Oracle® Database

Real Application Testing Addendum

11*g* Release 1 (11.1)

E12159-01

February 2008

ORACLE®

Oracle Database Real Application Testing Addendum, 11*g* Release 1 (11.1)

E12159-01

Primary Author: Immanuel Chan

Contributors: Pete Belknap, Karl Dias, Prabhaker Gongloor, Mughees Minhas, Khaled Yagoub

Contents

Preface

Oracle's 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.

Database Replay enables you to replay a full production workload on a test system to assess the overall impact of system changes. SQL Performance Analyzer enables you to assess the impact of system changes on SQL response time on a given SQL workload.

In this release, Oracle Real Application Testing supports the added functionality to read SQL trace files from Oracle Database 9*i* to construct a SQL tuning set that can be used as an input source for SQL Performance Analyzer. Once constructed, you can use SQL Performance Analyzer to execute the SQL tuning set on Oracle Database 10g Release 2 remotely over a database link. This functionality is provided so that you can use this option to test the impact on SQL response time of a database upgrade from Oracle Database 9*i* to Oracle Database 10*g* Release 2.

> **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*.

This preface contains the following topics:

Audience

This document provides information about how to use SQL Performance Analyzer to test database upgrades from Oracle Database 9*i* to Oracle Database 10*g* and subsequent releases. This document is intended for database administrators, application designers, and programmers who are responsible for upgrading and 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 Concepts*
- *Oracle Database Administrator's Guide*
- *Oracle Database 2 Day DBA*
- *Oracle Database Performance Tuning Guide*
- *Oracle Database 2 Day + 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.

Convention	Meaning
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

SQL Performance Analyzer

SQL Performance Analyzer enables you to assess the performance impact of any system change resulting in changes to SQL execution plans and performance characteristics. Examples of common system changes for which you can use SQL Performance Analyzer include:

- Database upgrade
- Configuration changes to the operating system, hardware, or database
- Database initialization parameter changes
- Schema changes, for example, adding new indexes or materialized views
- Gathering optimizer statistics
- Validating SQL tuning actions, for example, creating SQL profiles or implementing partitioning

This document specifically describes how to use SQL Performance Analyzer in a database upgrade from Oracle Database 9*i* to Oracle Database 10*g* Release 2. For complete information about SQL Performance Analyzer, and how to use it in other cases, see *Oracle Database Performance Tuning Guide*.

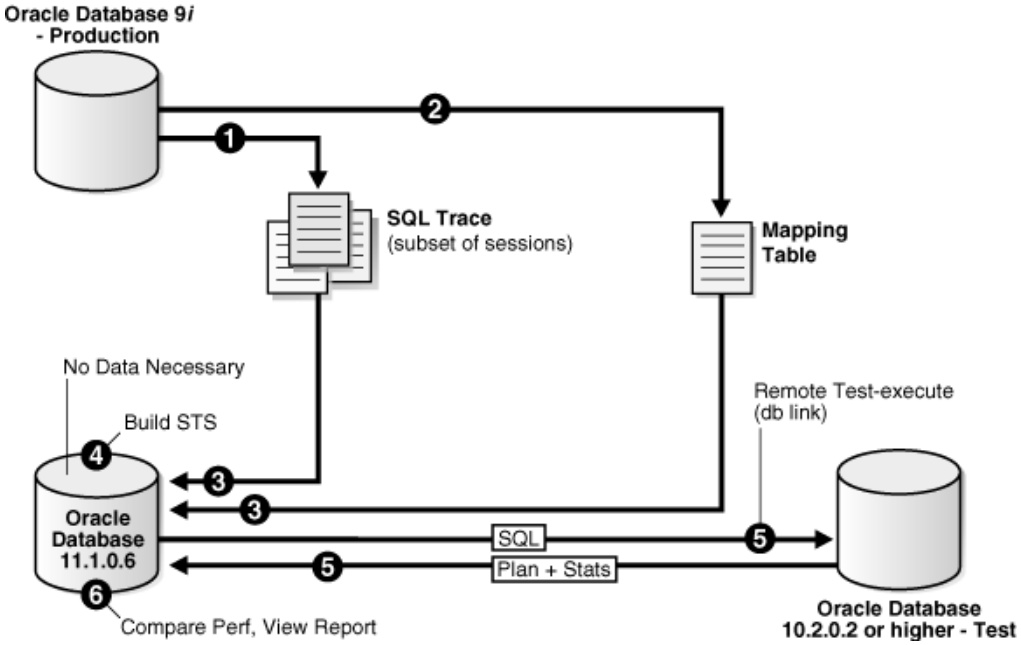
This chapter contains the following sections:

Testing Database Upgrade from Oracle Database 9*i* to Oracle Database 10*g*

SQL Performance Analyzer accepts a representative set of SQL statements stored in a SQL tuning set as its input source. Since SQL tuning sets are not supported in Oracle Database 9*i*, this release supports the added functionality to read SQL trace files from Oracle Database 9i to construct a SQL tuning set that can be used as an input source for SQL Performance Analyzer. Once constructed, you can use SQL Performance Analyzer to execute the SQL tuning set on Oracle Database 10g Release 2 remotely

over a database link. This functionality is provided so that you can use SQL Performance Analyzer to test the impact on SQL response time of a database upgrade from Oracle Database 9*i* to Oracle Database 10*g* Release 2, as illustrated in Figure 1–1.

