Oracle® Database

Real Application Testing User's Guide

11*g* Release 1 (11.1)

E12253-01

August 2008

ORACLE®

Oracle Database Real Application Testing User's Guide, 11g Release 1 (11.1)

E12253-01

Copyright © 2008, Oracle. All rights reserved.

Primary Author: Immanuel Chan

Contributors: Lance Ashdown, Pete Belknap, Supiti Buranawatanachoke, Romain Colle, Karl Dias, Leonidas Galanis, Prabhaker Gongloor, Mughees Minhas, Yujun Wang, Khaled Yagoub

Contents

4 Preprocessing a Database Workload

5 Replaying a Database Workload

6 Analyzing Replayed Workload

Part II SQL Performance Analyzer

7 Introduction to SQL Performance Analyzer

8 Creating an Analysis Task

9 Creating a Pre-Change SQL Trial

10 Creating a Post-Change SQL Trial

11 Comparing SQL Trials

12   Testing a Database Upgrade

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 using Oracle Real Application Testing. This document is intended for database administrators, application designers, and programmers who are responsible for performing real application testing on Oracle Database.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

http://www.oracle.com/accessibility/

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, 7 days a week. For TTY support, call 800.446.2398. Outside the United States, call +1.407.458.2479.

Related Documents

For more information about some of the topics discussed in this document, see the following documents in the Oracle Database Release 11.1 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*

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.
italic	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.

1

Introduction to Oracle Real Application Testing

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 before production deployment, Oracle Real Application Testing minimizes the risk of instabilities associated with changes.

Oracle Real Application Testing comprises two components:

- Database Replay
- SQL Performance Analyzer

Database Replay and SQL Performance Analyzer are complementary solutions that can be used for real application testing. Depending on the nature and impact of the system change, and on which system the test will be performed (production or test), you can use either solutions to perform your testing.

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

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

> **See Also:**
>
> - Part I, "Database Replay" for information about using Database Replay

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 executions 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
- Configuration changes to the operating system or hardware
- Schema changes
- Changes to database initialization parameters
- Refreshing optimizer statistics
- SQL tuning actions

> **See Also:**
>
> - Part II, "SQL Performance Analyzer" for information about using SQL Performance Analyzer

Part I

Database Replay

Database Replay enables you to replay a full production workload on a test system to assess the overall impact of system changes. This part contains information about how to capture, preprocess, and replay a database workload using Database Replay, as well as how to analyze the results of a replayed workload.

Part I contains the following chapters:

- Chapter 2, "Introduction to Database Replay"
- Chapter 3, "Capturing a Database Workload"
- Chapter 4, "Preprocessing a Database Workload"
- Chapter 5, "Replaying a Database Workload"
- Chapter 6, "Analyzing Replayed Workload"

2

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 can 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 Oracle9*i* Database, contact Oracle Support for more information.

Analyzing the effect of system changes using Database Replay involves the following steps, as illustrated in Figure 2–1:

Figure 2–1   Database Replay Workflow
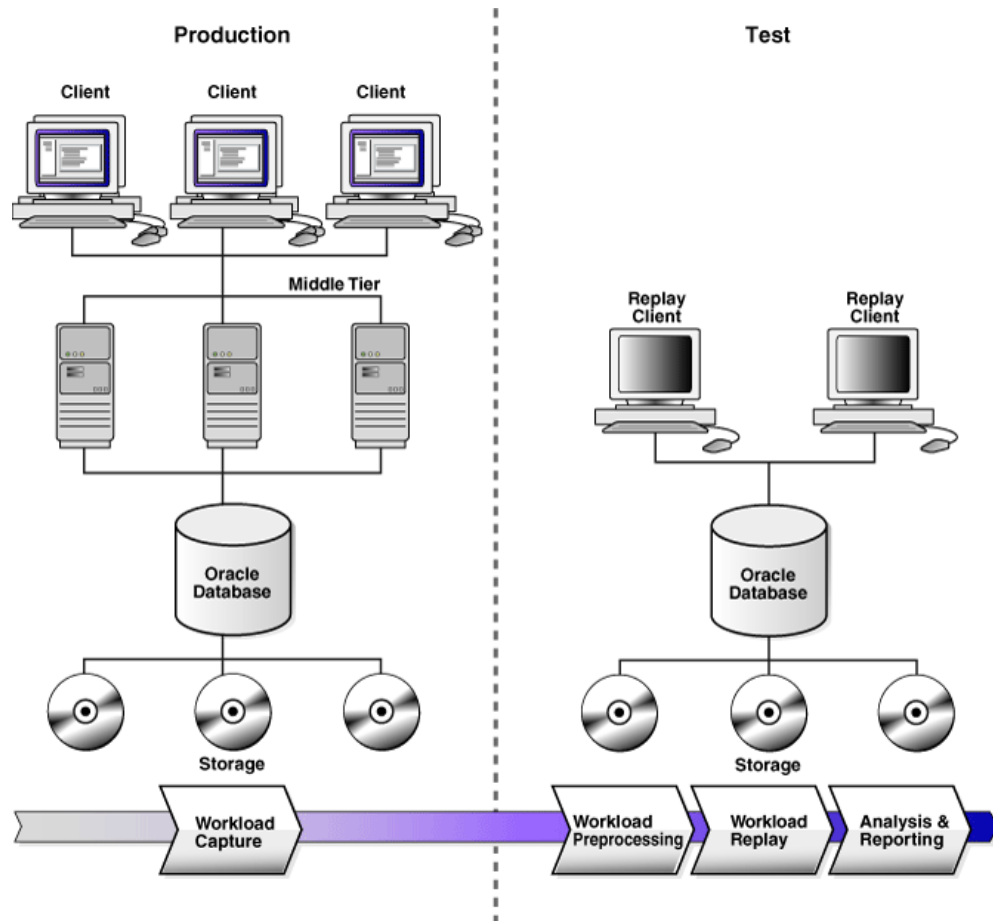
1. On the production system, capture the workload into capture files, as described in "Workload Capture" on page 2-2.

2. Copy the capture files to the test system and preprocess them, as described in "Workload Preprocessing" on page 2-3.

3. On the test system, replay the preprocessed files, as described in "Workload Replay" on page 2-3.

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 2-3.

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 3, "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 need to be preprocessed. Preprocessing transforms the captured data into replay files and 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 another 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 4, "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 data divergence and enable a more reliable analysis of the replay.

> **See Also:** Chapter 5, "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 report summary provides 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. 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.

For application-level validation, you should 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 once the workload directory object is backed up to another physical location.

> **See Also:** Chapter 6, "Analyzing Replayed Workload" for information about how to analyze data and performance divergence using Database Replay reports

3

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 2-2.

This chapter contains the following sections:

- Prerequisites for Capturing a Database Workload
- Workload Capture Options
- Workload Capture Restrictions
- Enabling and Disabling the Workload Capture Feature
- 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 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

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
- Defining the Workload Filters
- Setting Up the Capture Directory

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 data divergence when the workload is replayed. Any subsequent transactions with dependencies on the incomplete transactions may also generate errors during replay.

Before restarting the database, determine an appropriate time to shut down the production database prior to the workload capture time period 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 at 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 (RAC) configuration, workload capture is performed for the entire database. To enable a clean state before starting to capture the workload, all the instances need to be restarted. You can do this by:

1. Shutting down all the instances.

2. Restarting one of the instances.

3. Starting workload capture.

4. Restarting the rest of the instances.

> **See Also:** *Oracle Database Administrator's Guide* for information about restricting access to an instance at startup

Defining the Workload Filters

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. 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 captured in the workload, such as background processes that may already be running on the test system. For example, if the system where the workload will be replayed is running Oracle Enterprise Manager (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.

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.

For Oracle RAC, consider using a shared file system. Alternatively, you can set up capture directory paths that resolve to separate physical directories on each instance, but you will need to collect the capture files created in each of these directories into a single directory before preprocessing the workload capture.

Workload Capture Restrictions

The following types of client requests are not captured in a workload:

- Direct path load of data from external files using utilities such as SQL*Loader
- Shared server requests (Oracle MTS)
- Oracle Streams
- Advanced replication streams
- 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)

Enabling and Disabling the Workload Capture Feature

Oracle Database 10*g* Release 2 supports using Database Replay to capture a database workload that can be used to test database upgrades to Oracle Database 11*g* and subsequent releases. To use this feature, it must be enabled on the capture system running Oracle Database 10*g* Release 2 before a workload can be captured. By default, the workload capture feature is not enabled in Oracle Database 10*g* 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 10*g* Release 2.
>
> If you are capturing a database workload on a system running Oracle Database 11*g* 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 10*g* Release 2 (10.2) and cannot be used with subsequent releases.

To enable the workload capture feature on a system running Oracle Database 10*g* 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 10*g* 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

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. If for some reason Oracle Enterprise Manager is unavailable, you can capture database workloads using APIs, as described in "Capturing a Database Workload Using APIs" on page 3-13.

To capture a database workload using Enterprise Manager:

1. On the Software and Support page, under Real Application Testing, click **Database Replay**.

   The Database Replay page appears.

2. In the Go to Task column, click the icon that corresponds to the Capture Workload task.

   The Capture Workload: Plan Environment page appears.

3. Verify that all prerequisites are met before proceeding.

   For information about the prerequisites, see "Prerequisites for Capturing a Database Workload" on page 3-1.

For each verified prerequisite, check the box in the Acknowledge column. Once all prerequisites are verified, click **Next**.

The Capture Workload: Options page appears.

4. Select the workload capture options:

- Under Database Restart Options, select whether the database will be restarted before workload capture.

  It is recommended that the database be restarted before capturing a workload to enable a clean state for workload capture. Otherwise, potential problems may arise when replaying the workload. For more information, see "Restarting the Database" on page 3-2.

- Under Workload Filters, 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 Another Row** and enter the filter name, session attribute type, and attribute value in the corresponding fields. For more information, see "Defining the Workload Filters" on page 3-3.

