANY_DICTIONARY privilege for database users

To create a data subset definition:

1. From the Enterprise menu, select **Quality Management**, then **Data Subset Definitions**.
2. Open the **Actions** menu in the Data Subset Definitions page, then select **Create**, or just click the **Create** icon.

3. Define the data subset definition properties:

- a. Provide the requisite information in the General pop-up that appears, then click **Continue**.

You can select any source database associated with the Application Data Model.

If you are performing masking within the subset definition, you must select the same ADM and target used in creating the masking definition.

- b. Provide a job name, credentials, and specify a schedule in the Schedule Application Detail Collection pop-up that appears, then click **Submit**.

If you want to use new credentials, choose the New Credentials option. Otherwise, choose the Preferred Credentials or Named Credentials option.

The space estimate collection job runs, and then displays the Data Subset Definitions page. Your definition appears in the table, and the Most Recent Job Status column should indicate Scheduled, Running, or Succeeded, depending on the schedule option selected and time required to complete the job.

4. Select the definition within the table, open the **Actions** menu, then select **Edit**.

The Database Login page appears.

5. Select either Named Credentials or New Credentials if you have not already set preferred credentials, then click **Login**.

6. In the Applications subpage of the Edit page, move applications from the Available list to the Selected list as follows:

- If you are licensed for data masking and intend only to mask the data (no subsetting), select all applications.
- If you are licensed for test data management and intend only to subset the data (no masking), select specific applications as appropriate.
- If you are licensed for test data management and intend both to subset and mask the data, the applications selected must include those that the masking definitions require.

The names of application suites, applications, or application modules are maintained in the Application Data Model.

7. Click the **Table Rules** tab.

Note: If you are licensed for masking only, set the Default Table Rules option to include all rows and skip to step 12. The **Column Rules** tab, **Rule Parameters** tab, and additional features on the **Table Rules** tab are available only to test data management licensees.

You can add rules here to define the data to include in the subset.

8. Select **Actions**, then **Create** to display the Table Rule pop-up, or just click the **Create** icon.

- a. Select the application for which you want to provide a rule.

Associate the rule with all tables, a specific table, or a category of tables.

- b. In the Rows to Include section, select the option that best suits your needs for a representative sample of production data. If you do not want to include all

rows, you can include some rows by specifying a percentage portion of the rows. For finer granularity, you could specify a Where clause, such as where region_id=6.

For more information on specifying Where clauses, see step e.

- c. In the Include Related Rows section, do one of the following:

- Select **Ancestor and Descendant Tables**

This rule impacts the parent and child columns, and ensures that referential integrity is maintained, and that child columns are also selected as part of the subset.

- Select **Ancestor Tables Only**

This rule only impacts the parent columns, and ensures that referential integrity is maintained.

If you disable the Include Related Rows check box, referential integrity may not be maintained. However, you can subsequently provide additional rules to restore referential integrity. You can disable this check box whether or not you specify a Where clause.

- d. If you want to specify a Where clause, go to the next step. Otherwise, skip to step 9.

- e. Provide a rule parameter, if desired, for the clause.

For instance, if you specify a particular value for an employee ID as employee_id=:emp_id, you could enter query values for the default of 100:

- Select the Rows Where button and enter employee_id=:emp_id.
- Click **OK** to save the rule and return to the Table Rules tab.

If this is a new rule, a warning appears stating that "Rule parameters corresponding to the bind variables 'emp_id' should be created before generating subset."

- Select the table rule, click the **Rule Parameters** tab, then click **Create**.

The Rule Parameter Properties pop-up appears.

- Enter emp_id for the Name and 100 for the Value.

Note: The colon (:) preceding emp_id is only present in the Where clause, and not required when creating a new rule parameter.

- Click **OK** to save the properties, which now appear in the **Rule Parameters** tab.
- Skip to step 10.

9. Click **OK** to save the rule and return to the **Table Rules** tab.

The new rule is displayed in the list. The related tables are displayed in the table below. Related rows from the tables are included in the subset to provide referential integrity in the subset database.

10. In the Default Table Rows section of the **Table Rules** tab, choose whether you want to include or exclude the tables not affected by the defined rules in the subset.

When you select the Include All Rows option, all of the rows for the table are selected as part of the subset.

This is a global rule and applies to the entire subset. You can only select the Include All Rows option when all of the rules have a scope of None. A scope of None is established when you uncheck the Include Related Rows option in the Table Rule pop-up.

Note: For a subset definition that has column rules (see step 11), be sure to use table rules to include the corresponding tables. You can use the Default Table Rules option to include all tables not affected by table rules, if required.

11. *Optional:* Click the **Column Rules** tab to perform inline masking as part of the subset definition.

Note: You must be a test data management licensee to use the features on this tab.

- a. Click **Create** and enter search criteria to filter on columns within the schema. These would typically be vertical columns such as CLOB AND BLOB columns.

Note: If you are using column rules instead of masking definitions (see step 12), you can select no more than 10 columns in a given table. This restriction applies to the export method but not to the in-place delete method.

Click **OK**.

- b. Select a row or rows in the column search results and click **Manage Masking Formats**.
- c. In the pop-up dialog, select a masking format and value to apply to the columns. For multiselection, the same format must be appropriate for all columns. If you select multiple columns, ensure that the column rule format you choose is applicable to the selected columns. Use the columns (flags) **not null** and **unique** to enforce compliance.

Click **OK** to apply the masking format to the columns.

12. *Optional:* Click the **Data Masking** tab to include masking definitions as part of the subsetting operation or to perform at the source data masking only.

- a. Click **Add**.
- b. In the pop-up dialog, enter search criteria to retrieve appropriate definitions. Be sure to select the desired radio button (All or Any). All formats except compound masking and SQL expressions are supported for inline masking.

Note: No single table within a masking definition can have more than 10 masked columns if you are using the export method. The restriction does not apply to the in-place delete method.

Click **OK**.

The search results appear in the data masking table.

13. Click the *Space Estimates* tab.

- Note the value in the Estimated Subset Size GB column. The space estimates depend on optimizer statistics, and the actual distribution of data can only be calculated if histogram statistics are present.
- Whenever you add new rules, recheck the space estimates for updated values.
- Data in the Space Estimates subpage is sorted with the largest applications appearing at the top.

Note: Space estimates do not reflect the effect of data masking, if used.

If you provide a Where clause and subsequent rule parameter properties, the Space Estimates subpage is updated with the value contained in the **Rule Parameters** tab.

14. *Optional:* click the *Pre/Post Subset Scripts* tab.

- You can specify a pre-subset script to run on the subset database before you select subset data.
- You can specify a post-subset script to run on the subset database after you assemble the subset data.
- Either script type runs on the source database.

15. Click *Return*.

The definition is complete and displayed in the Data Subset Definitions table.

You can now proceed with script generation. Alternatively, you may want to save the script for future use. In either case, you must decide whether to export data to a dump file or delete data from a target database.

Tip: If you have a very large database of 4 terabytes, for instance, and you want to export a small percentage of the rows, such as 10%, it is more advantageous to use the export method. Using the in-place delete method would require 3.6 terabytes of data, which would not perform as quickly as the export method.

The in-place delete method is recommended when the amount of data being deleted is a small percentage of the overall data size.

- [Generating a Subset Script](#)
- [Saving a Subset Script](#)

Generating a Subset Script

To prepare and submit a job to generate a subset script:

1. Select the definition within the table, open the **Actions** menu, then select **Generate Subset**. The Subset Mode pop-up appears.

2. Select a target database that is either the same target database you used to create the subset model, or similar to this database regarding the table schema and objects.
3. Decide if you want to create a subset by writing subset data to export files, or by deleting data from a target database.

Choosing to delete data creates an in-place subset by removing/deleting unwanted data from a cloned copy of the production database, rather than a production database. Only data satisfying the rules are retained. You should never use this option on a production database.

Select either Named Credentials or New Credentials if you have not already set preferred credentials.

If you have defined any parameters from the **Rule Parameters** tab, they appear in the table at the bottom. You can change a parameter value by clicking on the associated field in the Value column.

4. Click **Continue** to access the Parameters pop-up. The contents of the pop-up depend on whether you chose the export or delete option in the previous step.

For **Writing Subset Data to Export Files**, provide the requisite information, then click **Continue** to schedule the job.

- Specify a subset directory where to save the export dump. The drop-down list consists of directory objects for which you have access. Alternatively, you can select a custom directory path. Click the check box if you want to speed the process by using an external directory. Recommended default: DATA_PUMP_DIR.
- Specify appropriate values if you want to override the defaults: enter a name for the export file; specify a maximum file size in megabytes; specify the maximum number of threads of active execution operating on behalf of the export job. This enables you to consider trade-offs between resource consumption and elapsed time.
- Select whether to enable dump file compression and encryption. Enter and confirm an encryption password, if appropriate. Log file generation is selected by default.

For **Deleting Data From a Target Database**, provide the requisite information, then click **Continue** to schedule the job.

- Specify a subset directory where to save the subset scripts. The drop-down list consists of directory objects for which you have access. Alternatively, you can select a custom directory path. Recommended default: DATA_FILE_DIR.
 - You must enable the check box indicating that the selected target is not a production database in order to proceed.
5. Click **Continue** to schedule the job from the Generate Subset Schedule pop-up, then click **Submit**. For the delete option, you must specify and confirm an encryption seed.

The Data Subset Definitions page reappears, and the Most Recent Job Status column shows that the subset job is running, and subsequently that it has succeeded.

After performing this procedure, you can now create a subset database with the generated export files at any time.

Saving a Subset Script

To prepare and submit a job to save a subset script:

1. Select the definition within the table, open the **Actions** menu, then select **Save Subset Script**. The Subset Mode pop-up appears.
2. Select a target database that is either the same target database you used to create the subset model, or similar to this database regarding the table schema and objects.
3. Decide if you want to create a subset by writing subset data to export files, or by deleting data from a target database.

Choosing to delete data creates an in-place subset by removing/deleting unwanted data from a cloned copy of the production database, rather than a production database. Only data satisfying the rules are retained. You should never use this option on a production database.

Select either Named Credentials or New Credentials if you have not already set preferred credentials.

If you have defined any parameters from the **Rule Parameters** tab, they appear in the table at the bottom. You can change a parameter value by clicking on the associated field in the Value column.

4. Click **Continue** to access the Parameters pop-up. The contents of the pop-up depend on whether you chose the export or delete option in the previous step.

For **Writing Subset Data to Export Files**, provide the requisite information, then click **Continue** to schedule the job.

- Specify a subset directory where to save the export dump. The drop-down list consists of directory objects for which you have access. Alternatively, you can select a custom directory path. Click the check box if you want to speed the process by using an external directory. Recommended default: DATA_PUMP_DIR.
- Specify appropriate values if you want to override the defaults: enter a name for the export file; specify a maximum file size in megabytes; specify the maximum number of threads of active execution operating on behalf of the export job. This enables you to consider trade-offs between resource consumption and elapsed time.
- Select whether to enable dump file compression and encryption. Enter and confirm an encryption password, if appropriate. Log file generation is selected by default.

For **Deleting Data From a Target Database**, provide the requisite information, then click **Continue** to schedule the job.

- Specify a subset directory where to save the subset scripts. The drop-down list consists of directory objects for which you have access. Alternatively, you can select a custom directory path. Recommended default: DATA_FILE_DIR.
 - You must enable the check box indicating that the selected target is not a production database in order to proceed.
5. Click **Continue**. A progress indicator tracks script generation. When complete, the Files table lists the results of script generation.
 6. Click **Download**. In the File Download pop-up that appears, click **Save**.
 7. In the Save As pop-up that appears, navigate to a file location and click **Save**.

The file containing the scripts (SubsetBundle.zip) now appears at the specified location on your desktop.