Figure 1–1 SQL Performance Analyzer Workflow for Database Upgrade from Oracle Database 9i to Oracle Database 10g Release 2



The production system which you are upgrading from should be running Oracle Database 9*i*. The test system which you are upgrading to should be running Oracle Database 10*g* Release 2. The database version can be release 10.2.0.2 or later. If you are upgrading to Oracle Database 10*g* release 10.2.0.2 or 10.2.0.3, you will also need to install a one-off patch before proceeding. To ensure that the analysis made by SQL Performance Analyzer is accurate, this 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 system running Oracle Database 11*g* Release 1. The database version should be release 11.1.0.6. You will also need to install a one-off patch for this release. 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, and SQL Performance Analyzer tasks will be executed remotely on the test system over a database link.

Once the upgrade environment is configured as described, you can use SQL Performance Analyzer in a database upgrade from Oracle Database 9*i* to Oracle Database 10*g* by completing the following steps, as illustrated in Figure 1–1:

1. Enable the SQL Trace facility on the production system running Oracle Database 9*i*.

   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. Create a mapping table on the production system running Oracle Database 9*i*.

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 running Oracle Database 9*i* to the system running Oracle Database 11*g*.
4. On the system running Oracle Database 11*g*, construct a SQL tuning set using the SQL trace files.

   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 system running Oracle Database 11*g*, use SQL Performance Analyzer to build a pre-upgrade SQL trial and a post-upgrade SQL trial:
   a. Convert the contents in the SQL tuning set into a pre-upgrade SQL trial that will be used as a baseline for comparison.
   b. Remotely test execute the SQL statements on the test system running Oracle Database 10*g* over a database link to build a post-upgrade SQL trial.
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.

The remaining sections in this chapter discuss each of these steps in greater detail.

Enabling SQL Trace on the Production System

Oracle Database 9*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.

After enabling SQL Trace on the production system running Oracle 9*i*, 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 system running Oracle Database 11*g*. Once the SQL workload is captured in the SQL trace files, disable SQL Trace on the production system running Oracle 9*i*.

> **See Also:** *Oracle Database Performance Tuning Guide* for additional considerations when using SQL Trace, such as setting initialization parameters to manage SQL trace files

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. Oracle Database 11*g* 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 running Oracle Database 9*i*:

```
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', '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 system running Oracle Database 11*g*.

> **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 system running Oracle Database 11*g*, 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 system running Oracle Database 11g.
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.

   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 1–1 describes the available parameters for the SELECT_SQL_TRACE function.

Table 1–1 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.
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

Running SQL Performance Analyzer

After building the SQL tuning set on the system running Oracle Database 11*g*, you can use it as an input source to run SQL Performance Analyzer. Running SQL Performance Analyzer involves creating SQL trials for Oracle Database 9*i* and Oracle Database 10*g* Release 2, and storing them in a central task container. A SQL trial represents a discrete set of performance data in the task and is generated automatically by the `EXECUTE_ANALYSIS_TASK` procedure as a place to store its results.

To run SQL Performance Analyzer:

1. Create a SQL Performance Analyzer task, as described in "Creating a SQL Performance Analyzer Task" on page 1-6.
2. Execute the task to convert production statistics from the SQL tuning set into a pre-upgrade SQL trial, as described in "Building the Pre-Upgrade SQL Trial" on page 1-6.
3. Perform a test execution to generate statistics and execution plans on the test system running Oracle Database 10*g* Release 2 to build a post-upgrade SQL trial, as described in "Building the Post-Upgrade SQL Trial" on page 1-7.

Creating a SQL Performance Analyzer Task

This section describes how to create a new SQL Performance Analyzer task on the system running Oracle Database 11*g* by using the `DBMS_SQLPA.CREATE_ANALYSIS_TASK` function. A task is a database container for SQL Performance Analyzer execution inputs and results.

Before creating the task, ensure that the SQL workload to use for the performance analysis is available in the form of a SQL tuning set on the system. Call the `CREATE_ANALYSIS_TASK` function using the following parameters:

- Set `task_name` to specify the 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.
- Use `basic_filter` to filter out SQL statements that you do not want to include in the trial.
- Set `order_by` to specify an order-by clause on the selected SQL.
- Set `top_sql` to consider only the top number of SQL statements after filtering and ranking.

The following example creates a SQL Performance Analyzer task named `my_spa_task` that will use the SQL tuning set named `my_sts_9i` as its input source:

```
VARIABLE t_name VARCHAR2(100);
EXEC :t_name := DBMS_SQLPA.CREATE_ANALYSIS_TASK(sqlset_name => 'my_sts_9i', -
       task_name => 'my_spa_task');
```

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

Building the Pre-Upgrade SQL Trial

After the SQL Performance Analyzer task is created on the system running Oracle Database 11*g*, you need to call the `EXECUTE_ANALYSIS_TASK` procedure to take the

execution plans and runtime statistics in the SQL tuning set and use them to build a pre-upgrade SQL trial.

To build the pre-upgrade SQL trial, call the EXECUTE_ANALYSIS_TASK procedure using the following parameters:

- Set the task_name parameter to the name of the SQL Performance Analyzer task that you want to execute.
- Set the execution_type parameter 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: *Oracle Database PL/SQL Packages and Types Reference* to learn about the DBMS_SQLPA.EXECUTE_ANALYSIS_TASK function

Building the Post-Upgrade SQL Trial

After the pre-upgrade SQL trial is built, you need to run a SQL Performance Analyzer task to perform a test execute or explain plan of SQL statements in the SQL tuning set on the test system running Oracle Database 10*g* Release 2 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 Oracle Database 11*g* can connect to Oracle Database 10*g* Release 2 to generate the execution plan and statistics for the SQL trial. The database link should exist on the system running Oracle Database 11*g* and connect to the test system running Oracle Database 10*g* Release 2.

To build the post-upgrade SQL trial, perform an explain plan or test execute using the system running Oracle Database 11*g* by calling the EXECUTE_ANALYSIS_TASK procedure with the DATABASE_LINK task parameter set to the global name of a public database link to be used. 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 performance data using the statistics and execution plans generated from the test system running Oracle Database 10*g*. 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: *Oracle Database PL/SQL Packages and Types Reference* to learn about the DBMS_SQLPA.SET_ANALYSIS_TASK_PARAMETER procedure

Comparing SQL Performance

After the pre-upgrade and post-upgrade SQL trials are built, you can compare the pre-upgrade version of performance data (from the production system) to the post-upgrade version (from the test system) by calling the DBMS_SQLPA.EXECUTE_ANALYSIS_TASK procedure or function to run a comparison analysis. Afterwards, SQL Performance Analyzer can generate a report that shows the results of the comparison and then interpret the results.

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 pre-upgrade SQL trial.
     – Set the execution_name2 parameter to the name of the post-upgrade SQL trial.
     – 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 formula combining 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',
                           'execution_name1', 'my_trial_9i',
                           'execution_name2', 'my_remote_trial_10g',
```

2. Call the DBMS_SQLPA.REPORT_ANALYSIS_TASK function to generate a report 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 `COMPARE PERFORMANCE` 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`, and `ALL`.
- Set the `section` parameter to limit the report to a particular section. 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 include 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', NULL, 100, 'my_exec_compare');
SET LONG 100000 LONGCHUNKSIZE 100000 LINESIZE 130
PRINT :rep
```

3. Review the SQL Performance Analyzer report.

   When reviewing the reports, the following considerations should be made to determine the validity of the results:

   - Hardware and data differences

     Any data or hardware differences between the two systems may produce a greater discrepancy in the results.

   - Use of the SQL Trace facility

     SQL tracing itself has an impact on the statistics generated. Consequently, performance data for the Oracle Database 9*i* trial may appear worse than actual. Therefore, any regression detected after comparing the SQL performance should be addressed before upgrading your production system.

     See Also:

     - For information about the SQL Performance Analyzer report, see *Oracle Database Performance Tuning Guide*
     - *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_SQLPA.EXECUTE_ANALYSIS_TASK` and `DBMS_SQLPA.REPORT_ANALYSIS_TASK` functions

Tuning Regressed SQL

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:

- Run the SQL Tuning Advisor on the regressed SQL statements using the test system running Oracle Database 10*g* Release 2

  For more information about using the SQL Tuning Advisor, see *Oracle Database Performance Tuning Guide*.

- Capture stored outlines on the production system and move them to the test system

  For more information about using stored outlines, see *Oracle Database Performance Tuning Guide*.

After tuning the regressed SQL statements, you should test these changes using SQL Performance Analyzer. Run a new remote test execution on the test system, followed by a second comparison (between this new SQL trial and the pre-upgrade 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 as part of the upgrade process of your production system.

Index