After selecting the desired workload capture options, click **Next**.

The Capture Workload: Parameters page appears.

Capture Workload: Parameters

Database **database**   (Cancel) (Back) Step 3 of 5 (Next)
Logged In As **IMMCHAN**

Workload Capture Parameters

    * Capture Name   CAPTURE-database-20080805235757

    * Directory Object   [▼] (Create Directory Object)
                      Select a directory object to hold the captured workload. The selected directory object cannot already contain a captured
                      workload.

Database Shutdown Parameters

⦿ Immediate
    Rollback active transactions and disconnect all connected users.

◯ Transactional
    Disconnect all connected users after transactions have completed.

◯ Abort
    Instantaneous shutdown by aborting the database instance.

Database Startup Parameters

⦿ Startup using current spfile.

◯ Specify parameter file (pfile) on database host.

    [   ]

    Specify the fully qualified name for the pfile.

5. Define the parameters for the workload capture:

   ■ Under Workload Capture Parameters, in the Capture Name field, enter a name for the workload capture. In the Directory Object list, select the directory where the captured workload will be stored. You must select a directory that does not already contain a workload capture. For more information, see "Setting Up the Capture Directory" on page 3-3.

      To create a new directory object, click **Create Directory Object**. The Create Directory Object page appears. In the Name field, enter a name for the directory object. In the Path field, enter the path to the directory object. To test if the directory exists in the file system, click **Test File System**. If the directory does not exist, it will need to be created first.

   ■ Under Database Shutdown Parameters, select the type of database shutdown method to perform. This option only appears if the database will be restarted before workload capture. The types of available database shutdown methods include:

      – Immediate

         An immediate shutdown will roll back all active transactions and disconnect all connected users prior to shutting down the database.

      – Transactional

         A transactional shutdown will first complete all active transactions and then disconnect the connected user prior to shutting down the database.

      – Abort

         An abort shutdown will shut down the database instantaneously by aborting all active transactions.

   ■ Under Database Startup Parameters, select if the database will restart using the current default server parameter file (spfile) or a specific parameter file (pfile). To select a pfile, enter the fully qualified name for the pfile. This option only appears if the database will be restarted before workload capture.

   After defining the parameters for the workload capture, click **Next**.

The Capture Workload: Schedule page appears.

6. Under Job Parameters, define the parameters for the job:

   ■ In the Job Name field, enter a name for the job name or accept the system generated name.

   ■ In the Description field, enter an optional description of the job.

7. Under Job Schedule, specify a start time and duration for the workload capture:

   ■ Under Start, select whether the job will run immediately by selecting **Immediately**, or at a later time by selecting **Later** and specifying the desired time using the Date and Time fields.

   ■ Under Capture Duration, specify how long the job will run by selecting **Duration** and specifying the desired duration using the Hours and Minutes fields. To not specify a capture duration, select **Not Specified**. If a capture duration is unspecified, the job must be stopped manually.

8. Under Job Credentials, enter the host and database login credentials:

   ■ Under Host Credentials, enter the username and password for the host system.

   ■ Under Database Credentials, enter the username and password for the database that will used for the workload capture. The user needs the DBA privilege in order to capture the workload.

   Click **Next**.

   The Capture Workload: Review page appears.

9. Review the job settings for the workload capture that have been defined.

To run the job, click **Submit**. To make changes, click **Back**. To cancel the workload capture without saving changes, click **Cancel**.

10. Depending on the job settings that have been defined:

■ If the job is scheduled to start immediately and the database will be restarted, the Confirmation: Restart Database page appears.

To restart the database, click **Yes**.

The Information: Restart Database page appears while the database is being restarted. Once the database is restarted, the workload capture begins automatically. Click **Refresh**.

The View Workload Capture page appears.

■ If the job is scheduled to start immediately but the database will not be restarted, the workload capture begins automatically and the View Workload Capture page appears.

■ If the job is scheduled to start at a later time, the Database Replay page appears with a confirmation that the job has been successfully created.

Once workload capture begins, you can monitor the capture process using the View Workload Capture page, as described in "Monitoring Workload Capture Using Enterprise Manager" on page 3-9.

> **Tip:** After capturing a workload on the production system, you need to preprocess the captured workload, as described in Chapter 4, "Preprocessing a Database Workload".

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 or delete a completed workload capture

If for some reason Oracle Enterprise Manager is unavailable, you can monitor workload capture using views, as described in "Monitoring Workload Capture Using Views" on page 3-15.

This section contains the following topics:

- Monitoring an Active Workload Capture
- Stopping an Active Workload Capture
- Managing 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. On the Software and Support page, under Real Application Testing, click **Database Replay**.

   The Database Replay page appears.

2. Under Active Capture and Replay, select the workload capture you want to monitor and click **View**.

   The View Workload Capture page appears.

3. Under Summary, information about the workload capture is displayed.

4. To view the workload profile, click the **Workload Profile** tab.

Under Average Active Sessions, the 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).

Under Comparison, various statistics for the workload capture are displayed, including database time, average active sessions, user calls, transactions, connects, and application errors. The statistics for the total session activity are displayed in the Total column, and the statistics for the captured session activity are displayed in the Capture column. The Percentage of Total column displays the percentage of total session activity that are being captured in the workload.

To view the workload capture report, click **View Workload Capture Report**.

5. To view workload filters used by the workload capture, click the **Workload Filters** tab.

   Details about the workload filters used by the workload capture are displayed, including the workload filter name, type, session attribute, and value.

6. To return to the Database Replay page, click **OK**.

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. On the Software and Support page, under Real Application Testing, click **Database Replay**.

   The Database Replay page appears.

2. Under Active Capture and Replay, select the workload capture you want to stop and click **Stop**.

   The Confirmation page appears.

3. To confirm that you want to stop the workload capture, click **Yes**.

   Once the workload capture is stopped, the Export AWR Data page appears.

4. To export the Automatic Workload Repository (AWR) data, click **Yes**.

   The View Workload Capture page appears.

   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. If you choose not to export AWR data, click **No**. You can still export AWR data from a completed workload capture at a later time from the View Workload Capture History page.

   > **See Also:** *Oracle Database Performance Tuning Guide* for information about the AWR

Managing a Completed Workload Capture

This section describes how to manage a completed workload capture using Enterprise Manager.

To manage a completed workload capture:

1. On the Software and Support page, under Real Application Testing, click **Database Replay**.

The Database Replay page appears.

2. Click **View Workload Capture History**.

The View Workload Capture History page appears.

3. To delete a workload capture, select the workload capture and click **Delete**.

4. To export AWR data for a workload capture, select the workload capture and click **Export AWR Data**.

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.

5. To view details about a workload capture, select the workload capture and click **View**.

The View Workload Capture page appears.

6. Under Summary, information about the workload capture is displayed.

7. To view the workload profile, click the **Workload Profile** tab.

Under Average Active Sessions, the 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 will be shown only when there is Active Session History (ASH) data available for the capture period.

Under Comparison, various statistics for the workload capture are displayed, including database time, average active sessions, user calls, transactions, connects, and application errors. The statistics for the total session activity are displayed in the Total column, and the statistics for the captured session activity are displayed in the Capture column. The Percentage of Total column displays the percentage of total session activity that are being captured in the workload.

To view the workload capture report, click **View Workload Capture Report**.

8. To view workload filters used by the workload capture, click the **Workload Filters** tab.

Details about the workload filters used by the workload capture are displayed, including the workload filter name, type, session attribute, and value.

9. To return to the Database Replay page, click **OK**.

> **See Also:** *Oracle Database Performance Tuning Guide* for information about ASH

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 3-5.

Capturing a database workload using the DBMS_WORKLOAD_CAPTURE package involves:

- Adding and Removing Workload Filters
- Starting a Workload Capture
- Stopping a Workload Capture
- Exporting AWR Data for Workload Capture

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for information about the DBMS_WORKLOAD_CAPTURE package

Adding and Removing Workload Filters

This section describes how to add and remove workload filters. For information about using workload filters, see "Defining the Workload Filters" on page 3-3.

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.

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 3-1. You should also review the workload capture options, as described in "Workload Capture Options" on page 3-2.

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_RESTRICTED prior to 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 3-2.

To start the workload capture, use the START_CAPTURE procedure:

```
BEGIN
  DBMS_WORKLOAD_CAPTURE.START_CAPTURE (name => 'dec06_peak',
                          dir => 'dec06',
                          duration => 600);
END;
/
```

In this example, a workload named dec06_peak will be captured for 600 seconds and stored in the operating system defined by the database directory object named dec06.

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 optional 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.

Stopping a Workload Capture

To stop the 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 4, "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 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. The `EXPORT_AWR` procedure uses the `capture_id` required parameter, which specifies the ID of the capture whose AWR snapshots will be exported. This procedure will work 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.

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 3-9.

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 these views

4

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 transforms the captured data into replay files and creates all necessary metadata. 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 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 move all the capture files created in each of these directories into a single directory on which preprocessing will be performed.

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:

- Preprocessing a Database Workload Using Enterprise Manager

- Preprocessing a Database Workload Using APIs

> **Tip:** Before you can preprocess a captured workload, you need to capture the workload on the production system, as described in Chapter 3, "Capturing a Database Workload".

Preprocessing a Database Workload Using Enterprise Manager

This section describes how to preprocess a captured workload using Enterprise Manager. The primary tool for preprocessing workload captures is Oracle Enterprise Manager. If for some reason Oracle Enterprise Manager is unavailable, you can preprocess workload captures using the APIs, as described in "Preprocessing a Database Workload Using APIs" on page 4-3.

To preprocess a captured workload using Enterprise Manager:

1. On the Software and Support page, under Real Application Testing, click **Database Replay**.

   The Database Replay page appears.

2. In the Go to Task column, click the icon that corresponds to the Preprocess Captured Workload task.

   The Preprocess Captured Workload page appears.