To run the saved script at a later time:

1. Port the ZIP file to the target database and extract it to a directory on which you have the requisite privileges.
2. Change directory to where you extracted the files.
3. Execute the following script from the SQL command line:

```
subset_exec.sql
```

Note that if you want to change generated parameter settings, you can do so by editing the following file in a text editor prior to executing the script:

```
subset_exec_params.lst
```

Importing and Exporting Subset Templates and Dumps

A subset template is an XML file that contains the details of the subset, consisting of the application, subset rules, rule parameters, and pre-scripts or post-scripts. When you create a subset definition and specify that you want to write subset data to export files, the export files become a template that you can subsequently import for reuse. You would import the template to perform subset operations on a different database.

Typically, the workflow is that you would first import a previously exported ADM template, which is another XML file, while creating an ADM. You would then import the related subset template while creating a data subset definition. You could alternatively select an existing ADM (skipping the import ADM flow) while importing the subset template.

Tip: Oracle also provides a set of ADM and subset templates that you can download. The ADM and subset templates provide the comprehensive subset definitions for packaged applications, such as Oracle E-Business Suite and Oracle Fusion Applications.

- [Importing a Subset Definition](#)
- [Exporting a Subset Definition](#)

Importing a Subset Definition

There are three methods of import:

- Import a subset definition XML file from the desktop
- Import a subset dump
- Import a subset definition XML file from the Software Library

To import a subset definition XML file from your desktop:

1. From the **Actions** menu, select **Import**, then select **File from Desktop**.
2. In the pop-up that appears:
 - Specify a name for the subset definition
 - The ADM on which the subset is based
 - A source database

- The location on your desktop from which you want to import the subset definition
 - Click **Continue**
3. In the pop-up that appears:
 - Enter a descriptive job name (or accept the default)
 - Provide credentials
 - Schedule the job
 - Click **Submit**
- After the job runs successfully, the imported subset appears in the list of subsets in the table on the Data Subset Definitions page.

To import a subset dump:

1. From the **Actions** menu, select **Import**, then select **Subset Dump**.
2. In the pop-up that appears:
 - Select a target database
 - Provide database and host credentials, then click **Login**.
 - Specify the location of the dump file, which can be in a selected directory on the target database or at a custom path on the target. Note that if the original export action used an external location for the dump files, the location must be specified as well.
 - Click **Continue**.
3. In the pop-up that appears:
 - Select whether to import both metadata and data, or data only. If data only, indicate if you want to truncate, that is, overlay existing data or append to existing data.
 - Perform tablespace remapping as necessary
 - Perform schema remapping as necessary
 - Select log file options
 - Click **Continue**
4. In the pop-up that appears:
 - Enter a descriptive job name (or accept the default)
 - Schedule the job
 - Click **Submit**

The job reads the dump files and loads the data into the selected target database.

To import a subset definition XML file from the Software Library:

1. From the **Actions** menu, select **Import**, then select **File from Software Library**.
2. In the Import Data Subset Definition from Software Library pop-up that appears:
 - Selected the desired subset definition XML file on the left
 - Provide the ADM properties on the right.
 - Click **Continue**

3. In the pop-up that appears:
 - Enter a descriptive job name (or accept the default)
 - Provide credentials
 - Schedule the job
 - Click **Submit**

After the job runs successfully, the imported subset appears in the list of subsets in the table on the Data Subset Definitions page.

Exporting a Subset Definition

There are two methods of export:

- Export a selected subset definition to the desktop
- Export a subset definition from the Software Library

To export a subset definition as an XML file to your desktop:

1. From the Data Subset Definitions page, select the subset definition you want to export.
2. From the **Actions** menu, select **Export**, then select **Selected Subset Definition**.
3. In the File Download pop-up that appears, click **Save**.
4. In the Save As pop-up that appears, navigate to a file location and click **Save**.

The system converts the subset definition into an XML file that now appears at the specified location on your desktop.

To export a subset definition from the Software Library:

1. From the **Actions** menu, select **Export**, then select **File from Software Library**.
2. In the Export Subset Definition from Software Library pop-up that appears, select the desired subset definition and click **Export**.
3. In the File Download pop-up that appears, click **Save**.
4. In the Save As pop-up that appears, navigate to a file location and click **Save**.

The system converts the subset definition into an XML file that now appears at the specified location on your desktop.

After the job runs successfully, the subset template appears in the list of subsets in the table on the Data Subset Definitions page.

Creating a Subset Version of a Target Database

After a subset is defined, analyzed, and validated, you can execute the subset operation to create a subset version of the source data.

The procedure assumes the following prerequisites:

- A subset definition already exists that contains the rules needed to subset the database.
- You have the requisite privileges to extract the data from the source and create the subset version in a target database. Depending on the subset technique, different levels of file or database privileges may be created. The required privileges include:

- EM_ALL_OPERATOR for Enterprise Manager Cloud Control users
- SELECT_CATALOG_ROLE for database users
- SELECT ANY DICTIONARY privilege for database users
- DBA privileges on a database for target database users

To create a subset version of a target database:

1. Create a subset operation by selecting a subset definition and associating it with a source database.

Enterprise Manager validates the subset definition against the source database and flags schema differences. Note that this association may be different from the original association that an application developer may have created.

2. Edit the definition to remap the defined schema to a test schema.

You are prompted to connect to a database, whereupon the database is associated with the subset definition. This also enables you to remap the vendor-provided schema names to actual schema names in the database.

3. Select one of the various subset creation techniques:

- Data Pump dump file followed by a Data Pump import
- In-place delete, in which rows in the specified database not matching the rule conditions are deleted
- In-transit subset creation or refresh

Enterprise Manager generates the appropriate response file (that is, SQL script, Data Pump script, or OS script), checks the target system for appropriate privileges to be able proceed with the operation, and estimates the size of the target.

4. After reviewing the analysis, submit the subset process.

Enterprise Manager executes the subset process and summarizes the results of the execution.

Inline Masking and Subsetting Scenarios

The scenarios described below assume that an Application Data Model (ADM) exists for a production (or test) database in which sensitive column details are captured. The steps outlined are at a high level. See ["Masking with an Application Data Model and Workloads"](#) on page 18-15 for details on creating a masking definition; see ["Creating a Data Subset Definition"](#) on page 17-2 for details on creating and editing a subset definition.

Consider the following scenarios:

- [Mask and Export Production Data](#)
- [Mask and Delete Operation on a Test Database](#)
- [Mask Sensitive Data and Export a Subset of a Production Database](#)
- [Perform Subset, Mask, and Delete Operations on a Test Database](#)
- [Apply Column Rules](#)
- [Export a Subset Definition That Uses Inline Masking](#)
- [Import a Subset Definition That Uses Inline Masking](#)

- [Import a Subset Dump](#)
- [Save Subset Script Bundle](#)

Mask and Export Production Data

As the Security Administrator, you want to create copies of the production database by exporting the data with masked values; that is, the export dump will have only masked values and no sensitive data.

1. Create a masking definition. Implies the following:
 - a. Select an appropriate ADM.
 - b. Search and select sensitive columns (includes dependent columns and recommended masking formats).
 - c. Review suggested formats and edit as necessary.
 - d. Save the results.
2. Create a subset definition. Implies the following:
 - a. Select an appropriate ADM.
 - b. Submit the create subset job.
3. Edit the subset definition.

On the **Data Masking** tab, search for and select masking definitions. System validation checks for overlapping columns that use multiple masking definitions.
4. Generate the subset using the Export option.

Summarizing the outcome:

- Generates and executes a script to create a mapping table and a mapping function. Also creates a table to map the column(s) to the respective mapping function.
- Copies subsetting and masking scripts to the target database.
- Generates an export dump of production data, replacing sensitive data with masked values using the mapping function.

Mask and Delete Operation on a Test Database

As the Security Administrator, you want to create a usable test database by masking sensitive information. The resulting database will have only masked values and no sensitive data.

1. Create a masking definition on a cloned database. Implies the following:
 - a. Select an appropriate ADM.
 - b. Search and select sensitive columns (includes dependent columns and recommended masking formats).
 - c. Review suggested formats and edit as necessary.
 - d. Save.
2. Create a subset definition. Implies the following:
 - a. Select an appropriate ADM.
 - b. Submit the create subset job.
3. Edit the subset definition.

On the **Data Masking** tab, search and select masking definitions. System validation checks for overlapping columns that use multiple masking definitions.

4. Generate the subset using the In Place Delete option.

Summarizing the outcome:

- Copies subsetting and masking scripts to the target database.
- Performs data subsetting based on subset rules, if specified.
- Sequentially executes the pregenerated data masking scripts on the target database.
- Creates a masked copy of the production database for use in testing.

Mask Sensitive Data and Export a Subset of a Production Database

As the Security Administrator, you want to create copies of the production database by exporting a subset of production data with masked values.

1. Create a masking definition. Implies the following:
 - a. Select an appropriate ADM.
 - b. Search and select sensitive columns (includes dependent columns and recommended masking formats).
 - c. Review suggested formats and edit as necessary.
 - d. Save.
2. Create a subset definition. Implies the following:
 - a. Select an appropriate ADM.
 - b. Submit the create subset job.
3. Edit the subset definition.
 - a. Define table rules, resulting in space estimates.
 - b. On the **Data Masking** tab, search and select masking definitions. System validation checks for overlapping columns that use multiple masking definitions.
4. Generate the subset using the Export option.

Summarizing the outcome:

- Generates and executes a script to create a mapping table and a mapping function. Also creates a table to map the column(s) to the respective mapping function.
- Copies subsetting and masking scripts to the target database.
- Generates an export dump of production data, replacing sensitive data with masked values using the mapping function.

Perform Subset, Mask, and Delete Operations on a Test Database

As the Security Administrator, you want to create a usable test database by masking sensitive information. On import, the database will have only masked values and no sensitive data.

1. Create a masking definition. Implies the following:
 - a. Select an appropriate ADM.

- b. Search and select sensitive columns (includes dependent columns and recommended masking formats).
 - c. Review suggested formats and edit as necessary.
 - d. Save.
2. Create a subset definition. Implies the following:
 - a. Select an appropriate ADM.
 - b. Submit the create subset job.
3. Edit the subset definition.
 - a. Define table rules, resulting in space estimates.
 - b. On the **Data Masking** tab, search and select masking definitions. System validation checks for overlapping columns that use multiple masking definitions.
4. Generate the subset using the In Place Delete option.

Summarizing the outcome:

- Copies subsetting and masking scripts to the target database.
- Performs data subsetting based on subset rules, if specified.
- Following subset completion, sequentially executes the pregenerated data masking scripts on the target database.
- Applies masking definitions and subsetting rules, resulting in a masked database of reduced size.

Apply Column Rules

As the Security Administrator, you want to create a targeted subset by selecting large-sized columns and setting them to null or a fixed value. Table rules can also be used to further reduce database size. Impact of size reduction is immediately visible and applied to the final subset.

1. Create a subset definition. Implies the following:
 - a. Select an appropriate ADM.
 - b. Submit the create subset job.
2. Edit the subset definition.
 - a. Click the **Table Rules** tab and select from existing options, if desired.
 - b. Click the **Column Rules** tab, then click **Create**.
 - c. Specify filtering criteria to search for large-sized columns and select the desired columns in the results table.
 - d. Click **Manage Masking Formats** and select a format from the drop-down list. Enter a value if appropriate to the selection.
 - e. Click **OK** and review the updated space estimates.
3. Generate the subset, using either the Export or In Place Delete option.

Summarizing the outcome:

- Generates an export dump/subset of production data.
- Column rules are applied on the target database.

- If table rules were also applied, the resulting subset reflects the combined effect of table and column rules.

Export a Subset Definition That Uses Inline Masking

As the Security Administrator, you want to export a subset definition for reuse.

1. Create a subset definition. Implies the following:
 - a. Select an appropriate ADM.
 - b. Submit the create subset job.
2. Edit the subset definition.
 - a. Create rules to compute space estimates.
 - b. On the **Data Masking** tab, search and select masking definitions. System validation checks for overlapping columns that use multiple masking definitions.
3. Select the subset definition on the Subset home page and export it.

The subset definition is saved on the client machine as an XML file that potentially contains the following:

- Information on selected applications
- Rules and rule parameters
- Selected masking definitions
- Columns to set to null
- Pre- and post-scripts

Had column rules been used, they would replace the masking definitions in the list.

Import a Subset Definition That Uses Inline Masking

As the Security Administrator, you want to import a subset definition XML file to create replicas of the subset definition previously exported.

1. Import a subset definition.
2. Select an exported XML template that contains exported masking definitions. System validation:
 - Checks for overlapping columns that use multiple masking definitions.
 - Ensures that the masking definition specified is part of the same ADM as the current subset model.
3. Submit the job to create the subset model.

Summarizing the outcome:

- Creates a subset definition model
- Applies specified rules and calculates space estimates
- Remembers masking definitions that were part of the XML

Import a Subset Dump

As the Security Administrator, you want to import a subset dump, which might contain either or both of the following:

- A masked version of a production database

- A subset version of a production database

Note that this example assumes a previous export dump.

1. On the subset home page, select **Import Subset Dump** from the **Actions** menu.
2. Provide credentials, a dump name, and select the dump location.
3. Provide the import type, tablespace options, and log file location details.
4. Schedule the job and submit.

The job reads the dump files and loads the data into the selected target database.

Save Subset Script Bundle

As the Security Administrator, you want to save a subset script bundle so that it can be executed on a target database external to Enterprise Manager.

This example presupposes the existence of a subset model that has required table rules and masking definitions.

1. On the subset home page, from the **Actions** menu, select **Generate**, then select **Subset**.
2. Complete the mode page as follows:
 - a. Indicate the method of subset creation.
 - b. Specify which credentials to use.
 - c. Provide rule parameters as appropriate.
 - d. Click **Continue**.
3. Complete the parameters page as follows:
 - a. Select the location where to save the subset export.
 - b. If the subset is to be stored externally, click the check box and select the location.
 - c. Specify an export file name. Note that you can use the % wildcard.
 - d. Specify the maximum file size and number of threads.
 - e. Indicate whether to generate a log file and specify a log file name.
 - f. Click **Continue**.
4. Note the progress of the script file generation. When complete, click **Download**.
5. Specify where to save the SubsetBundle.zip file.

Masking Sensitive Data

This chapter provides conceptual information about the components that comprise Oracle Data Masking, and procedural information about performing the task sequence, such as creating masking formats and masking definitions. Data masking presupposes that you have created an Application Data Model (ADM) with defined sensitive columns.

Topics discussed in this chapter include:

- [Overview of Oracle Data Masking](#)
- [Format Libraries and Masking Definitions](#)
- [Recommended Data Masking Workflow](#)
- [Data Masking Task Sequence](#)
- [Defining Masking Formats](#)
- [Masking with an Application Data Model and Workloads](#)
- [Masking a Test System to Evaluate Performance](#)
- [Upgrade Considerations](#)
- [Using the Shuffle Format](#)
- [Using Group Shuffle](#)
- [Using Conditional Masking](#)
- [Using Data Masking with LONG Columns](#)

Note: The procedures in this chapter are applicable to Oracle Enterprise Manager 12.1 or higher Cloud Control only.

Overview of Oracle Data Masking

Enterprises run the risk of breaching sensitive information when copying production data into non-production environments for the purposes of application development, testing, or data analysis. Oracle Data Masking helps reduce this risk by irreversibly replacing the original sensitive data with fictitious data so that production data can be shared safely with non-production users. Accessible through Oracle Enterprise Manager, Oracle Data Masking provides end-to-end secure automation for provisioning test databases from production in compliance with regulations.

Data Masking Concepts

Data masking (also known as data scrambling and data anonymization) is the process of replacing sensitive information copied from production databases to test non-production databases with realistic, but scrubbed, data based on masking rules. Data masking is ideal for virtually any situation when confidential or regulated data needs to be shared with non-production users. These users may include internal users such as application developers, or external business partners such as offshore testing companies, suppliers and customers. These non-production users need to access some of the original data, but do not need to see every column of every table, especially when the information is protected by government regulations.

Data masking enables organizations to generate realistic and fully functional data with similar characteristics as the original data to replace sensitive or confidential information. This contrasts with encryption or Virtual Private Database, which simply hides data, and the original data can be retrieved with the appropriate access or key. With data masking, the original sensitive data cannot be retrieved or accessed.

Names, addresses, phone numbers, and credit card details are examples of data that require protection of the information content from inappropriate visibility. Live production database environments contain valuable and confidential data—access to this information is tightly controlled. However, each production system usually has replicated development copies, and the controls on such test environments are less stringent. This greatly increases the risks that the data might be used inappropriately. Data masking can modify sensitive database records so that they remain usable, but do not contain confidential or personally identifiable information. Yet, the masked test data resembles the original in appearance to ensure the integrity of the application.

Security and Regulatory Compliance

Masked data is a sensible precaution from a business security standpoint, because masked test information can help prevent accidental data escapes. In many cases, masked data is a legal obligation. The Enterprise Manager Data Masking Pack can help organizations fulfill legal obligations and comply with global regulatory requirements, such as Sarbanes-Oxley, the California Database Security Breach Notification Act (CA Senate Bill 1386), and the European Union Data Protection Directive.

The legal requirements vary from country to country, but most countries now have regulations of some form to protect the confidentiality and integrity of personal consumer information. For example, in the United States, The Right to Financial Privacy Act of 1978 creates statutory Fourth Amendment protection for financial records, and a host of individual state laws require this. Similarly, the U.S. Health Insurance Portability and Accountability Act (HIPAA) created protection of personal medical information.

Roles of Data Masking Users

The following types of users participate in the data masking process for a typical enterprise:

- Application database administrator or application developer

This user is knowledgeable about the application and database objects. This user may add additional custom database objects or extensions to packaged applications, such as the Oracle E-Business suite.

- Information security administrator

This user defines information security policies, enforces security best practices, and also recommends the data to be hidden and protected.

Related Oracle Security Offerings

Besides data masking, Oracle offers the following security products:

- Virtual Private Database or Oracle Label Security — Hides rows and data depending on user access grants.
- Transparent Data Encryption — Hides information stored on disk using encryption. Clients see unencrypted information.
- DBMS_CRYPTO — Provides server packages that enable you to encrypt user data.
- Database Vault — Provides greater access controls on data.

Agent Compatibility for Data Masking

Data Masking supports Oracle Database 9i and newer releases. If you have a version prior to 11.1, you can use it by implementing the following work-around.

Replace the following file...

```
AGENT_HOME/sysman/admin/scripts/db/reorg/reorganize.pl
```

... with this file:

```
OMS_HOME/sysman/admin/scripts/db/reorg/reorganize.pl
```

Supported Data Types

The list of supported data types varies by release.

- Grid Control 10g Release 5 (10.2.0.5), Database 11g Release 2 (11.2), and Cloud Control 12c Release 1 (12.1.0.1) and Release 2 (12.1.0.2)

– Numeric Types

The following Numeric Types can use Array List, Delete, Fixed Number, Null Value, Post Processing Function, Preserve Original Data, Random Decimal Numbers, Random Numbers, Shuffle, SQL Expression, Substitute, Table Column, Truncate, Encrypt, and User Defined Function masking formats:

- * NUMBER
- * FLOAT
- * RAW
- * BINARY_FLOAT
- * BINARY_DOUBLE

– String Types

The following String Types can use Array List, Delete, Fixed Number, Fixed String, Null Value, Post Processing Function, Preserve Original Data, Random Decimal Numbers, Random Digits, Random Numbers, Random Strings, Shuffle, SQL Expression, Substitute, Substring, Table Column, Truncate, Encrypt, and User Defined Function masking formats:

- * CHAR
- * NCHAR

- * VARCHAR2
- * NVARCHAR2
- Date Types

The following Date Types can use Array List, Delete, Null Value, Post Processing Function, Preserve Original Data, Random Dates, Shuffle, SQL Expression, Substitute, Table Column, Truncate, Encrypt, and User Defined Function masking formats:

 - * DATE
 - * TIMESTAMP
- Grid Control 11g Release 1 (11.1) and Cloud Control 12c Release 1 (12.1.0.1) and Release 2 (12.1.0.2)
 - Large Object (LOB) Data Types

The following Data Types can use Fixed Number, Fixed String, and Null Value masking formats:

 - * BLOB
 - * CLOB
 - * NCLOB

Format Libraries and Masking Definitions

To mask data, the Data Masking Pack provides two main features:

- Masking format library

The format library contains a collection of ready-to-use masking formats. The library consists of format routines that you can use for masking. A masking format can either be one that you create, or one from the list of Oracle-supplied default masking formats.

As a matter of best practice, organizations should create masking formats for all commonly regulated information so that the formats can be applied to the sensitive data regardless of which database the sensitive data resides in. This ensures that all sensitive data is consistently masked across the entire organization.
- Masking definitions

A masking definition defines a data masking operation to be implemented on one or more tables in a database. Masking definitions associate table columns with formats to use for masking the data. They also maintain the relationship between columns that are not formally declared in the database using related columns.

You can create a new masking definition or use an existing definition for a masking operation. To create a masking definition, you specify the column of the table for which the data should be masked and the format of masked data. If the columns being masked are involved in unique, primary key, or foreign key constraints, data masking generates the values so that the constraints are not violated. Masking ensures uniqueness per character using decimal arithmetic. For example, a 5-character string generates a maximum of only 99999 unique values. Similarly, a 1-character string generates a maximum of only 9 unique values.

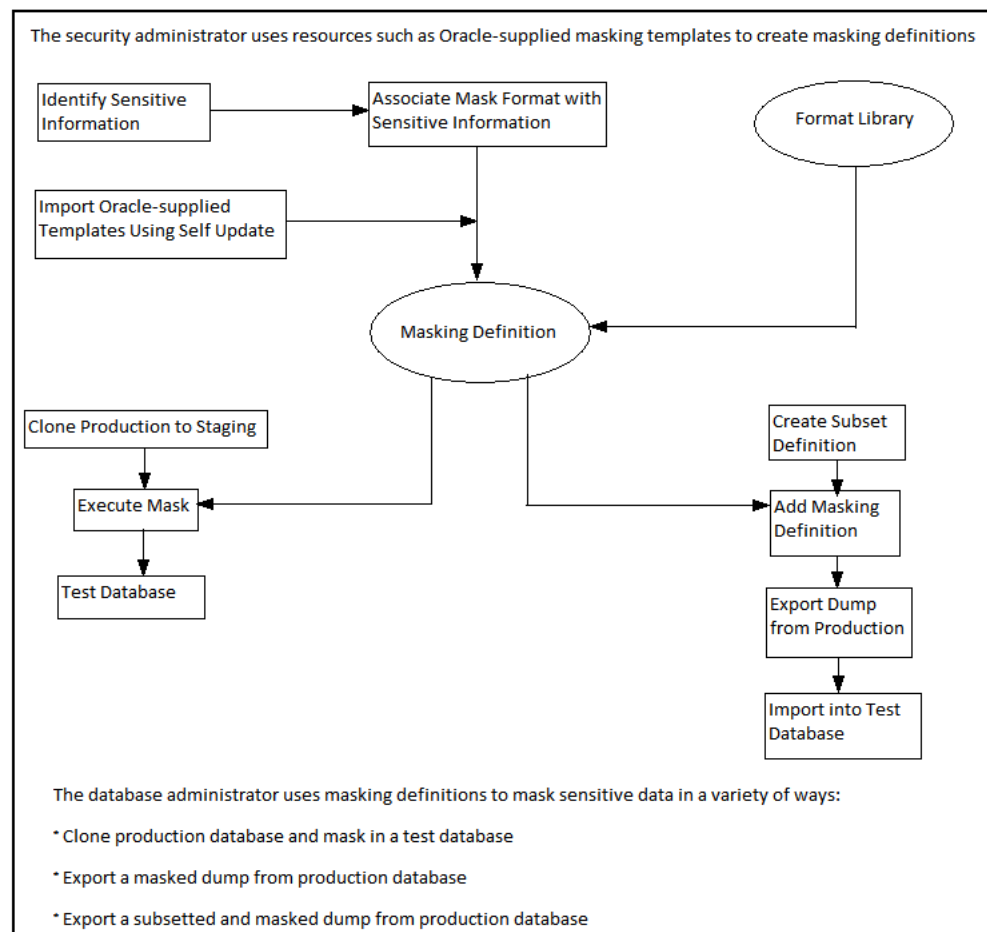
You would typically export masking definitions to files and import them on other systems. This is important when the test and production sites reside on different Oracle Management Systems or on entirely different sites.