3. In the Directory Object list, select a directory that contains the captured workload that you want to preprocess.

   After a directory is selected, the Preprocess Captured Workload page will be refreshed to display the Capture Summary section, which contains information about the captured workload in the selected directory.

   To view additional details about the captured workload, expand **Capture Details**. The expanded Capture Details section displays the workload profile and details for the captured workload.

4. Click **Preprocess Workload**.

   The Preprocess Captured Workload: Database Version page appears.

5. Ensure that the current database version displayed matches the database version on the intended replay system and click **Next**.

Preprocessing must be performed on a system that is running the same version of Oracle Database as the replay system.

The Preprocess Captured Workload: Schedule page appears.

6. Define the parameters for the preprocessing job.

- Under Job Parameters, enter a name and a description for the job.

- Under Start, select whether the job will run immediately by selecting **Immediately**, or at a later time by selecting **Later** and specifying the desired time using the Date and Time fields.

- Under Host Credentials, enter the user name and password information for the database host that will used for the preprocessing.

After defining the job parameters, click **Next**.

The Preprocess Captured Workload: Review page appears.

7. Review the selected options for the preprocessing job.

To preprocess the captured workload, click **Submit**. To make changes, click **Back**. To cancel preprocessing without saving changes, click **Cancel**.

> **Tip:** After preprocessing a captured workload, you can replay it on the test system, as described in Chapter 5, "Replaying a Database Workload".

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 "Preprocessing a Database Workload Using Enterprise Manager" on page 4-1.

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 5, "Replaying a Database Workload".

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for information about the DBMS_WORKLOAD_REPLAY package

5

Replaying a Database Workload

This chapter describes how to replay a database workload on the test system. 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 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 Views

> **Tip:** Before you can replay a database workload, you must first:
>
> - Capture the workload on the production system, as described in Chapter 3, "Capturing a Database Workload"
> - Preprocess the captured workload, as described in Chapter 4, "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 data 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 3-1.

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 data 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.

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

- Remapping Connections

- Specifying Replay Options

- 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 replay 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 need to be configured so that Web services accessed during the workload capture will point to the proper URLs during replay.

- E-mails

  To avoid resending E-mail notifications during replay, the mail server on 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.

Remapping Connections

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 (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.

Specifying Replay Options

After the database is restored and connections are remapped, you can set the following replay options as appropriate:

- Preserving COMMIT Order
- Controlling Session Logins
- Controlling Think Time

Preserving COMMIT Order

The `synchronization` parameter controls whether the `COMMIT` order in the captured workload will be preserved during replay. By default, this option is enabled to preserve the `COMMIT` order in the captured workload during replay. All transactions will be executed only after all dependent transactions have been committed.

You can disable this option, but the replay will likely yield significant data divergence. However, this may be desirable if the workload consists primarily of independent transactions, and divergence during unsynchronized replay is acceptable.

Controlling Session Logins

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 Think Time

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.

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)

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` and `password` specify the username and password used to connect to the host where the wrc executable is installed. The parameter `server` specifies the server where the wrc executable is installed. 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 calibration 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 thread 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 11.1.0.7.0 – Production on Thu March 27
14:06:33 2008

Copyright (c) 1982, 2008, 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 in conjunction 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 both 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/oracle@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. At this point, workload replay can begin. 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 11.1.0.7.0 - Production on Thu March 27 13:44:48
2008

Copyright (c) 1982, 2008, 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 for some reason Oracle Enterprise Manager is unavailable, you can also replay database workloads using APIs, as described in "Replaying a Database Workload Using APIs" on page 5-16.

To replay a database workload using Enterprise Manager:

1. On the Software and Support page, under Real Application Testing, click **Database Replay**.

The Database Replay page appears.

2. In the Go to Task column, click the icon that corresponds to the Replay Workload task.

The Replay Workload page appears.

3. In the Directory Object list, select a directory that contains the preprocessed workload that you want to replay.

After a directory is selected, the Replay Workload page will be refreshed to display the Capture Summary and the Replay History sections. For more information, see "Setting Up the Replay Directory" on page 5-2.

The Capture Summary section displays information about the preprocessed workload capture in the selected directory. To view additional details about the workload capture, expand **Capture Details**. The expanded Capture Details section displays the workload profile and workload filters used during the workload capture.

4. Click **Set Up Replay**.

The Replay Workload: Prerequisites page appears.

5. Verify that all prerequisites are met before proceeding.

If you are replaying the workload on a test system, ensure that the test system is properly prepared for replay. For more information, see "Setting Up the Test System" on page 5-1.

Once all prerequisites are completed, click **Continue**.

The Replay Workload: References to External Systems page appears.

6. Verify potential references to all external systems and modify any invalid references.

Use the links available on the Replay Workload: References to External Systems page to verify the database links, directory objects, and Oracle Streams that may be referenced during the workload capture process. There may be other references to external systems that are not included in these categories. For more information, see "Resolving References to External Systems" on page 5-2.

Once all references to external systems have been verified and modified as necessary, click **Continue**.

The Replay Workload: Choose Initial Options page appears.

7. In the Replay Name field, you may enter a name for the replay, or simply use the name generated by the system.

8. Under Initial Options, select whether to use default replay options or replay options from a previous replay (if one is available). If more than one previous replay exist, select the replay you want to use from the Replay Name list.

    Click **Next**.

    The Replay Workload: Customize Options page appears.

9. Remap captured connection strings to connection strings that point to the replay system.

    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 will be used by the replay client.

For more information, see "Remapping Connections" on page 5-3.

10. Specify the replay options using the replay parameters.

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 5-3.

	Connection Mappings	**Replay Parameters**	

Some replay parameters can be modified to change the behavior of the replay. Refer to system documentation for more information.

Name	Description	Value
synchronization	This parameter determines if synchronization will be used during workload replay. If this parameter is set to TRUE, the COMMIT order in the captured workload will be preserved during replay and all replay actions will be executed only after all dependent COMMIT actions have completed. The default value is TRUE.	TRUE ▾
connect_time_scale	This parameter scales the elapsed time from when the workload capture started to when the session connects with the specified value and is interpreted as a % value. The default value is 100.	100 %
think_time_scale	This 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.	100 %
think_time_auto_correct	This parameter reduces the think time if workload replay goes slower than workload capture. If this parameter is set to TRUE, the system will correct the think time (based on the think_time_scale parameter) between calls when user calls take longer to complete during replay than during capture. The default value is TRUE.	TRUE ▾

After setting the replay parameters, click **Next**.

The Replay Workload: Prepare Replay Clients page appears.

11. Ensure that replay clients are prepared for replay.

Before proceeding, the replay clients need to be prepared. For more information, see "Setting Up Replay Clients" on page 5-4.

After all replay clients are ready to start, click **Next**.

The Replay Workload: Wait for Client Connections page appears.

Replay Workload: Wait for Client Connections

Database	database		Cancel	Back	Step 4 of 5	Next
Capture Name	CAPTURE-database-20080806013843					
Logged In As	IMMCHAN					

The database is waiting for connections from the Replay Clients. Start the Replay Clients now.

When all the Replay Clients have connected, proceed to the next step to continue the replay setup.

The database is waiting for connections from Replay Clients.

⚠ This operation may take some time to complete. If you close this browser window or navigate to a different page, your place in the replay process will not be saved.

Client Connections

SID	Host	OS Process ID	OS User Name	Program
No items found				

12. Start the replay clients.

For information about starting replay clients, see "Setting Up Replay Clients" on page 5-4.

As replay clients are started, the replay client connections will be displayed under Client Connections. When all replay clients have connected, click **Next**.

The Replay Workload: Review page appears.

13. Review the options and parameters that have been defined for the workload replay.

Before starting replay, reset the system clock to a value that is as close to the capture start time as possible. This minimizes any data divergence that may result from replaying a time-sensitive workload. For more information, see "Resetting the System Time" on page 5-2.

To begin replay, click **Submit**. If no replay clients are connected, this button will be disabled. To make changes, click **Back**. To cancel replay without saving changes, click **Cancel**.

Once the replay is started, the View Workload Replay page appears. For information about monitoring an active workload replay, see "Monitoring an Active Workload Replay" on page 5-12.

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 for some reason Oracle Enterprise Manager is unavailable, you can monitor workload replay using views, as described in "Monitoring Workload Replay Using Views" on page 5-19.

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. On the Software and Support page, under Real Application Testing, click **Database Replay**.

   The Database Replay page appears.

2. Under Active Capture and Replay, select the workload replay you want to monitor and click **View**.

   The View Workload Replay page appears.

3. Under Summary, information about the workload replay is displayed.

4. To view the workload profile, click the **Workload Profile** tab.

   Under Elapsed Time Comparison, the chart shows how much time it has taken to replay the same workload compared to the elapsed time during the workload capture. If the Replay bar is shorter than the Capture bar, the replay system is processing the workload faster than the capture system. If the workload replay is still running, a portion of the Capture bar will be shown as not yet replayed.

   Under Divergence, any error and data discrepancies between the replay system and the capture system are displayed. This information can be used as a measure of the replay quality.

To view a detailed comparison of the workload during capture and replay, expand **Detailed Comparison**.

▼Detailed Comparison			
	Capture	Replay	Percentage of Capture
Duration (hh:mm:ss)	00:04:07	00:03:07	75.71
Database Time (hh:mm:ss)	00:00:01	00:00:03	300.00
Average Active Sessions	0.00	0.01	N/A
User Calls	85	65	76.47

This section displays the following information:

- Duration

  The duration that was captured in a workload is compared to the amount of time it is taking to replay the workload. In the Capture column, the duration of the time period that was captured is shown. In the Replay column, the amount of time it is taking to replay the workload is shown. The Percentage of Capture column shows the percentage of the captured duration that it is taking to replay the workload. If the value is under 100 percent, the replay system is processing the workload faster than the capture system. If the value is over 100 percent, the replay system is processing the workload slower than the capture system.