See Also: The online help topic "Creating a Data Masking Definition" as well as the help for each Data Masking page

Recommended Data Masking Workflow

Figure 18–1 shows that the production database is cloned to a staging region and then masked there. During the masking process, the staging and test areas are tightly controlled like a production site.

Figure 18–1 Data Masking Workflow



Data masking is an iterative and evolving process handled by the security administrator and implemented by the database administrator. When you first configure data masking, try out the masking definition on a test system, then add a greater number of columns to the masking definition and test it to make sure it functions correctly and does not break any application constraints. During this process, you should exercise care when removing all imbedded references to the real data while maintaining referential integrity.

After data masking is configured to your satisfaction, you can use the existing definition to repeatedly mask after cloning. The masking definition, however, would need to evolve as new schema changes require new data and columns to be masked.

After the masking process is complete, you can distribute the database for wide availability. If you need to ship the database to another third-party site, you are required to use the Data Pump Export utility, and then ship the dump file to the remote site. However, if you are retaining the masked data in-house, see ["Data Masking Task Sequence"](#) on page 18-6.

You can also perform inline, or at the source, data masking while creating a subset definition. See ["Creating a Data Subset Definition"](#) on page 17-2 for more information.

Data Masking Task Sequence

The task sequence in this section demonstrates the data masking workflow and refers you to additional information about some of the tasks in the sequence. Before reviewing this sequence, note that there are two options for completing this process:

- Exporting/importing to another database
You can clone the production database to a staging area, mask it, then export/import it to another database before delivering it to in-house testers or external customers. This is the most secure approach.
- Making the staging area the new test region
You can clone the production database to a mask staging area, then make the staging area the new test region. In this case, you should not grant testers SYSDBA access or access to the database files. Doing so would compromise security. The masked database contains the original data in unused blocks and in the free list. You can only purge this information by exporting/importing the data to another database.

The following basic steps guide you through the data masking process, with references to other sections for supporting information.

1. Review the application database and identify the sources of sensitive information.
2. Define mask formats for the sensitive data. The mask formats may be simple or complex depending on the information security needs of the organization.
For more information, see ["Creating New Masking Formats"](#) on page 18-7 and ["Using Oracle-supplied Predefined Masking Formats"](#) on page 18-9.
3. Create a masking definition to associate table columns to these mask formats. Data masking determines the database foreign key relationships and adds foreign key columns to the mask.
For more information, see ["Masking with an Application Data Model and Workloads"](#) on page 18-15.
4. Save the masking definition and generate the masking script.
5. Verify if the masked data meets the information security requirements. Otherwise, refine the masking definition, restore the altered tables, and reapply the masking definition until the optimal set of masking definitions has been identified.
6. Clone the production database to a staging area, selecting the masking definition to be used after cloning. Note that you can clone using Enterprise Manager, which enables you to add masking to the Enterprise Manager clone workflow. However, if you clone outside of Enterprise Manager, you must initiate masking from

Enterprise Manager after cloning is complete. The cloned database should be controlled with the same privileges as the production system, because it still contains sensitive production data.

After cloning, be sure to change the passwords as well as update or disable any database links, streams, or references to external data sources. Back up the cloned database, or minimally the tables that contain masked data. This can help you restore the original data if the masking definition needs to be refined further.

For more information, see ["Cloning the Production Database"](#) on page 18-23.

7. After masking, test all of your applications, reports, and business processes to ensure they are functional. If everything is working, you can export the masking definition to keep it as a back-up.
8. After masking the staging site, make sure to drop any tables named `MGMT_DM_TT` before cloning to a test region. These temporary tables contain a mapping between the original sensitive column value and the mask values, and are therefore sensitive in nature.

During masking, Enterprise Manager automatically drops these temporary tables for you with the default "Drop temporary tables created during masking" option. However, you can preserve these temporary tables by deselecting this option. In this case, you are responsible for deleting the temporary tables before cloning to the test region.

9. After masking is complete, ensure that all tables loaded for use by the substitute column format or table column format are going to be dropped. These tables contain the mask values that table column or substitute formats will use. It is recommended that you purge this information for security reasons.

For more information, see ["Deterministic Masking Using the Substitute Format"](#) on page 18-15.

10. Clone the database to a test region, or use it as the new test region. When cloning the database to an external or unsecured site, you should use Export or Import. Only supply the data in the database, rather than the database files themselves.
11. As part of cloning production for testing, provide the masking definition to the application database administrator to use in masking the database.

Defining Masking Formats

A masking definition requires one or more masking formats for any columns included in the masking definition. When adding columns to a masking definition, you can either create masking formats manually or import them from the format library. It is often more efficient to work with masking formats from the format library.

Creating New Masking Formats

This section describes how to create new masking formats using Enterprise Manager.

To create a masking format in the format library:

1. From the Enterprise menu, select **Quality Management**, then **Data Masking Formats**. Alternatively, if you are in the Database home page, select **Data Masking Format Library** from the Schema menu.

The Format Library appears with predefined formats that Oracle Enterprise Manager provides.

2. Click **Create**.

The Create Format page appears, where you can define a masking format.

See Also: The online help for information on page user controls

3. Provide a required name for the new format, select a format entry type from the **Add** list, then click **Go**.

A page appears that enables you to provide input for the format entry you have selected. For instance, if you select Array List, the subsequent page enables you to enter a list of values, such as New York, New Jersey, and New Hampshire.

4. Continue adding additional format entries as needed.

5. When done, provide an optional user-defined or post-processing function (see ["Providing User-defined and Post-processing Functions"](#) on page 18-8), then click **OK** to save the masking format.

The Format Library page reappears with your newly created format displayed in the Format Library table. You can use this format later to mask a column of the same sensitive type.

See Also: The online help for information on the Format Library and Create Format pages

Providing User-defined and Post-processing Functions

If desired, you can provide user-defined and post-processing functions on the Create Format page. A user-defined choice is available in the Add list, and a post-processing function field is available at the bottom of the page.

- User-defined functions

To provide a user-defined function, select **User Defined Function** from the Add list, then click **Go** to access the input fields.

A user-defined function passes in the original value as input, and returns a mask value. The data type and uniqueness of the output values must be compatible with the original output values. Otherwise, a failure occurs when the job runs.

Combinable, a user-defined function is a PL/SQL function that can be invoked in a SELECT statement. Its signature is returned as:

```
Function udf_func (rowid varchar2, column_name varchar2, original_value  
varchar2) returns varchar2;
```

- rowid is the min (rowid) of the rows that contain the value original_value 3rd argument.
- column_name is the name of the column being masked.
- original_value is the value being masked.

That is, it accepts the original value as an input string, and returns the mask value.

Both input and output values are varchar2. For instance, a user-defined function to mask a number could receive 100 as input, the string representation of the number 100, and return 99, the string representation of the number 99. Values are cast appropriately when inserting to the table. If the value is not castable, masking fails.

- Post-processing functions

To provide a post-processing function, enter it in the **Post Processing Function** field.

A post-processing function has the same signature as a user-defined function, but passes in the mask value the masking engine generates, and returns the mask value that should be used for masking, as shown in the following example:

```
Function post_proc_udf_func (rowid varchar2, column_name varchar2, mask_value
varchar2) returns varchar2;
```

- rowid is the min (rowid) of the rows that contain the value mask_value.
- column_name is the name of the column being masked.
- mask_value is the value being masked.

Using Masking Format Templates

After you have created at least one format, you can use the format definition as a template in the Create Format page, where you can implement most of the format using a different name and changing the entries as needed, rather than needing to create a new format from scratch.

To create a new format similar to an existing format, select a format on the Format Library page and click **Create Like**. The masking format you select can either be one you have previously defined yourself, or one from the list of out-of-box masking formats. You can use these generic masking format definitions for different applications.

For instructional details about the various Oracle-supplied predefined masking format definitions and how to modify them to suit your needs, see ["Using Oracle-supplied Predefined Masking Formats"](#) on page 18-9.

Using Oracle-supplied Predefined Masking Formats

Enterprise Manager provides several out-of-box predefined formats. All predefined formats and built-in formats are random. The following sections discuss the various Oracle-supplied format definitions and how to modify them to suit your needs:

- [Patterns of Format Definitions](#)
- [Category Definitions](#)

See Also: ["Installing the DM_FMTLIB Package"](#) on page 11 for information on installing the DM_FMTLIB package so that you can use the predefined masking formats

Patterns of Format Definitions

All of the format definitions adhere to these typical patterns:

- Generate a random number or random digits.
- Perform post-processing on the above-generated value to ensure that the final result is a valid, realistic value.

For example, a valid credit card number must pass Luhn's check. That is, the last digit of any credit card number is a checksum digit, which is always computed. Also, the first few digits indicate the card type (MasterCard, Amex, Visa, and so forth). Consequently, the format definition of a credit card would be as follows:

- Generate random and unique 10-digit numbers.

- Using a post-processing function, transform the values above to a proper credit card number by adding well known card type prefixes and computing the last digit.

This format is capable of generating 10 billion unique credit card numbers.

Category Definitions

The following sections discuss different categories of these definitions:

- [Credit Card Numbers](#)
- [United States Social Security Numbers](#)
- [ISBN Numbers](#)
- [UPC Numbers](#)
- [Canadian Social Insurance Numbers](#)
- [North American Phone Numbers](#)
- [UK National Insurance Numbers](#)
- [Auto Mask](#)

By default, these mask formats are also available in different format styles, such as a hyphen (-) format. If needed, you can modify the format style.

Credit Card Numbers

Out of the box, the format library provides many different formats for credit cards. The credit card numbers generated by these formats pass the standard credit card validation tests by the applications, thereby making them appear like valid credit card numbers.

Some of the credit card formats you can use include:

- MasterCard numbers
- Visa card numbers
- American Express card numbers
- Discover Card numbers
- Any credit card number (credit card numbers belong to all types of cards)

You may want to use different styles for storing credit card numbers, such as:

- Pure numbers
- 'Space' for every four digits
- 'Hyphen' (-) for every four digits, and so forth

To implement the masked values in a certain format style, you can set the `DM_CC_FORMAT` variable of the `DM_FMTLIB` package. To install the package, see ["Installing the DM_FMTLIB Package"](#) on page 18-11.

United States Social Security Numbers Out of the box, you can generate valid U.S. Social Security (SSN) numbers. These SSNs pass the normal application tests of a valid SSN.

You can affect the format style by setting the `DM_SSN_FORMAT` variable of the `DM_FMTLIB` package. For example, if you set this variable to '-', the typical social security number would appear as '123-45-6789'.

ISBN Numbers Using the format library, you can generate either 10-digit or 13-digit ISBN numbers. These numbers adhere to standard ISBN number validation tests. All of these ISBN numbers are random in nature. Similar to other format definitions, you can affect the "style" of the ISBN format by setting values to `DM_ISBN_FORMAT`.

UPC Numbers Using the format library, you can generate valid UPC numbers. They adhere to standard tests for valid UPC numbers. You can affect the formatting style by setting the `DM_UPC_FORMAT` value of the `DM_FMTLIB` package.

Canadian Social Insurance Numbers Using the format library, you can generate valid Canadian Social Insurance Numbers (SINs). These numbers adhere to standard tests of Canadian SINs. You can affect the formatting style by setting the `DM_CN_SIN_FORMAT` value of the `DM_FMTLIB` package.

North American Phone Numbers Out of the box, the format library provides various possible U.S. and Canadian phone numbers. These are valid, realistic looking numbers that can pass standard phone number validation tests employed by applications. You can generate the following types of numbers:

- Any North American phone numbers
- Any Canadian phone number
- Any U.S.A. phone number

UK National Insurance Numbers Using the format library, you can generate valid unique random UK National Insurance Numbers (NINs). These numbers adhere to standard tests of UK NINs. A typical national insurance number would appear as 'GR 12 56 34 RS'.

Auto Mask This format scrambles characters and numbers into masked characters and numbers and while retaining the format and length of the data, including special characters; for example, 'ABCD_343-ddg' masked as 'FHDT_657-tte'.

Installing the DM_FMTLIB Package

The predefined masking formats use functions defined in the `DM_FMTLIB` package. This package is automatically installed in the `DBSNMP` schema of your Enterprise Manager repository database. To use the predefined masking formats on a target database (other than the repository database), you must manually install the `DM_FMTLIB` package on that database.

To install the DM_FMTLIB package:

1. Locate the following scripts in your Enterprise Manager installation:

```
$PLUGIN_HOME/sql/db/latest/masking/dm_fmtlib_pkgdef.sql
$PLUGIN_HOME/sql/db/latest/masking/dm_fmtlib_pkgbody.plb
```

Where `PLUGIN_HOME` can be any of the locations returned by the following SQL `SELECT` statement, executed as `SYSMAN`:

```
select PLUGIN_HOME from gc_current_deployed_plugin where
plugin_id='oracle.sysman.db' and destination_type='OMS';
```

2. Copy these scripts to a directory in your target database installation and execute them using `SQL*Plus`, connected as a user that can create packages in the `DBSNMP` schema.

You can now use the predefined masking formats in your masking definitions.

3. Select and import any predefined masking format into a masking definition by clicking the **Import Format** button on the Define Column Mask page.

Providing a Masking Format to Define a Column

When you create a masking definition ("[Masking with an Application Data Model and Workloads](#)" on page 18-15), you will be either importing a format or selecting one from the available types in the Define Column Mask page. Format entry options are as follows:

- **Array List**

The data type of each value in the list must be compatible with that of the masked column. Uniqueness must be guaranteed if needed. For example, for a unique key column that already has 10 distinct values, the array list should also contain at least 10 distinct values.

- **Delete**

Deletes the specified rows as identified by the condition clauses. If a column includes a delete format for one of its conditions, a foreign key constraint or a dependent column cannot refer to the table.

- **Encrypt**

Encrypts column data by specifying a regular expression. The column values in all the rows must match the regular expression. This format can be used to mask data consistently across databases. That is, for a given value it always generates the same masked value.

For example, the regular expression `[([1-9][0-9]{2})[_][0-9]{3}[-][0-9]{4}]` generates U.S. phone numbers such as (123) 456-7890.

This format supports a subset of the regular expression language. It supports encrypting strings of fixed widths. However, it does not support `*` or `+` syntax of regular expressions.

If a value does not match the format specified, the encrypted value may no longer produce one-to-one mappings. All non-confirming values are mapped to a single encrypted value, thereby producing a many-to-one mapping.

- **Fixed Number**

The type of column applicable to this entry is a `NUMBER` column or a `STRING` column. For example, if you mask a column that has a social security number, one of the entries can be Fixed Number 900. This format is combinable.

- **Fixed String**

The type of column applicable to this entry is a `STRING` column. For example, if you mask a column that has a License Plate Number, one of the entries can be Fixed String CA. This format is combinable.

- **Null Value**

Masks the column using a value of `NULL`. The column must be nullable.

- **Post-Processing Function**

This is a special function that you can apply to the mask value that the masking engine generates. This function takes the mask value as input and returns the actual mask value to be used for masking.

The post-processing function is called after the mask value is generated. You can use it, for instance, to add commas or dollar signs to a value. For example, if a mask value is a number such as 12000, the post processing function can modify this to \$12,000. Another use is for adding checksums or special encodings for the mask value that is produced.

In the following statement:

```
Function post_proc_udf_func (rowid varchar2, column_name varchar2, mask_value
varchar2) returns varchar2;
```

- rowid is the min (rowid) of the rows that contains the value mask_value 3rd argument.
- column_name is the name of the column being masked.
- mask_value is the value being masked.

- **Preserve Original Data**

Retains the original values for rows that match the specified condition clause. This is used in cases where some rows that match a condition do not need to be masked.

- **Random Dates**

The uniqueness of the Date column is not maintained after masking. This format is combinable.

- **Random Digits**

This format generates unique values within the specified range. For example, for a random digit with a length of [5,5], an integer between [0, 99999] is randomly generated, left padded with '0's to satisfy the length and uniqueness requirement. This is a complementary type of random number, which will not be padded. When using random digits, the random digit pads to the appropriate length in a string. It does not pad when used for a number column. This format is combinable.

Data masking ensures that the generated values are unique, but if you do not specify enough digits, you could run out of unique values in that range.

- **Random Numbers**

If used as part of a mixed random string, these have limited usage for generating unique values. This format generates unique values within the specified range. For example, a starting value of 100 and ending value of 200 generates an integer number ranging from 100 to 200, both inclusive. Note that Oracle Enterprise Manager release 10.2.0.4.0 does not support float numbers. This format is combinable.

- **Random Strings**

This format generates unique values within the specified range. For example, a starting length of 2 and ending length of 6 generates a random string of 2 - 6 characters in length. This format is combinable.

- **Shuffle**

This format randomly shuffles the original column data. It maintains data distribution except when a column is conditionally masked and its values are not unique.

For more information, see ["Using the Shuffle Format"](#) on page 18-26.

- **Substitute**

This format uses a hash-based substitution for the original value and always yields the same mask value for any given input value. Specify the substitution masking table and column. This format has the following properties:

- The masked data is not reversible. That is, this format is not vulnerable to external security breaches because the original value is replaced, so it is not possible to retrieve the original value from the mask value.
- Masking multiple times with a hash substitute across different databases yields the same mask value. This characteristic is valid across multiple databases or multiple runs assuming that the same substitution values are used in the two runs. That is, the actual rows and values in the substitution table do not change. For example, suppose the two values Joe and Tom were masked to Henry and Peter. When you repeat the same mask on another database using the same substitution table, if there were Bob and Tom, they might be replaced with Louise and Peter. Notice that even though the two runs have different data, Tom is always replaced with Peter.
- This format does not generate uniqueness.

- **Substring**

Substring is similar to the database `substr` function. The start position can be either a positive or a negative integer. For example, if the original string is `abcd`, a substring with a start position of 2 and length of 3 generates a masked string of `bcd`. A substring with start position of -2 and length of 3 generates a masked string of `cd`. This format is combinable.

- **Table Column**

A table column enables you to select values from the chosen column as the replacement value or part thereof. The data type and uniqueness must be compatible. Otherwise, a failure occurs when the job runs. This format is combinable.

- **Truncate**

Truncates all rows in a table. If one of the columns in a table is marked as truncated, the entire table is truncated, so no other mask formats can be specified for any of the other columns. If a table is being truncated, it cannot be referred to by a foreign key constraint or a dependent column.

- **User Defined Function**

The data type and uniqueness of the output values must be compatible with the original output values. Otherwise, a failure occurs when the job runs.

In the following statement:

```
Function udf_func (rowid varchar2, column_name varchar2, original_value  
varchar2) returns varchar2;
```

- `rowid` is the min (rowid) of the rows that contain the value `original_value` 3rd argument.
- `column_name` is the name of the column being masked.
- `original_value` is the value being masked.

Deterministic Masking Using the Substitute Format

You may occasionally need to consistently mask multiple, distinct databases. For instance, if you run HR, payroll, and benefits that have an employee ID concept on three separate databases, the concept may be consistent for all of these databases, in that an employee's ID can be selected to retrieve the employee's HR, payroll, or benefits information. Based on this premise, if you were to mask the employee's ID because it actually contains his/her social security number, you would have to mask this consistently across all three databases.

Deterministic masking provides a solution for this problem. You can use the Substitute format to mask employee ID column(s) in all three databases. The Substitute format uses a table of values from which to substitute the original value with a mask value. As long as this table of values does not change, the mask is deterministic or consistent across the three databases.

See Also: The online help for Define Column Mask page for more information on the Substitute format

Masking with an Application Data Model and Workloads

Before creating a masking definition, note the following prerequisites and advisory information:

- Ensure that you have the following minimum privileges for data masking:
 - `EM_ALL_OPERATOR` for Enterprise Manager Cloud Control users
 - `SELECT_CATALOG_ROLE` for database users
 - `SELECT ANY DICTIONARY` privilege for database users
 - `EXECUTE` privileges for the `DBMS_CRYPT` package
- Ensure the format you select does not violate check constraints and does not break any applications that use the data.
- For triggers and PL/SQL packages, data masking recompiles the object.
- Exercise caution when masking partitioned tables, especially if you are masking the partition key. In this circumstance, the row may move to another partition.
- Data Masking does not support clustered tables, masking information in object tables, XML tables, and virtual columns. Relational tables are supported for masking.
- If objects are layered on top of a table such as views, materialized views, and PL/SQL packages, they are recompiled to be valid.

If you plan to mask a test system intended for evaluating performance, the following practices are recommended:

- Try to preserve the production statistics and SQL profiles after masking by adding a pre-masking script to export the SQL profiles and statistics to a temporary table, then restoring after masking completes.
- Run a SQL Performance Analyzer evaluation to understand the masking impact on performance. Any performance changes other than what appears in the evaluation report are usually related to application-specific changes on the masked database.

To create a masking definition:

1. From the **Enterprise** menu, select **Quality Management**, then **Data Masking Definitions**.

The Data Masking Definitions page appears, where you can create and schedule new masking definitions and manage existing masking definitions.

See Also: the online help for the Data Masking Definitions page for more information on page user controls

2. Click **Create** to go to the Create Masking Definition page.

A masking definition includes information regarding table columns and the format for each column. You can choose which columns to mask, leaving the remaining columns intact.

See Also: The online help for the Create Masking Definition page for more information on page user controls

3. Provide a required **Name**, **Application Data Model**, and **Reference Database**.

When you click the search icon and select an Application Data Model (ADM) name from the list, the system automatically populates the Reference Database field.

- **Optional:** Check **Ensure Workload Masking Compatibility** if you want to mask Capture files and SQL Tuning Sets.

When you enable this check box, the masking definition is evaluated to determine if the SQL Expression format or conditional masking is being used. If either is in use when you click OK, the option becomes unchecked and an error message appears asking you to remove these items before selecting this option.

Note: Before proceeding to the next step, one or more sensitive columns must already be defined in the Application Data Model. See ["Managing Sensitive Column Types"](#) on page 16-5 for more information.

4. Click **Add** to go to the Add Columns page, where you can choose which sensitive columns in the ADM you want to mask.

See Also: The online help for the Add Columns page for more information on page user controls

5. Enter search criteria, then click **Search**.

The sensitive columns you defined in the ADM appear in the table below.

6. Either select one or more columns for later formatting on the Create Masking Definition page, or formatting now if the data types of the columns you have selected are identical.

See Also: ["Supported Data Types"](#) on page 3 for information on data types

7. *Optional:* if you want to mask selected columns as a group, enable **Mask selected columns as a group**. The columns that you want to mask as a group must all be from the same table.

Enable this check box if you want to mask more than one column together, rather than separately. When you select two or more columns and then later define the format on the Define Group Mask page, the columns appear together, and any choices you make for format type or masking table apply collectively to all of the columns.

After you define the group and return to this page, the Column Group column in the table shows an identical number for each entry row in the table for all members of the group. For example, if you have defined your first group containing four columns, each of the four entries in this page will show a number 1 in the Column Group column. If you define another group, the entries in the page will show the number 2, and so forth. This helps you to distinguish which columns belong to which column groups.