- Database time

  The database time that is consumed in the time period that was captured is compared to the amount of database time that is being consumed while replaying the workload.

- Average active sessions

  The number of average active sessions captured in the workload is compared to the number of average active session that are being replayed.

- User calls

  The number of user calls captured in the workload is compared to the number of user calls that are being replayed.

To view the workload capture report, click **View Workload Replay Report**.

5. To view the connection strings used in the capture and the replay systems, click the **Connection Mappings** tab.

6. To view replay parameters used by the workload replay, click the **Replay Parameters** tab.

7. To stop the workload replay, click **Stop Replay**.

8. To return to the Database Replay page, click **OK**.

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. On the Software and Support page, under Real Application Testing, click **Database Replay**.

   The Database Replay page appears.

2. In the Go to Task column, click the icon that corresponds to the Replay Workload task.

   The Replay Workload page appears.

3. In the Directory Object list, select a directory that contains the preprocessed workload that was used for the replay.

   After a directory is selected, the Replay Workload page will be refreshed to display the Capture Summary and the Replay History sections.

4. The Replay History section displays previous replays of the workload capture. To view details about a previous replay, select the replay and click **View**.

   The View Workload Replay page appears.

5. Under Summary, information about the workload replay is displayed.

6. To view the workload profile, click the **Workload Profile** tab.

   Under Elapsed Time Comparison, the chart shows how much time it has taken to replay the same workload compared to the elapsed time during the workload capture. If the Replay bar is shorter than the Capture bar, the replay system is processing the workload faster than the capture system.

   Under Divergence, any error and data discrepancies between the replay system and the capture system are displayed. This information can be used as a measure of the replay quality.

   To view a detailed comparison of the workload during capture and replay, expand **Detailed Comparison**. This section displays the following information:

   - Duration

     The duration that was captured in a workload is compared to the amount of time it took to replay the workload. In the Capture column, the duration of the time period that was captured is shown. In the Replay column, the amount of time it took to replay the workload is shown. The Percentage of Capture column shows the percentage of the captured duration that it took to replay the workload. If the value is under 100 percent, the replay system processed the workload faster than the capture system. If the value is over 100 percent, the replay system processed the workload slower than the capture system.

   - Database time

     The database time that is consumed in the time period that was captured is compared to the amount of database time that is consumed when replaying the workload.

   - Average active sessions

     The number of average active sessions captured in the workload is compared to the number of average active session that are replayed.

   - User calls

     The number of user calls captured in the workload is compared to the number of user calls that are replayed.

   To view the workload capture report, click **View Workload Replay Report**.

7. To view the connection strings used in the capture and the replay systems, click the **Connection Mappings** tab.

8. To view replay parameters used by the workload replay, click the **Replay Parameters** tab.

9. To run a report, click the **Report** tab.

There are several types of reports you can run for a completed workload replay:

- Workload Replay

  The Workload Replay report contains information that can be used to measure data and performance divergence between the capture system and the replay system. To run this report, under Workload Replay Report, click **Run Report**. For information about using the Workload Replay report, see "Using a Workload Replay Report" on page 6-5.

- AWR Compare Period

  The AWR Compare Period report can be used to compare the AWR data in one workload capture or replay with another. Before running this report, AWR data for the captured or replayed workload must have been previously exported. To run this report, under AWR Compare Period Report, select the first and second workload captures or replays you want to compare and click **Run Report**. If AWR data is not previously exported from the captured or replayed workload, you will be prompted to import the AWR data before continuing. For more information about the AWR Compare Period report, see *Oracle Database 2 Day + Performance Tuning Guide*.

- AWR

  The AWR report shows the AWR data contained in a workload that was captured or replayed. Before running this report, AWR data must have been previously exported from the captured or replayed workload. To run this report, under AWR Report, select the workload capture or replay for which you want to generate an AWR report and click **Run Report**. If AWR data is not previously exported from the captured or replayed workload, you will be

prompted to import the AWR data before continuing. For more information about the AWR report, see *Oracle Database Performance Tuning Guide*.

- ASH

  The ASH report contains active session history (ASH) information for a specified duration of a workload that was captured or replayed. Before running this report, AWR data must have been previously exported from the captured or replayed workload. To run this report, under ASH Report, select the workload capture or replay for which you want to generate an ASH report. Specify the duration using the Start Date, Start Time, End Date, and End Time fields. You can also apply filters using the Filter field. Once the duration and filters are specified, click **Run Report**. If AWR data is not previously exported from the captured or replayed workload, you will be prompted to import the AWR data before continuing. For more information about the ASH report, see *Oracle Database 2 Day + Performance Tuning Guide*.

The Report window opens while the report is being generated. Once the report is generated, you can save the report by clicking **Save to File**.

10. To return to the Database Replay page, click **OK**.

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 5-7.

Replaying a database workload using the DBMS_WORKLOAD_REPLAY package is a multi-step process that involves:

- Initializing Replay Data
- Connection Remapping
- Setting Workload Replay Options
- Starting a Workload Replay
- Stopping a Workload Replay
- Exporting 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 4-3 for information about preprocessing a workload capture
>
> - "Setting Up the Test System" on page 5-1 for information preparing the test system

Connection Remapping

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 "Remapping Connections" on page 5-3.

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.

Setting Workload Replay Options

After the replay data is initialized and the connections are appropriately 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 5-2.

To prepare workload replay on the replay system, use the PREPARE_REPLAY procedure:

```
BEGIN
  DBMS_WORKLOAD_REPLAY.PREPARE_REPLAY (synchronization => TRUE);
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.

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 TRUE, the COMMIT order in the captured workload will be preserved during replay and all replay actions will be executed only after all dependent COMMIT actions have completed. The default value is TRUE.

- The connect_time_scale optional 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 optional 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 optional 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 and can be set to either TRUE or FALSE. The default value is TRUE.

For more information about setting these parameters, see "Specifying Replay Options" on page 5-3.

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 4-3

- Initialize the replay data, as described in "Initializing Replay Data" on page 5-16

- Specify the replay options, as described in "Setting Workload Replay Options" on page 5-17

- Start the replay clients, as described in "Starting Replay Clients" on page 5-5

To start a workload replay, use the START_REPLAY procedure:

```
BEGIN
  DBMS_WORKLOAD_REPLAY.START_REPLAY ();
END;
/
```

Stopping a Workload Replay

To stop 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. The EXPORT_AWR procedure uses the replay_id required parameter, which specifies the ID of the replay whose AWR snapshots will be exported. This procedure will work 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.

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 both workload captures and workload replays, 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 monitor workload replay divergence.

- 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 on these views

6

Analyzing Replayed Workload

This chapter describes how to generate and analyze Database Replay reports. There are two types of reports for Database Replay: the workload capture report and the workload replay report. The workload capture report contains captured workload statistics, information about the top session activities that were captured, and any workload filters used during the capture process. The workload replay report contains information that can be used to measure performance differences between the capture system and the replay system.

This section contains the following topics:

- Generating a Workload Capture Report Using Enterprise Manager

- Generating a Workload Capture Report Using APIs

- Using a Workload Capture Report

- Generating a Workload Replay Report Using Enterprise Manager

- Generating a Workload Replay Report Using APIs

- Using a Workload Replay Report

> **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.

Generating a Workload Capture Report 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 a Workload Capture Report Using APIs" on page 6-2.

To generate a workload capture report using Enterprise Manager:

1. On the Software and Support page, under Real Application Testing, click **Database Replay**.

    The Database Replay page appears.

2. Click **View Workload Capture History**.

    The View Workload Capture History page appears.

3. Select the workload capture for which you want to run a workload capture report and click **View**.

   The View Workload Capture page appears.

4. To view the workload capture report, click **View Workload Capture Report**.

   The Report window opens while the report is being generated.

5. Once the report is generated, you can save the report by clicking **Save to File**.

   For information about how to use a workload capture report, see "Using a Workload Capture Report" on page 6-3.

Generating a Workload Capture Report 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 "Generating a Workload Capture Report Using Enterprise Manager" on page 6-1.

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 parameter 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 use a workload capture report, see "Using a Workload Capture Report" on page 6-3.

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for information about the DBMS_WORKLOAD_CAPTURE package

Using a Workload Capture Report

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 are 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

Generating a Workload Replay Report 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 a Workload Replay Report Using APIs" on page 6-4

To generate a workload replay report using Enterprise Manager:

1. On the Software and Support page, under Real Application Testing, click **Database Replay**.

   The Database Replay page appears.

2. In the Go to Task column, click the icon that corresponds to the Replay Workload task.

   The Replay Workload page appears.

3. In the Directory Object list, select a directory that contains the preprocessed workload that was used for the replay for which you want to generate a workload replay report.

   After a directory is selected, the Replay Workload page will be refreshed to display the Capture Summary and the Replay History sections.

4. Under Replay History, select the replay for which you want to generate a workload replay report and click **View**.

The View Workload Replay page appears.

5. Click **View Workload Replay Report**.

For information about using the workload replay report, see "Using a Workload Replay Report" on page 6-5.

Generating a Workload Replay Report 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 "Generating a Workload Replay Report Using Enterprise Manager" on page 6-3.

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:

To generate a workload replay report, use the 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 using the workload replay report, see

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for information about the DBMS_WORKLOAD_REPLAY package

Using a Workload Replay Report

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 data 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

Part II

SQL Performance Analyzer

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

Part II contains the following chapters:

7

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 needs to 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 system-wide 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 7–1:

Figure 7–1   SQL Performance Analyzer Workflow
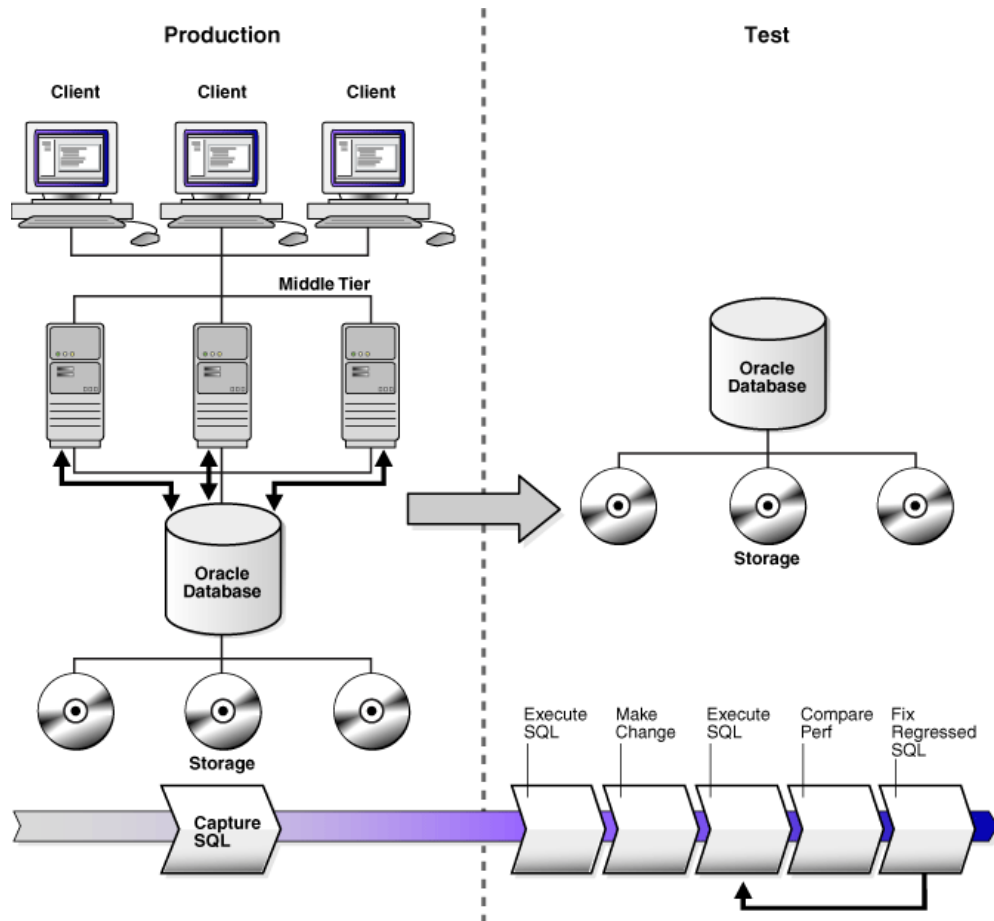
1. Capture the SQL workload that you intend to analyze and store it in a SQL tuning set, as described in "Capturing the SQL Workload" on page 7-3.

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 7-4.

3. On the test system, create a SQL Performance Analyzer task, as described in "Creating a SQL Performance Analyzer Task" on page 7-4.

4. Build the pre-change SQL trial by executing the SQL statements stored in the SQL tuning set, as described in "Measuring the Pre-Change SQL Performance" on page 7-5

5. Perform the system change, as described in "Making a System Change" on page 7-6

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 7-6

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 7-6

8. Tune any regressed SQL statements that are identified, as described in "Fixing Regressed SQL Statements" on page 7-7.

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.

Capturing the SQL Workload

Before running the 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), 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 Performance 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 Performance 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 and represents a particular test execution or explain plan operation performed by SQL Performance Analyzer when testing a system change. When creating a SQL Performance Analyzer task, you will need to select a SQL tuning set as its input source. The SQL tuning set remains constant in the SQL Performance Analyzer task and is executed in isolation during each SQL trial. Thus, performance differences between trials are caused by environmental differences.

> **See Also:**
>
> - Chapter 8, "Creating an Analysis Task" for information about how to create a SQL Performance Analyzer task

Measuring the Pre-Change SQL Performance

Execute the SQL workload to create a pre-change SQL trial before making the system change. Executing a SQL workload runs each of the SQL statements contained in the workload to completion. Each SQL statement in the SQL tuning set is executed once—one at a time—separately from other SQL statements without preserving their initial order of execution or concurrency. To avoid a potential impact to the database, DDLs are not supported; only the query portion of DMLs are executed. During execution, SQL Performance Analyzer generates execution plans and computes execution statistics for each SQL statement in the workload.

Depending on its size, executing a SQL workload can be time and resource intensive. When executing a SQL workload, you can choose to generate execution plans only, without collecting execution statistics. This technique shortens the time to run the execution 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.

Another method to execute a SQL workload is to run SQL Performance Analyzer on a system running Oracle Database 11*g*, and remotely execute the SQL statements on a separate database using a database link that you specify. SQL Performance Analyzer will establish a connection to the remote database using the specified database link and execute the SQL statements on that database, collect the execution plans and runtime 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 runtime statistics are stored in a SQL trial.

See Also:

- for information about how to measure the pre-change performance

- 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, execute the SQL workload again to create post-change SQL trial. SQL Performance Analyzer generates execution plans and computes execution statistics for each SQL statement in the workload a second time, resulting in 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:

- 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 response time of the SQL workload and 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 runtime statistics, including:

- CPU time

- Buffer gets

- Disk reads

- Disk writes

- 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.

See Also:

- for information about how to compare performance measurements

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 11, "Comparing SQL Trials" for information about fixing regressed SQL statements

8

Creating an Analysis Task

Once you have captured a SQL workload that you want to analyze into a SQL tuning set, 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 and represents a particular test execution or explain plan operation performed by SQL Performance Analyzer when testing a system change. When creating a SQL Performance Analyzer task, you will need to select a SQL tuning set as its input source. The SQL tuning set remains constant in the SQL Performance Analyzer task and is executed in isolation during each SQL trial. Thus, performance differences between trials are caused by environmental differences. For more information, see "Creating a SQL Performance Analyzer Task" on page 7-4.

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

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 7-3.

Creating an Analysis Task Using Enterprise Manager

There are two types of workflow available in Oracle Enterprise Manager for creating a SQL Performance Analyzer task:

- Parameter change workflow

  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 8-2.

■ Guided workflow

Use the guided workflow to compare SQL performance for all types of system changes other than a database initialization parameter change, as described in "Using the Guided Workflow" on page 8-5.

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 when the sort area size is increased from 1 MB to 2 MB.

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. A task is a container for the results of SQL replay trials. A SQL trial captures the execution performance of a SQL tuning set under specific environmental conditions. SQL Performance Analyzer then performs a second trial with the parameter set to the changed value. Any regression or change in performance are 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 8-5.

To create a SQL Performance Analyzer task using the parameter change workflow:

1. On the Software and Support page, under Real Application Testing, click **SQL Performance Analyzer**.

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

2. On the SQL Performance Analyzer page, click **Parameter Change**.

   The Parameter Change page appears.

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

4. 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.

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

6. 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.

7. 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 **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 of time.

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

8. 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.

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

- If you selected **Generate Plans** in Step 6, then select **Optimizer Cost**.

- If you selected **Execute SQLs** in Step 6, then select one of the following options:

    – **Elapsed Time**

    – **CPU Time**

    – **Buffer Gets**

    – **Disk Reads**

    – **Direct Writes**

    – **Optimizer Cost**

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

10. 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.

11. Click **Submit**.

A confirmation message 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 Last Run Status icon changes to a check mark.

SQL Performance Analyzer Tasks						
Delete						
Select	Name	Owner	Description	Last Run Status	Created	Last Modified ▽
⊙	SORT_TIME	SYS	Double the value of SORT_AREA_SIZE	✔	Apr 6, 2008 6:10:52 AM	Apr 6, 2008 6:13:02 AM

12. 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 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 workload comparisons

13. Click the icon in the Comparison Report column.

The SQL Performance Analyzer Task Result page appears.

14. Review the results of the performance analysis, as described in "Reviewing the SQL Performance Analyzer Report Using Oracle Enterprise Manager" on page 11-3.

15. In cases when regression are identified, click the icon in the SQL Tune Report column to view a SQL tuning report.

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 1-2.

> **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 8-2.

To create a SQL Performance Analyzer task using the guided workflow:

1. On the Software and Support page, under Real Application Testing, click **SQL Performance Analyzer**.

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

2. On the SQL Performance Analyzer page, 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 1-2.

   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.

3. 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.

4. In the Name field, enter the name of the task.

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

6. 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.

7. 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.

8. 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 9, "Creating a Pre-Change SQL Trial".

Creating an Analysis Task Using APIs

This section describes how to create a new 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 7-3.

Call the CREATE_ANALYSIS_TASK function to prepare the analysis of a SQL tuning set 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 9, "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

9

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. For information about setting up the database environment, see "Setting Up the Test System" on page 7-4.

Once the environment on the test system is properly configured, you can build the pre-change version of performance data by executing the SQL workload before performing the system change. SQL Performance Analyzer will store the results from executing the SQL statements in a pre-change SQL trial. For more information, see "Measuring the Pre-Change SQL Performance" on page 7-5

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 8, "Creating an Analysis Task".

Creating a Pre-Change SQL Trial Using Enterprise Manager

To collect the pre-change SQL performance data using Oracle Enterprise Manager:

1. On the Guided Workflow page, click the **Execute** icon for the Replay SQL Tuning Set 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

Cancel | Submit

SQL Trials capture execution performance of the SQL Tuning Set under a given optimizer environment.

SQL Performance Analyzer Task IMMCHAN.PERF_ANALYZER_TASK
SQL Tuning Set
* SQL Trial Name SQL_TRIAL_1207494888380
SQL Trial Description
Creation Method Execute SQLs Locally
Per-SQL Time Limit Unlimited
☑ TIP Time limit is on elapsed time of test execution of SQL.

Schedule
Time Zone America/Los_Angeles
⦿ Immediately
◯ Later
Date Apr 6, 2008
(example: Apr 6, 2008)
Time 7 : 11 : 00 ⦿ 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 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.

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

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

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 **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 of time.

- 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 10, "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.EXECUTE_ANALYSIS_TASK` procedure.

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 either 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.

- 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.

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 10, "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

10

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 7-6.

"SQL Performance Analyzer" on page 1-2 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 7-6.

This chapter described 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 9, "Creating a Pre-Change SQL Trial".

Creating a Post-Change SQL Trial Using Oracle Enterprise Manager

To collect the post-change SQL performance data using Oracle Enterprise Manager:

1. On the Guided Workflow page, click the **Execute** icon for the Replay SQL Tuning Set in Changed Environment step.

   The Create Replay 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 **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 of time.

   - 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 11, "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.EXECUTE_ANALYSIS_TASK` procedure.

Call the `EXECUTE_ANALYSIS_TASK` procedure using the parameters described in "Creating a Pre-Change SQL Trial Using APIs" on page 9-3. Be sure to specify a different value for `execution_name`.

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 11, "Comparing SQL Trials".

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* to learn about the `DBMS_SQLPA.EXECUTE_ANALYSIS_TASK` function

11

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 after the system change. For more information, see "Comparing Performance Measurements" on page 7-6.

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 10, "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

To analyze SQL performance before and after the system change using Oracle 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.

In this example, the SQL_TRIAL_1207494888380 and SQL_TRIAL_
1207499034916 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**

   - **Buffer Gets**

   - **Disk Reads**

   - **Direct Writes**

   - **Optimizer Cost**

   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, as described in "Reviewing the SQL Performance Analyzer Report Using Oracle Enterprise Manager" on page 11-3.

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 11–1 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 11–1   SQL Performance Analyzer Report
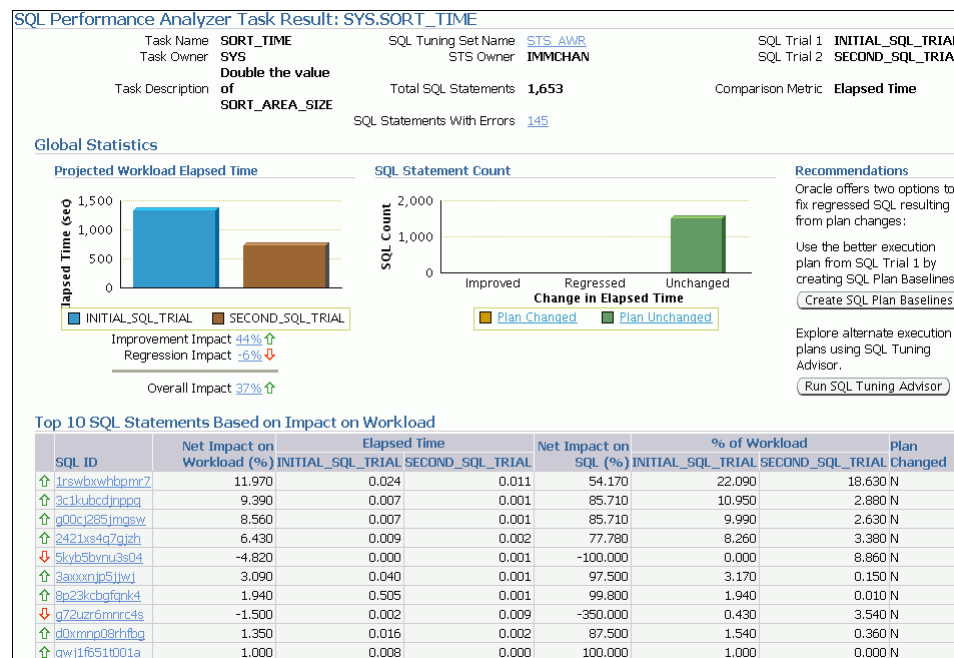
> **Tip:**   Before you can view the SQL Performance Analyzer report, compare the pre-change version of performance data with the post-change version, as described in "Comparing SQL Trials Using Oracle Enterprise Manager" on page 11-1

To generate and review the SQL Performance Analyzer report:

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

The SQL Performance Analyzer Task Result page appears.

2. Review the general information about the performance analysis, as described in "Reviewing the SQL Performance Analyzer Report: General Information" on page 11-4.

3. Review general statistics, as described in "Reviewing the SQL Performance Analyzer Report: Global Statistics" on page 11-4.

4. Optionally, review the detailed statistics, as described in "Reviewing the SQL Performance Analyzer Report: Global Statistics Details" on page 11-5.

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 Result page, review the summary at the top of the page, which includes the following information.

Task Name	SORT_TIME	SQL Tuning Set Name	STS_AWR	SQL Trial 1	INITIAL_SQL_TRIAL
Task Owner	SYS	STS Owner	IMMCHAN	SQL Trial 2	SECOND_SQL_TRIAL
Task Description	Double the value of SORT_AREA_SIZE	Total SQL Statements	1,653	Comparison Metric	Elapsed Time
		SQL Statements With Errors	145		

This summary includes the following information:

- The name, owner, and description of the SQL Performance Analyzer task

- The name and owner of the SQL tuning set

- The total number of SQL statements in the tuning set and the number of failing statements

- 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 SQL Performance Analyzer Task Result page appears.

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 11-4.

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.

   The chart shows the two trials on the x-axis and the elapsed time (in seconds) on the y-axis.
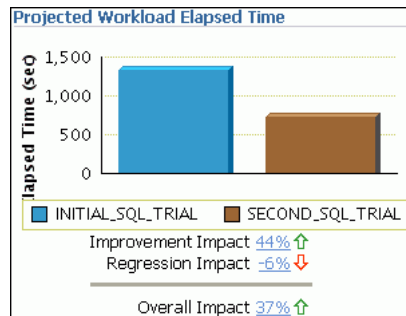
   The most important statistic is the overall impact, which is given as a percentage. The overall impact is the difference between the improvement impact and the regression impact. You can click the link for any impact statistic to obtain more details, as described in "Reviewing the SQL Performance Analyzer Report: Global Statistics Details" on page 11-5.

   In this example, the improvement impact is 44%, while the regression impact is -6%, so the overall impact of the system change is an improvement of approximately 37%.

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.
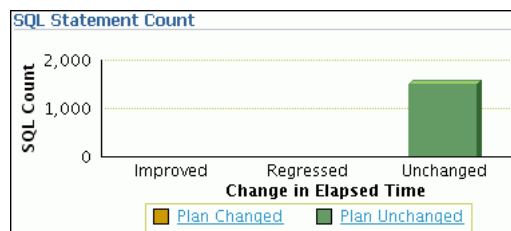
   This chart enables you to quickly weigh the relative performance of the SQL statements. You can click any bar in the chart to obtain more details, as described in "Reviewing the SQL Performance Analyzer Report: Global Statistics Details" on page 11-5.

   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.

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.

   - % of Workload

     This column indicates the percentage of the total workload consumed by this SQL statement.

   - Plan Changed

     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 CPU time, buffer gets, and optimizer cost.

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.

Creating SQL Plan Baselines

Creating SQL plan baselines enables the optimizer to avoid performance regressions by using better execution plans.

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

Running SQL Tuning Advisor explores alternate execution plans that may prevent performance regressions.

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 Using APIs

- Tuning Regressed SQL Statements 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 9-3.

     - 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 10-3 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`, and `optimizer_cost`.

   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 `DBMS_SQLPA.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, and XML.

- Set the level parameter to specify the format of the recommendations. Possible values include TYPICAL (default), BASIC, IMPROVED, REGRESSED, CHANGED_PLANS, ERRORS, and ALL.

- 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.

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