8. Either click **Add** to add the column to the masking definition, return to the Create Masking Definition page and define the format of the column later, or click **Define Format and Add** to define the format for the column now.

The Define Format and Add feature can save you significant time. When you select multiple columns to add that have the same data type, you do not need to define the format for each column as you would when you click Add. For instance, if you search for Social Security numbers (SSN) and the search yields 100 SSN columns, you could select them all, then click **Define Format and Add** to import the SSN format for all of them.

9. Do one of the following:

- If you clicked **Add** in the previous step:

You will eventually need to define the format of the column in the Create Masking Definition page before you can continue. When you are ready to do so, click the icon in the page Format column for the column you want to format. Depending on whether you decided to mask selected columns as a group on the Add Columns page, either the Define Column mask or Define Group mask appears. Read further in this step for instructions for both cases.

- If you clicked **Define Format and Add** in the previous step and did not check **Mask selected columns as a group**:

The Define Column Mask page appears, where you can define the format for the column before adding the column to the Create Masking Definition page, as explained below:

- Provide a format entry for the required Default condition by either selecting a format entry from the list and clicking **Add**, or clicking **Import Format**, selecting a predefined format on the Import Format page, then clicking **Import**.

The Import Format page displays the formats that are marked with the same sensitive type as the masked column.

For information about Oracle-supplied predefined masking format definitions, see ["Using Oracle-supplied Predefined Masking Formats"](#) on page 18-9.

For descriptions of the choices available in the Format Entry list, see ["Providing a Masking Format to Define a Column"](#) on page 18-12.

- Add another condition by clicking **Add Condition** to add a new condition row, then provide one or more format entries as described in the previous step.

- When you have finished formatting the column, click **OK** to return to the Create Masking Definition page.
 - If you clicked **Define Format and Add** in the previous step and checked **Mask selected columns as a group**:

The Define Group Mask page appears, where you can add format entries for group columns that appear in the Create Masking Definition page, as explained below:

 - Select one of the available format types. For complete information on the format types, see the online help for the Defining the Group Masking Format topic.

For descriptions of the choices available in the Format Entry list, see "[Providing a Masking Format to Define a Column](#)" on page 18-12.

 - Optionally add a column to the group.
 - When you have finished formatting the group, click **OK** to return to the Create Masking Definition page.

Your configuration appears in the Columns table. The sensitive columns you selected earlier now appear on this page. The selected columns are the primary key, and the foreign key columns are listed below. These columns are masked as well.
10. Expand **Show Advanced Options** and decide whether the selected default data masking options are satisfactory.

For more information, see "[Selecting Data Masking Advanced Options](#)" on page 18-20.
 11. Click **OK** to save your definition and return to the Data Masking Definitions page.

At this point, super administrators can see each other's masking definitions.
 12. Select the definition and click **Generate Script**. The schedule job dialog opens. You may have to log in to the database first.

Complete the schedule job dialog by providing the required information, then click **Submit**.
 13. A message appears denoting that the job was submitted successfully and you return to the Data Masking Definitions page, where the status is "Generating Script." Click **View Job Details** to open the job summary page.

When the job completes, click **Log Report** to check whether sufficient disk space is available for the operation, and to determine the impact on other destination objects, such as users, after masking. If any tables included in the masking definition have columns of data type `LONG`, a warning message may appear. For more information, see "[Using Data Masking with LONG Columns](#)" on page 28.
 14. When the status on the Data Masking Definitions page is "Script Generated," select the script and choose from the following actions:
 - **Clone Database**—to clone and mask the database using the Clone Database wizard. For more information, see "[Cloning the Production Database](#)" on page 18-23.
 - **Create Like**—to display the masking definition in the Create Masking Definition page where you can customize the definition.
 - **Export**—to export the definition as an XML file for use in other repositories.

- **Save Script**—to save the entire PL/SQL script to your desktop.
- **View Script**—to view the PL/SQL script, which you can edit and save. You can also view errors and warnings, if any, in the impact report.

Click **Go** to execute the selected action.

15. If you are already working with a test database and want to directly mask the data in this database, click **Schedule Job**.

- Provide the requisite information and desired options. You can specify the database at execution time to any database. The system assumes that the database you select is a clone of the source database. By default, the source database from the ADM is selected.
- Click **Submit**.

The Data Masking Definitions page appears. The job has been submitted to Enterprise Manager and the masking process appears. The Status column on this page indicates the current stage of the process.

See Also: The online help for Scheduling a Data Masking Job for more information on page user controls

Note that you can also perform data masking at the source as part of a data subsetting definition. See "[Creating a Data Subset Definition](#)" on page 17-2 for more information.

Adding Dependent Columns

Dependent columns are defined by adding them to the Application Data Model. The following prerequisites apply for the column to be defined as dependent:

- A valid dependent column should not already be included for masking.
- The column should not be a foreign key column or referenced by a foreign key column.
- The column data should conform to the data in the parent column.

If the column does not meet these criteria, an "Invalid Dependent Columns" message appears when you attempt to add the dependent column.

Masking Dependent Columns for Packaged Applications

The following procedure explains how to mask data across columns for packaged applications in which the relationships are not defined in the data dictionary.

To mask dependent columns for packaged applications:

1. Go to Data Discovery and Modeling and create a new Application Data Model (ADM) using metadata collection for your packaged application suite.

When metadata collection is complete, edit the newly created ADM.

2. Manually add a referential relationship:
 - a. From the Referential Relationships tab, open the **Actions** menu, then select **Add Referential Relationship**.

The Add Referential Relationship pop-up window appears.

- b. Select the requisite Parent Key and Dependent Key information.

- c. In the Columns Name list, select a dependent key column to associate with a parent key column.
- d. Click **OK** to add the referential relationship to the ADM.

The new dependent column now appears in the referential relationships list.

- 3. Perform sensitive column discovery.

When sensitive column discovery is complete, review the columns found by the discovery job and mark them sensitive or not sensitive as needed.

When marked as sensitive, any discovery sensitive column also marks its parent and the other child columns of the parent as sensitive. Consequently, it is advisable to first create the ADM with all relationships. ADM by default, or after running drivers, may not contain denormalized relationships. You need to manually add these.

For more information about sensitive column discovery, see step 6 on page 16-4.

- 4. Go to Data Masking and create a new masking definition.
- 5. Select the newly created ADM and click **Add**, then **Search** to view this ADM's sensitive columns.
- 6. Select columns based on your search results, then import formats for the selected columns.

Enterprise Manager displays formats that conform to the privacy attributes.

- 7. Select the format and generate the script.
- 8. Execute the masking script.

Enterprise Manager executes the generated script on the target database and masks all of your specified columns.

Selecting Data Masking Advanced Options

The following options on the Masking Definitions page are all checked by default, so you need to uncheck the options that you do not want to enable:

- [Data Masking Options](#)
- [Random Number Generation](#)
- [Pre- and Post-mask Scripts](#)

Data Masking Options

The data masking options include:

- Disable redo log generation during masking

Masking disables redo logging and flashback logging to purge any original unmasked data from logs. However, in certain circumstances when you only want to test masking, roll back changes, and retry with more mask columns, it is easier to uncheck this box and use a flashback database to retrieve the old unmasked data after it has been masked. You can use Enterprise Manager to flashback a database.

Note: Disabling this option compromises security. You must ensure this option is enabled in the final mask performed on the copy of the production database.

- Refresh statistics after masking

If you have already enabled statistics collection and would like to use special options when collecting statistics, such as histograms or different sampling percentages, it is beneficial to turn off this option to disable default statistics collection and run your own statistics collection jobs.

- Drop temporary tables created during masking

Masking creates temporary tables that map the original sensitive data values to mask values. In some cases, you may want to preserve this information to track how masking changed your data. Note that doing so compromises security. These tables must be dropped before the database is available for unprivileged users.

- Decrypt encrypted columns

This option decrypts columns that were previously masked using Encrypt format. To decrypt a previously encrypted column, the seed value must be the same as the value used to encrypt.

Decrypt only recovers the original value if the original format used for the encryption matches the original value. If the originally encrypted value did not conform to the specified regular expression, when decrypted, the encrypted value cannot reproduce the original value.

- Use parallel execution when possible

Oracle Database can make parallel various SQL operations that can significantly improve their performance. Data Masking uses this feature when you select this option. You can enable Oracle Database to automatically determine the degree of parallelism, or you can specify a value. For more information about using parallel execution and the degree of parallelism, see the *Oracle Database Data Warehousing Guide*.

- Recompile invalid dependent objects after masking

The masking process re-creates the table to be masked and as a consequence, all existing dependent objects (packages, procedures, functions, MViews, Views, Triggers) become invalid. You can specify that the masking process recompile these invalid objects after creating the table, by selecting the check box. Otherwise, invalid objects are recompiled using `utl_comp` procedures at the end of masking.

If you choose this option, indicate whether to use serial or parallel execution. You can enable Oracle Database to automatically determine the degree, or you can specify a value. For more information about using parallel execution and the degree of parallelism, see the *Oracle Database Data Warehousing Guide*.

Random Number Generation

The random number generation options include:

- Favor Speed

The `DBMS_RANDOM` package is used for random number generation.

- Favor Security

The `DBMS_CRYPTO` package is used for random number generation. Additionally, if you use the Substitute format, a seed value is required when you schedule the masking job or database clone job.

Pre- and Post-mask Scripts

When masking a test system to evaluate performance, it is beneficial to preserve the object statistics after masking. You can accomplish this by adding a pre-masking script to export the statistics to a temporary table, then restoring them with a post-masking script after masking concludes.

Use the Pre Mask Script text box to specify any user-specified SQL script that must run before masking starts.

Use the Post Mask Script text box to specify any user-specified SQL script that must run after masking completes. Since masking modifies data, you can also perform tasks, such as rebalancing books or calling roll-up or aggregation modules, to ensure that related or aggregate information is consistent.

The following examples show pre- and post-masking scripts for preserving statistics.

Example 18–1 Pre-masking Script for Preserving Statistics

```
variable sts_task VARCHAR2(64);

/*Step :1 Create the staging table for statistics*/

exec dbms_stats.create_stat_table(ownname=>'SCOTT',stattab=>'STATS');

/* Step 2: Export the table statistics into the staging table. Cascade results in
all index and column statistics associated with the specified table being exported
as well. */

exec
dbms_stats.export_table_stats(ownname=>'SCOTT',tabname=>'EMP',
partname=>NULL,stattab=>'STATS',statid=>NULL,cascade=>TRUE,statown=>'SCOTT');
exec
dbms_stats.export_table_stats(ownname=>'SCOTT',tabname=>'DEPT',
partname=>NULL,stattab=>'STATS',statid=>NULL,cascade=>TRUE,statown=>'SCOTT');

/* Step 3: Create analysis task */
3. exec :sts_task := DBMS_SQLPA.create_analysis_task(sqlset_name=>
'scott_test_sts',task_name=>'SPA_TASK', sqlset_owner=>'SCOTT');

/*Step 4: Execute the analysis task before masking */
exec DBMS_SQLPA.execute_analysis_task(task_name => 'SPA_TASK',
execution_type=> 'explain plan', execution_name => 'pre-mask_SPA_TASK');
```

Example 18–2 Post-masking Script for Preserving Statistics

```
*Step 1: Import the statistics from the staging table to the dictionary tables*/

exec
dbms_stats.import_table_stats(ownname=>'SCOTT',tabname=>'EMP',
partname=>NULL,stattab=>'STATS',statid=>NULL,cascade=>TRUE,statown=>'SCOTT');
exec
dbms_stats.import_table_stats(ownname=>'SCOTT',tabname=>'DEPT',
partname=>NULL,stattab=>'STATS',statid=>NULL,cascade=>TRUE,statown=>'SCOTT');

/* Step 2: Drop the staging table */

exec dbms_stats.drop_stat_table(ownname=>'SCOTT',stattab=>'STATS');

/*Step 3: Execute the analysis task before masking */
exec DBMS_SQLPA.execute_analysis_task(task_name=>'SPA_TASK',
execution_type=>'explain plan', execution_name=>'post-mask_SPA_TASK');
```



```
/*Step 4: Execute the comparison task */
exec DBMS_SQLPA.execute_analysis_task(task_name =>'SPA_TASK',
execution_type=>'compare', execution_name=>'compare-mask_SPA_TASK');
```

See Also: ["Masking a Test System to Evaluate Performance"](#) on page 24 for a procedure that explains how to specify the location of these scripts when scheduling a data masking job

Cloning the Production Database

When you clone and mask the database, a copy of the masking script is saved in the Enterprise Manager repository and then retrieved and executed after the clone process completes. Therefore, it is important to regenerate the script after any schema changes or modifications to the production database.

To clone and optionally mask the masking definition's target database:

1. From the Data Masking Definitions page, select the masking definition you want to clone, select **Clone Database** from the Actions list, then click **Go**.

The Clone Database: Source Type page appears.

The Clone Database wizard appears, where you can create a test system to run the mask.

2. Specify the type of source database backup to be used for the cloning operation, then click **Continue**.
3. Proceed through the wizard steps as you ordinarily would to clone a database. For assistance, refer to the online help for each step.
4. In the Database Configuration step of the wizard, add a masking definition, then select the Run SQL Performance Analyzer option as well as other options as desired or necessary.
5. Schedule and then run the clone job.

Importing a Data Masking Template

You can import and re-use a previously exported data masking definition template, including templates for Fusion Applications, saved as an XML file to the current Enterprise Manager repository.

Note the following advisory information:

- The XML file format must be compliant with the masking definition XML format.
- Verify that the name of the masking definition to be imported does not already exist in the repository.
- Verify that the target name identifies a valid Enterprise Manager target.

To import a data masking template:

1. From the Data Masking Definitions page, click **Import**.

The Import Masking Definition page appears.

2. Specify the ADM associated with the template. The Reference Database is automatically provided.
3. Browse for the XML file, or specify the name of the XML file, then click **Continue**.

The Data Masking Definitions Page reappears and displays the imported definition in the table list for subsequent viewing and masking.

Masking a Test System to Evaluate Performance

After you have created a data masking definition, you may want to use it to analyze the performance impact from masking on a test system. The procedures in the following sections explain the process for this task for masking only, or cloning and masking.

Using Only Masking for Evaluation

To use only masking to evaluate performance:

1. From the Data Masking Definitions page, select the masking definition to be analyzed, then click **Schedule Job**.

The Schedule Data Masking Job page appears.

2. At the top of the page, provide the requisite information.

The script file location pertains to the masking script, which also contains the pre- and post-masking scripts you created in "[Pre- and Post-mask Scripts](#)" on page 18-22.

3. In the Encryption Seed section, provide a text string that you want to use for encryption.

This section only appears for masking definitions that use the Substitute or Encrypt formats. The seed is an encryption key used by the encryption/hash-based substitution APIs, and makes masking more deterministic instead of being random.

4. In the Workloads section:

- a. Select the **Mask SQL Tuning Sets** option, if desired.

If you use a SQL Tuning Set that has sensitive data to evaluate performance, it is beneficial to mask it for security, consistency of data with the database, and to generate correct evaluation results.

- b. Select the **Capture Files** option, if desired, then select a capture directory.

When you select this option, the contents of the directory is masked. The capture file masking is executed consistently with the database.

5. In the Detect SQL Plan Changes Due to Masking section, leave the Run SQL Performance Analyzer option unchecked.

You do not need to enable this option because the pre- and post-masking scripts you created, referenced in step 2, already execute the analyzer.

6. Provide credentials and scheduling information, then click **Submit**.

The Data Masking Definitions page reappears, and a message appears stating that the Data Masking job has been submitted successfully.

During masking of any database, the AWR bind variable data is purged to protect sensitive bind variables from leaking to a test system.

7. When the job completes successfully, click the link in the SQL Performance Analyzer Task column to view the executed analysis tasks and Trial Comparison Report, which shows any changes in plans, timing, and so forth.

Using Cloning and Masking for Evaluation

Using both cloning and masking to evaluate performance is very similar to the procedure described in the previous section, except that you specify the options from the Clone Database wizard, rather than from the Schedule Data Masking Job page.

To use both cloning and masking to evaluate performance:

1. Follow the steps described in ["Cloning the Production Database"](#) on page 18-23.
2. At step 4, the format of the Database Configuration step appears different from the Schedule Data Masking Job page discussed in ["Using Only Masking for Evaluation"](#), but select options as you would for the Schedule Data Masking Job page.
3. Continue with the wizard steps to complete and submit the cloning and masking job.

Upgrade Considerations

Upgrading data masking definitions from 10 or 11 Grid Control to 12c Cloud Control assumes that you have completed the following tasks:

- Upgraded Enterprise Manager to 12c
- Downloaded the latest database plug-in using Self Update and deployed the plug-in to OMS and Management Agent

Completing these tasks automatically upgrades the masking definitions and creates for each a shell Application Data Model (ADM) that becomes populated with the sensitive columns and their dependent column information from the legacy mask definition. The ADM, and hence data masking, then remains in an unverified state, because it is missing the dictionary relationships.

Proceed as follows to complete the masking definition upgrade:

1. From the **Enterprise** menu, select **Quality Management**, then select **Data Discovery and Modeling**.
2. For each shell ADM (verification status is Needs Upgrade), do the following:
 - a. Select the ADM in the table.
 - b. From the Actions menu, select **Upgrade and Verify**.
 - c. Schedule and submit the job.

When the job completes, verification status should be Valid.
3. From the **Enterprise** menu, select **Quality Management**, then select **Data Masking Definitions**.
4. For each upgraded masking definition, do the following:
 - a. Open the masking definition for editing.
 - b. In **Advanced Options**, select the "Recompile invalid dependent objects after masking" option, with Parallel and Default settings.
 - c. Click **OK** to save your changes.
5. Next, schedule a script generation job for each upgraded masking definition.

You can now resume masking with the upgraded data masking definitions.

See Also: ["Adding Dependent Columns"](#) on page 19 for information on dependent columns

Consider these other points regarding upgrades:

- You can combine multiple upgraded ADMs by exporting an ADM and performing an Import Content into another ADM.
- An upgraded ADM uses the same semantics as for upgrading a legacy mask definition (discussed above), in that you would need to perform a validation.
- An 11.1 Grid Control E-Business Suite (EBS) masking definition based on an EBS masking template shipped from Oracle is treated as a custom application after the upgrade. You can always use the approach discussed in the first bulleted item above to move into a newly created EBS ADM with all of the metadata in place. However, this is not required.

Using the Shuffle Format

A shuffle format is available that does not preserve data distribution when the column values are not unique and also when it is conditionally masked. For example, consider the Original Table ([Table 18–1](#)) that shows two columns: EmpName and Salary. The Salary column has three distinct values: 10, 90, and 20.

Table 18–1 Original Table (Non-preservation)

EmpName	Salary
A	10
B	90
C	10
D	10
E	90
F	20

If you mask the Salary column with this format, each of the original values is replaced with one of the values from this set. Assume that the shuffle format replaces 10 with 20, 90 with 10, and 20 with 90 ([Table 18–2](#)).

Table 18–2 Mapping Table (Non-preservation)

EmpName	Salary
10	20
90	10
20	90

The result is a shuffled Salary column as shown in the Masked Table ([Table 18–3](#)), but the data distribution is changed. While the value 10 occurs three times in the Salary column of the Original Table, it occurs only twice in the Masked Table.

Table 18–3 Masked Table (Non-preservation)

EmpName	Salary
A	20

Table 18–3 (Cont.) Masked Table (Non-preservation)

EmpName	Salary
B	10
C	20
D	20
E	10
F	90

If the salary values had been unique, the format would have maintained data distribution.

Using Group Shuffle

Group shuffle enables you to perform a shuffle within discrete units, or groups, where there is a relationship among the members of the group. Consider the case of shuffling the salaries of employees. [Table 18–4](#) illustrates the group shuffle mechanism, where employees are categorized as managers (M) or workers (W), and salaries are shuffled within job category.

Table 18–4 Group Shuffle Using Job Category

Employee	Job Category	Salary	Shuffled Salary
Alice	M	90	88
Bill	M	88	90
Carol	W	72	70
Denise	W	57	45
Eddie	W	70	57
Frank	W	45	72

Using Conditional Masking

To demonstrate how conditional masking can handle duplicate values, add to [Table 18–4](#) another job category, assistant (A), where the employee in this category, George, earns the same as Frank. Assume the following conditions:

- If job category is M, replace salary with a random number between 1 and 10.
- If job category is W, set salary to a fixed number (01).
- Default is to preserve the existing value.

Applying these conditions results in the masked values shown in [Table 18–5](#):

Table 18–5 Using Job Category for Group Shuffle

Employee	Job Category	Salary	Conditional Result
Alice	M	90	5
Bill	M	88	7
Carol	W	72	01
Denise	W	57	01

Table 18–5 (Cont.) Using Job Category for Group Shuffle

Employee	Job Category	Salary	Conditional Result
Eddie	W	70	01
Frank	W	45	01
George	A	45	45

Conditional masking works when there are duplicate values provided there are no dependent columns or foreign keys. If either of these is present, a "bleeding condition" results in the first of two duplicate values becoming the value of the second. So, in the example, George's salary is not preserved, but becomes 01.

Using Data Masking with LONG Columns

When data masking script generation completes, an impact report appears. If the masking definition has tables with columns of data type LONG, the following warning message is displayed in the impact report:

```
The table <table_name> has a LONG column. Data Masking uses "in-place" UPDATE to
mask tables with LONG columns. This will generate undo information and the
original data will be available in the undo tablespaces during the undo retention
period. You should purge undo information after masking the data. Any orphan rows
in this table will not be masked.
```

A

Application Data Model (ADM)

- application tables, viewing and editing, 16-3
 - associating a database to, 16-5, 16-6
 - creating, 16-2
 - prerequisites for, 16-2
 - definition of, 16-1
 - discovering sensitive columns, 16-4, 18-19, 18-20
 - editing application tables, 16-3
 - manually adding referential relationships, 16-4, 18-19
 - relationship to other DDM components, 16-1
 - sensitive columns
 - automatically discovering, 16-4
 - changing the type for, 16-5
 - creating a type for, 16-5
 - manually adding, 16-5
 - source database status, 16-8
 - upgrading, 16-9
 - verifying source database, 16-8
 - viewing application tables, 16-3
 - viewing referential relationships, 16-4
- application database administrator, role of in data masking, 18-2

C

capture subset

- about, 14-3
 - generating, 14-9
- changing the type for sensitive columns, 16-5
- cloning production database in data masking, 18-6
- ### Consolidated Database Replay
- about, 14-1
 - connection remapping, 14-6, 14-16
 - initializing, 14-15
 - preparing, 14-17
 - replay directory
 - setting, 14-9
 - replay options, 14-6
 - reporting, 14-7
 - running, 14-14
 - sample scenario, 14-18
 - starting, 14-18
 - steps, 14-2

test system

- capture directory, 14-4
 - setting up, 14-4
 - user remapping, 14-6, 14-16
 - using with APIs, 14-8
 - workload captures
 - supported, 14-3
- ### creating
- Application Data Model (ADM), 16-2
 - data masking definition, 18-15
 - data subset definition, 17-2
 - masking definition, 18-4
 - masking formats, 18-7
 - sensitive column type, 16-5