```
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 Using APIs" on page 11-10.

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 Using APIs

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 11–1 shows the General Information section of a sample report.

Example 11–1   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:                         After Change Execution:
 ------------------------------------------       ------------------------------------------
 Execution Name      : my_exec_BEFORE_change      Execution Name      : my_exec_AFTER_change
 Execution Type      : TEST EXECUTE               Execution Type      : TEST EXECUTE
 Description         :                            Description         :
 Scope               : COMPREHENSIVE              Scope               : COMPREHENSIVE
 Status              : COMPLETED                  Status              : COMPLETED
 Started             : 05/21/2007 11:22:06        Started             : 05/21/2007 11:25:56
 Last Updated        : 05/21/2007 11:24:01        Last Updated        : 05/21/2007 11:28:30
 Global Time Limit   : 1800                       Global Time Limit   : 1800
 Per-SQL Time Limit  : UNUSED                     Per-SQL Time Limit  : UNUSED
 Number of Errors    : 0                          Number of Errors    : 0


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

In Example 11–1, 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 11–2 shows the Overall Performance Statistics subsection of a sample report.

Example 11–2   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%. 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:

- Weight, 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 11–3 shows the Performance Statistics of SQL Statements subsection of a sample report. The report has been altered slightly to fit on the page.

Example 11–3   Performance Statistics of SQL Statements

```
SQL Statements Sorted by their Absolute Value of Change Impact on the Workload
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
|        |        |              | Impact | Metric  | Metric  | Impact | % Wrkld | % Wrkld | Plan   |
```

obj_id	sql_id	Wrkld	Before	After	on SQL	Before	After	Change
205	73s2sgy2svfrw	29.01%	1681683	220590	86.88%	33.39%	8.42%	y
206	gq2a407mv2hsy	29.01%	1681683	220590	86.88%	33.39%	8.42%	y
204	2wtgxbjz6u2by	-10.08%	1653012	2160529	-30.7%	32.82%	82.48%	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 11–4 shows an example of the Errors subsection of a SQL Performance Analyzer report.

Example 11–4   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
- Single Execution Statistics
- Execution Plans

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

Example 11–5 shows the SQL Details subsection of a sample report.

Example 11–5   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 11–5, the report summarizes the regressed SQL statement whose ID is
`2wtgxbjz6u2by` and corresponding object ID is `204`.

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

Example 11–6 shows the Single Execution Statistics subsection of a sample report.

Example 11–6   Single Execution Statistics

```
Execution Statistics:
---------------------------

-------------------------------------------------------------------------------------
|             | Impact on | Value   | Value   | Impact     | % Workload | % Workload |
| Stat Name   | Workload  | Before  | After   | on SQL     | Before     | 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 |            |            |            |
-------------------------------------------------------------------------------------

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 11–7 shows the Execution Plans subsection of a sample report.

Example 11–7   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 |
--------------------------------------------------------------------------------------------------
```

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 or SQL plan baselines to implement a point solution for them.

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.

> **See Also:**
>
> - *Oracle Database Performance Tuning Guide* for information about using the SQL Tuning Advisor
>
> - *Oracle Database Performance Tuning Guide* for information about using SQL plan baselines

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.

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

12

Testing a Database Upgrade

This chapter describes how to use SQL Performance Analyzer in a database upgrade. For information about using SQL Performance Analyzer in other cases, see "SQL Performance Analyzer" on page 1-2.

SQL Performance Analyzer supports testing database upgrades from Oracle9*i* and later releases to Oracle Database 10*g* Release 2 or newer releases. The methodology used to test a database upgrade from Oracle Database 10*g* Release 1 and previous releases is slightly different from the one used to test a database upgrade from Oracle Database 10*g* Release 2 and later releases, so both methodologies are described in this chapter.

This chapter contains the following sections:

- Upgrading from Oracle Database 10g Release 1 and Older Releases
- Upgrading from Oracle Database 10g Release 2 and Newer Releases
- Tuning Regressed SQL Statements After Testing a Database Upgrade

Upgrading from Oracle Database 10*g* Release 1 and Older Releases

SQL Performance Analyzer supports testing database upgrades of Oracle9*i* and Oracle Database 10*g* Release 1 to Oracle Database 10g Release 2 and later releases by executing the SQL tuning set on the upgraded database remotely over a database link, as illustrated in Figure 12–1. 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 in Oracle9*i*, a SQL tuning set needs to be constructed so that it can be used as an input source for SQL Performance Analyzer if you are upgrading from Oracle9*i*.

Figure 12–1 SQL Performance Analyzer Workflow for Database Upgrade from Oracle9i to Oracle Database 10g Release 2
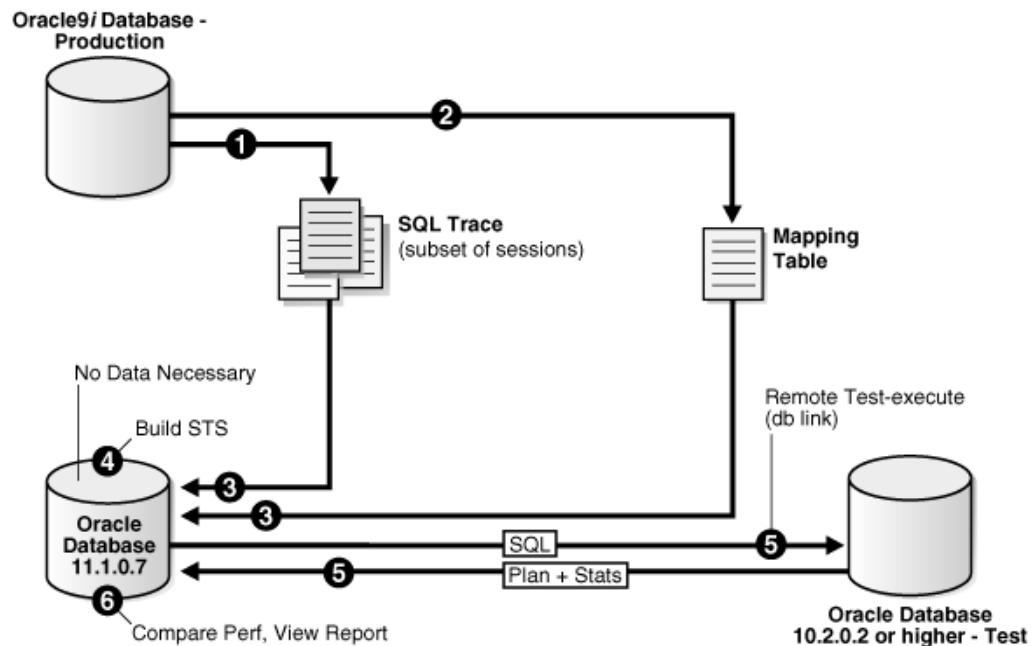
The production system which you are upgrading from should be running Oracle9*i* or Oracle Database 10*g* Release 1. The test system which you are upgrading to should be running Oracle Database 10*g* Release 2 or a newer release. The database version can be release 10.2.0.2 or higher. If you are upgrading to Oracle Database 10*g* 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.

To ensure that the analysis made by SQL Performance Analyzer is accurate, the test system should contain an exact copy of the data found on the production system because the performance on both systems will be compared to each other. Furthermore, the hardware configurations on both systems should also be as similar as possible.

Next, you will need to set up a separate SQL Performance Analyzer system running Oracle Database 11*g* Release 1 or a newer release. The database version should be release 11.1.0.7 or higher. 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 Oracle9*i* or Oracle Database 10*g* 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 12-3.

   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 12-4.

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 12-4.

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 12-4.

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 build a pre-upgrade SQL trial and a post-upgrade SQL trial:

   a. Create a new 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, as described in "Building the Pre-Upgrade SQL Trial for Oracle 10g Release 1 and Older Releases" on page 12-6.

   b. Remotely test execute the SQL statements on the test system over a database link to build a post-upgrade SQL trial, as described in "Building the Post-Upgrade SQL Trial Upgrading from Oracle Database 10g Release 1 and Older Releases" on page 12-8.

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. You can then repeat the process of executing the SQL tuning set and comparing its performance to a previous execution to test any fixes or additional changes made. Repeat these steps until you are satisfied with the outcome of the analysis.

Enabling SQL Trace on the Production System

Oracle9*i* 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 Performance 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 := 'mapping',
  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 12–1 describes the available parameters for the SELECT_SQL_TRACE function.

Table 12–1   DBMS_SQLTUNE.SELECT_SQL_TRACE Function Parameters

Parameter	Description
directory	Specifies the directory object pointing to the directory where the SQL trace files are stored.
file_name	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.
mapping_table_name	Specifies the name of the mapping table. The default mapping table name is mapping. 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.

Table 12–1 (Cont.) DBMS_SQLTUNE.SELECT_SQL_TRACE Function Parameters

Parameter	Description
`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 `MAXSB4`.

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_SQLTUNE` package

Building the Pre-Upgrade SQL Trial for Oracle 10*g* Release 1 and Older Releases

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.

You can build the pre-upgrade SQL trial using Oracle Enterprise Manager or APIs, as described in the following sections:

- Building the Pre-Upgrade SQL Trial for Oracle 10g Release 1 and Older Releases Using Enterprise Manager
- Building the Pre-Upgrade SQL Trial for Oracle 10g Release 1 and Older Releases Using APIs

Building the Pre-Upgrade SQL Trial for Oracle 10*g* Release 1 and Older Releases Using Enterprise Manager

Once the SQL tuning set is built, you can build a pre-upgrade SQL trial by running SQL Performance Analyzer on the SQL Performance Analyzer system.

To build the pre-upgrade SQL trial using Enterprise Manager:

1. On the Database Home page, click **Advisor Central**.

    The Advisor Central page appears.

2. Click **SQL Performance Analyzer**.

    The SQL Performance Analyzer page appears.

3. Click **Guided Workflow**.

    The Guided Workflow page appears.

4. Create a SQL Performance Analyzer task using the SQL tuning set as its input source:

    a. 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.

    b. In the Name field, enter the name of the task.

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

    d. Under SQL Tuning Set, in the Name field, enter the name of the SQL tuning set that was built.

    e. Click **Create**.

    The Guided Workflow page appears.

5. Convert the contents in the SQL tuning set into a pre-upgrade SQL trial that will be used as a baseline for comparison.

    a. On the Guided Workflow page, click the **Execute** icon for the Replay SQL Tuning Set in Initial Environment step.

    The Create SQL Trial page appears.

    b. In the SQL Trial Name field, enter the name of the SQL trial.

    c. In the SQL Trial Description field, enter a description of the SQL trial.

    d. In the Creation Method list, select **Build From SQL Tuning Set**.

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

    f. Under Schedule, select **Immediately** to start the task now, or **Later** to schedule the task to start at a time specified using the Date, Time, and Time Zone fields.

    g. 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.

    See Also:

    ■ "Using the Guided Workflow" on page 8-5

    ■ "Creating a Pre-Change SQL Trial Using Enterprise Manager" on page 9-1

Building the Pre-Upgrade SQL Trial for Oracle 10*g* Release 1 and Older Releases Using APIs