D

Data Discovery and Modeling, 16-1

data masking

- Add Columns page, 18-16
- adding columns to a masking definition, 18-7
- advanced options, 18-20
- application database administrator, role of, 18-2
- auto mask, 18-11
- Canadian social insurance numbers, 18-11
- cloning and masking, 18-25
- cloning the production database, 18-6, 18-23
- Create Masking Definition page, 18-16
- creating formats, 18-7
- creating masking definition, 18-4, 18-15
- Data Masking Definitions page, 18-16
- Define Column Mask page, 18-15
- defining new formats, 18-7
- dependent columns, adding, 18-19
- description, 18-2
- deterministic masking, 18-15
- DM_FMTLIB package, installing, 18-11
- encryption seed, 18-24
- evaluating performance, 18-24
- format entry options, 18-12
- importing data masking templates, 18-23
- information security administrator, role of, 18-2
- ISBN numbers, 18-11
- major task steps, 18-6
- mask format libraries, description of, 18-4
- masking definitions, description of, 18-4

- masking dependent columns, 18-19
- masking format templates, using, 18-9
- masking selected columns as a group, 18-16
- minimum privileges, 18-15
- North American phone numbers, 18-11
- other Oracle security products, 18-3
- patterns of format definitions, 18-9
- post-processing functions, 18-8
- pre- and post-masking scripts, 18-22
- predefined masking formats, 18-9
- primary key, 18-4
- random number generation options, 18-21
- security administrator, 18-5
- shuffle format examples, 18-26
- social security numbers, 18-10
- staging region, 18-5
- substitute format, 18-15
- supported data types, 18-3
- UK national insurance numbers, 18-11
- UPC numbers, 18-11
- upgrading, 18-26
- user-defined functions, 18-8
- using with LONG columns, 18-28
- workflow, 18-6
- data subsetting
 - Ancestor and Descendant Tables, 17-4
 - Ancestor Tables Only, 17-4
 - creating a definition, 17-2
 - exporting subset templates, 17-11
 - generating a subset, 17-6
 - importing subset templates, 17-9
 - providing rules, 17-3
 - required privileges, 17-2, 17-11
 - saving a subset script, 17-8
 - space estimates, 17-6
 - specifying Where clause, 17-4
- data, hiding with data masking, 18-2
- Database Replay
 - about, 1-2
 - methodology, 8-1
 - replay clients
 - about, 8-3, 11-5
 - calibrating, 11-5
 - starting, 11-5, 11-6
 - replay filter set
 - about, 11-4
 - reporting, 8-3, 12-1
 - replay compare period reports, 12-9, 12-10
 - SQL Performance Analyzer reports, 12-14
 - usage, 1-2, 8-1
 - workflow, 8-1
 - workload capture
 - about, 9-1
 - capture directory, 9-2
 - capture files, 8-2
 - capturing, 8-2, 9-7, 9-16, 9-17
 - exporting data, 9-19
 - importing data, 9-19
 - managing, 9-15
 - monitoring, 9-14, 9-20
 - options, 9-2
 - prerequisites, 9-1
 - reporting, 12-1
 - restarting the database, 9-2
 - restrictions, 9-4
 - stopping, 9-15, 9-19
 - workload filters
 - about, 9-3, 11-4
 - defining, 9-17
 - exclusion filters, 9-4, 11-5
 - inclusion filters, 9-4, 11-4
 - workload preprocessing
 - about, 8-3, 10-1
 - preprocessing, 10-9
 - workload replay
 - about, 8-3, 11-1
 - cancelling, 11-25
 - exporting data, 11-25
 - filters, 11-20
 - importing data, 11-25
 - monitoring, 11-15, 11-26
 - options, 11-3, 11-19
 - pausing, 11-24
 - replaying, 11-8, 11-17
 - reporting, 12-4
 - resuming, 11-24
 - starting, 11-24
 - steps, 11-2
- database upgrades
 - testing, 7-1
- database version
 - production system, 7-2, 7-10
 - system running SQL Performance Analyzer, 7-2, 7-10
 - test system, 7-2, 7-10
- DBMS_SPM package
 - LOAD_PLANS_FROM_SQLSET function, 6-26
- DBMS_SQLPA package
 - CREATE_ANALYSIS_TASK function, 3-13
 - EXECUTE_ANALYSIS_TASK procedure, 4-4, 5-3, 6-10, 7-9, 7-15
 - REPORT_ANALYSIS_TASK function, 6-11
 - SET_ANALYSIS_TASK_PARAMETER procedure, 3-14, 3-15
- DBMS_SQLTUNE package
 - CREATE_TUNING_TASK function, 6-23
 - SELECT_SQL_TRACE function, 7-5
- DBMS_WORKLOAD_CAPTURE package
 - ADD_FILTER procedure, 9-17
 - DELETE_FILTER procedure, 9-17
 - EXPORT_AWR procedure, 9-19
 - FINISH_CAPTURE procedure, 9-19
 - GET_CAPTURE_INFO procedure, 12-2
 - IMPORT_AWR function, 9-20
 - REPORT function, 12-2
 - START_CAPTURE procedure, 9-18
- DBMS_WORKLOAD_REPLAY package
 - ADD_CAPTURE function, 14-11
 - ADD_SCHEDULE_ORDERING

- function, 14-13
- GET_REPLAY_DIRECTORY function, 14-10
- INITIALIZED_CONSOLIDATED_REPLAY
 - procedure, 14-15
- REMOVE_CAPTURE procedure, 14-12
- SET_REPLAY_DIRECTORY procedure, 14-9
- ADD_FILTER procedure, 11-21
- BEGIN_REPLAY_SCHEDULE procedure, 14-11
- CANCEL_REPLAY procedure, 11-25
- COMPARE_PERIOD_REPORT procedure, 12-10
- COMPARE_SQLSET_REPORT procedure, 12-14
- CREATE_FILTER_SET procedure, 11-21
- DELETE_FILTER procedure, 11-21
- END_REPLAY_SCHEDULE procedure, 14-14
- EXPORT_AWR procedure, 11-25
- GENERATE_CAPTURE_SUBSET
 - procedure, 14-9
- GET_DIVERGING_STATEMENT function, 11-26
- GET_REPLAY_INFO procedure, 12-8
- IMPORT_AWR function, 11-25
- INITIALIZE_REPLAY procedure, 11-17
- PAUSE_REPLAY procedure, 11-24
- PREPARE_CONSOLIDATED_REPLAY
 - procedure, 14-18
- PREPARE_REPLAY procedure, 11-19
- PROCESS_CAPTURE procedure, 10-9
- REMAP_CONNECTION procedure, 11-18, 14-16
- REPORT function, 12-8
- RESUME_REPLAY procedure, 11-24
- SET_REPLAY_TIMEOUT procedure, 11-23
- SET_USER_MAPPING procedure, 11-18, 14-17
- START_CONSOLIDATED_REPLAY
 - procedure, 14-18
- START_REPLAY procedure, 11-24
- USE_FILTER_SET procedure, 11-22
- deterministic masking, 18-15
- discovering sensitive columns, 16-4, 18-19

E

- editing application tables, 16-3
- exporting
 - subset templates, data subsetting, 17-11

H

- hiding data using data masking, 18-2

I

- importing
 - data masking templates, 18-23
 - subset templates, data subsetting, 17-9
- information security administrator, role of in data
 - masking, 18-2

M

- manually adding sensitive columns, 16-5
- mapping table
 - about, 7-4

- creating, 7-3, 7-4
- moving, 7-3, 7-4
- mask format libraries, 18-4
- masking definitions, 18-4
 - credit card numbers, 18-10
- masking formats
 - entry options, 18-12
 - predefined, 18-9

O

- Oracle Data Masking Pack, 16-1
- Oracle Test Data Management Pack, 16-1

P

- post-processing functions, data masking, 18-8
- prerequisites for creating an ADM, 16-2
- primary key, data masking and, 18-4
- privileges
 - data subsetting, 17-2
 - minimum for data masking, 18-15

R

- Real Application Testing
 - about, 1-1
 - components, 1-1
- referential relationships
 - manually adding, 16-4, 18-19
 - viewing, 16-4
- regulatory compliance using masked data, 18-2
- replay schedule
 - about, 14-5
 - defining, 14-10, 14-11
 - saving, 14-14
 - schedule orders
 - adding, 14-12
 - removing, 14-13
 - workload captures
 - adding, 14-11
 - removing, 14-12
- Right to Financial Privacy Act of 1978, 18-2

S

- Sarbanes-Oxley regulatory requirements, 18-2
- schedule order
 - about, 14-6
 - viewing, 14-13
- security
 - compliance with masked data, 18-2
 - data masking, 18-2
 - list of Oracle products, 18-3
 - mask format libraries, 18-4
 - masking definitions, 18-4
- security administrator, data masking and, 18-5
- sensitive columns
 - changing the type for, 16-5
 - creating the type for, 16-5
 - discovering, 16-4, 18-19

- discovering automatically, 16-4
- manually adding, 16-5
- performing discovery of, 18-20
- SQL Performance Analyzer
 - about, 1-1
 - comparing performance, 6-10, 7-3, 7-11
 - creating a task, 3-1, 3-2, 3-13
 - executing the SQL workload, 4-2, 4-4
 - executing the SQL workload after a change, 5-2, 5-3
 - initial environment
 - establishing, 4-1
 - input source, 7-1
 - making a change, 5-1
 - methodology, 2-1
 - monitoring, 6-26
 - performance data
 - collecting post-change version, 5-1
 - collecting pre-change version, 4-1
 - comparing, 6-1
 - remote test execution, 7-6, 7-11, 7-12
 - reporting, 2-8
 - setting up the test system, 2-4
 - SQL Performance Analyzer report
 - active reports, 6-7
 - general information, 6-5, 6-12
 - global statistics, 6-5
 - global statistics details, 6-6
 - result details, 6-15
 - result summary, 6-13
 - reviewing, 6-3, 6-12
 - SQL tuning set
 - selecting, 2-5, 3-1
 - SQL workload
 - capturing, 2-3
 - executing, 2-5, 2-7
 - transporting, 2-4
 - system change
 - making, 5-1
 - task
 - creating, 7-3, 7-11
 - usage, 1-2
 - using, 2-1
 - workflow, 2-1
 - Exadata simulation, 3-9
 - guided, 3-12
 - optimizer statistics, 3-6
 - parameter change, 3-3
- SQL plan baselines
 - creating, 6-26
- SQL statements
 - regressed, 1-2, 2-8, 6-8, 6-22, 6-24, 7-3, 7-11, 7-17
- SQL Trace
 - about, 7-3
 - enabling, 7-2, 7-4
 - trace level, 7-4
- SQL trace files
 - about, 7-3
 - moving, 7-3, 7-4
- SQL trials

- about, 2-5, 2-6
- building
 - post-upgrade version, 7-3, 7-6, 7-11
 - pre-upgrade version, 7-3, 7-9, 7-11, 7-15
- comparing, 6-2
- SQL tuning set
 - about, 2-3
 - building, 7-5
 - comparing, 6-17
 - constructing, 7-3
 - converting, 7-3
- staging region, data masking and, 18-5
- statuses for Source Database Status column, ADM, 16-8
- substitute format, data masking and, 18-15
- supported data types, data masking, 18-3

U

- upgrade environment, 7-2, 7-10
- upgrading
 - ADM, 16-9
 - data masking, 18-26
- user-defined functions, data masking and, 18-8

V

- verifying source database, ADM, 16-8
- viewing
 - application tables, 16-3
 - referential relationships, 16-4

W

- Where clause, specifying for data subsetting, 17-4
- Workload Analyzer
 - about, 10-9
 - running, 10-10
- Workload Intelligence
 - about, 13-1
 - BuildModel program
 - about, 13-4
 - options, 13-4
 - syntax, 13-4
 - creating a database user, 13-3
 - creating a job, 13-3
 - creating a workload model, 13-4
 - FindPatterns program
 - about, 13-5
 - options, 13-5
 - syntax, 13-5
 - GenerateReport program
 - about, 13-6
 - options, 13-6
 - syntax, 13-6
 - generating report, 13-6
 - identifying patterns, 13-5
 - LoadInfo program
 - about, 13-3
 - options, 13-4
 - syntax, 13-3

pattern, 13-2
sample scenario, 13-7
template, 13-2
using, 13-3