After creating a SQL Performance Analyzer task on the SQL Performance Analyzer system, you can use APIs to build the pre-upgrade SQL trial from the execution plans and run-time statistics in the SQL tuning set. To do so, 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 `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');
```

> **See Also:**
>
> - "Creating an Analysis Task Using APIs" on page 8-7
>
> - "Creating a Pre-Change SQL Trial Using APIs" on page 9-3
>
> - *Oracle Database PL/SQL Packages and Types Reference* to learn about the `DBMS_SQLPA.EXECUTE_ANALYSIS_TASK` function

Building the Post-Upgrade SQL Trial Upgrading from Oracle Database 10*g* Release 1 and Older Releases

After the pre-upgrade SQL trial is built, you need to perform a test execute or explain plan of SQL statements in the SQL tuning set on the test system to build a post-upgrade SQL trial. SQL Performance Analyzer remotely test executes the SQL statements using a database link that you need to specify, so that the SQL Performance Analyzer system can connect to the test system and generate the execution plans and statistics for the SQL trial. The database link should exist on the SQL Performance Analyzer system and connect to the test system.

You can build the post-upgrade SQL trial using either Oracle Enterprise Manager or APIs, as described in the following sections:

- Building the Post-Upgrade SQL Trial Upgrading from Oracle Database 10g Release 1 and Older Releases Using Enterprise Manager

- Building the Post-Upgrade SQL Trial Upgrading from Oracle Database 10g Release 1 and Older Releases Using APIs

Building the Post-Upgrade SQL Trial Upgrading from Oracle Database 10*g* Release 1 and Older Releases Using Enterprise Manager

To build the post-upgrade SQL trial using Enterprise Manager:

1. On the Guided Workflow page, click the **Execute** icon for the Replay SQL Tuning Set 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, select **Execute SQLs Remotely**.

5. In the Per-SQL Time Limit list, select **Customize** and specify a reasonable time limit for executing the SQL workload.

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 new database link using the Create Database Link page.

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

8. Under Schedule, select **Immediately** to start the task now, or **Later** to schedule the task to start at a time specified using the Date, Time, and Time Zone fields.

9. 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.

> **See Also:**
>
> - "Using the Guided Workflow" on page 8-5
> - "Creating a Post-Change SQL Trial Using Oracle Enterprise Manager" on page 10-1

Building the Post-Upgrade SQL Trial Upgrading from Oracle Database 10*g* Release 1 and Older Releases Using APIs

To build the post-upgrade SQL trial using APIs, perform an explain plan or test execute using the SQL Performance Analyzer system by calling the EXECUTE_ANALYSIS_TASK procedure. 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.

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.

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 a Post-Change SQL Trial Using APIs" on page 10-3
> - *Oracle Database PL/SQL Packages and Types Reference* to learn about the DBMS_SQLPA.SET_ANALYSIS_TASK_PARAMETER procedure

Upgrading from Oracle Database 10*g* 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 10*g* 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.

The production system which you are upgrading from should be running Oracle Database 10*g* Release 2 or a newer release. Initially, the test system should also be running the same release. To ensure that the analysis made by SQL Performance Analyzer is accurate, the test system should contain an exact copy of the production data found on the production system. Furthermore, the hardware configuration should also be as similar to the production system as possible.

Next, you will need to set up a separate SQL Performance Analyzer system running Oracle Database 11*g* Release 1 or a newer release. The database version should be release 11.1.0.7 or higher. 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 10*g* 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 7-3.

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 7-4.

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 Performance Tuning Guide*

4. On the SQL Performance Analyzer system, create a new 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 "Building the Pre-Upgrade SQL Trial for Oracle Database 10g Release 2 and Newer Releases" on page 12-11.

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 "Building the Post-Upgrade SQL Trial Upgrading from Oracle Database 10g Release 2 and Newer Releases" on page 12-13.

7. Compare SQL performance and fix regressed SQL:

   SQL Performance Analyzer compares the performance of SQL statements captured during the pre-upgrade SQL trial to those captured 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. You can then repeat the process of executing the SQL tuning set and comparing its performance to a previous execution to test any fixes or additional changes made. Repeat these steps until you are satisfied with the outcome of the analysis.

Building the Pre-Upgrade SQL Trial for Oracle Database 10*g* Release 2 and Newer Releases

Once the SQL tuning set is transported to the SQL Performance Analyzer system, you need to perform a test execute or explain plan of SQL statements in the SQL tuning set on the test system to build a pre-upgrade SQL trial. SQL Performance Analyzer remotely test executes the SQL statements using a database link that you need to specify, so that the SQL Performance Analyzer system can connect to the test system to generate the execution plan and statistics for the SQL trial. The database link should exist on the SQL Performance Analyzer system and connect to the test system.

You can build the pre-upgrade SQL trial using Oracle Enterprise Manager or APIs, as described in the following sections:

- Building the Pre-Upgrade SQL Trial for Oracle Database 10g Release 2 and Newer Releases Using Enterprise Manager

- Building the Pre-Upgrade SQL Trial for Oracle Database 10g Release 2 and Newer Releases Using APIs

Building the Pre-Upgrade SQL Trial for Oracle Database 10*g* Release 2 and Newer Releases Using Enterprise Manager

To build the pre-upgrade SQL trial using Enterprise Manager:

1. On the Database Home page, click **Advisor Central**.

   The Advisor Central page appears.

2. Click **SQL Performance Analyzer**.

   The SQL Performance Analyzer page appears.

3. Click **Guided Workflow**.

   The Guided Workflow page appears.

4. Create a SQL Performance Analyzer task using a SQL tuning set as its input source:

   a. 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.

b. In the Name field, enter the name of the task.

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

d. Under SQL Tuning Set, in the Name field, enter the name of the SQL tuning set that contains the SQL workload to be analyzed

e. Click **Create**.

The Guided Workflow page appears.

5. Convert the contents in the SQL tuning set into a pre-upgrade SQL trial that will be used as a baseline for comparison.

a. On the Guided Workflow page, click the **Execute** icon for the Replay SQL Tuning Set in Initial Environment step.

The Create SQL Trial page appears.

b. In the SQL Trial Name field, enter the name of the SQL trial.

c. In the SQL Trial Description field, enter a description of the SQL trial.

d. In the Creation Method list, select **Execute SQLs Remotely**.

e. In the Per-SQL Time Limit list, select **Customize** and specify a reasonable time limit for executing the SQL workload.

f. 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 new database link using the Create Database Link page.

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

h. Under Schedule, select **Immediately** to start the task now, or **Later** to schedule the task to start at a time specified using the Date, Time, and Time Zone fields.

i. 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.

6. Upgrade the test system.

> **See Also:**
>
> - "Using the Guided Workflow" on page 8-5
> - "Creating a Pre-Change SQL Trial Using Enterprise Manager" on page 9-1

Building the Pre-Upgrade SQL Trial for Oracle Database 10*g* Release 2 and Newer Releases Using APIs

After creating a SQL Performance Analyzer task on the SQL Performance Analyzer system, you can use APIs to build the pre-upgrade SQL trial by performing an explain plan or test execute of SQL statements in the SQL tuning set. To do so, 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'));
```

> **See Also:**
>
> - "Creating an Analysis Task Using APIs" on page 8-7
>
> - "Creating a Pre-Change SQL Trial Using APIs" on page 9-3
>
> - *Oracle Database PL/SQL Packages and Types Reference* to learn about the DBMS_SQLPA.EXECUTE_ANALYSIS_TASK function

Building the Post-Upgrade SQL Trial Upgrading from Oracle Database 10*g* Release 2 and Newer Releases

After the pre-upgrade SQL trial is built, you need to upgrade the test system. Once the database has been upgraded, perform a test execute or explain plan of SQL statements in the SQL tuning on the upgraded test system to build a post-upgrade SQL trial. SQL Performance Analyzer remotely test executes the SQL statements using a database link that you need to specify, so that it can connect to the upgraded test system to generate the execution plan and statistics for the SQL trial. The database link should exist on the SQL Performance Analyzer system and connect to the upgraded test system.

You can build the post-upgrade SQL trial using either Oracle Enterprise Manager or APIs, as described in the following sections:

- Building the Post-Upgrade SQL Trial Upgrading from Oracle Database 10g Release 2 and Newer Releases Using Enterprise Manager

- Building the Post-Upgrade SQL Trial Upgrading from Oracle Database 10g Release 2 and Newer Releases Using APIs

Building the Post-Upgrade SQL Trial Upgrading from Oracle Database 10*g* Release 2 and Newer Releases Using Enterprise Manager

To build the post-upgrade SQL trial using Enterprise Manager:

1. On the Guided Workflow page, click the **Execute** icon for the Replay SQL Tuning Set 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, select **Execute SQLs Remotely**.

5. In the Per-SQL Time Limit list, select **Customize** and specify a reasonable time limit for executing the SQL workload.

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 new database link using the Create Database Link page.

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

8. Under Schedule, select **Immediately** to start the task now, or **Later** to schedule the task to start at a time specified using the Date, Time, and Time Zone fields.

9. 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.

   > **See Also:**
   > - "Using the Guided Workflow" on page 8-5
   > - "Creating a Post-Change SQL Trial Using Oracle Enterprise Manager" on page 10-1

Building the Post-Upgrade SQL Trial Upgrading from Oracle Database 10*g* Release 2 and Newer Releases Using APIs

To build the post-upgrade SQL trial using APIs, perform an explain plan or test execute using the SQL Performance Analyzer system by calling the EXECUTE_ANALYSIS_TASK procedure with the DATABASE_LINK task parameter set 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. 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.

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_11g', -
        execution_params => dbms_advisor.arglist('database_link',
                                            'LINK.A.B.C.BIZ.COM'));
```

> **See Also:**
>
> - "Creating a Post-Change SQL Trial Using APIs" on page 10-3
> - *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.

If you are upgrading from Oracle Database 10*g* and newer releases, you can tune the regressed SQL statements by using the SQL Tuning Advisor or SQL plan baselines, as described in Chapter 11, "Comparing SQL Trials".

If you are upgrading from Oracle 9*i* Database, you will need to use the `DBMS_SQLTUNE` package to tune the regressed SQL statements one at a time.

> **Tip:**
>
> - *Oracle Database Performance Tuning Guide* for information about using the `DBMS_SQLTUNE` package

Index

about, 7-5
building
   post-upgrade version, 12-3, 12-8, 12-11, 12-13
   pre-upgrade version, 12-3, 12-7, 12-10, 12-11,
      12-12, 12-13
SQL tuning set
   about, 7-3
   building, 12-5
   constructing, 12-3
   converting, 12-3, 12-6, 12-7, 12-12
     using APIs, 12-7

U

upgrade environment, 12-2, 12-10

W

workload capture
   about, 2-2
   capture directory, 3-3
   capture files, 2-2
   options, 3-2
   prerequisites, 3-1
   restarting the database, 3-2
   restrictions, 3-3
   workload filters, 3-3
workload filters
   about, 3-3
   exclusion filters, 3-3
   inclusion filters, 3-3
workload preprocessing
   about, 2-3
   replay files, 2-3
workload replay
   about, 2-3
   replay client, 2-3