

SQL Performance Analyzer

An Oracle White Paper

November 2007

SQL Performance Analyzer

Introduction	3
Common Usage Scenarios	3
Using SQL Performance Analyzer	4
Capture SQL Workload	4
SQL Tuning Set	5
Transport SQL Workload	6
Execute SQL Pre-Change	6
Make Change	7
Execute SQL Post-Change	7
Compare SQL Performance	7
Comparison Report SQL	7
Re-iterate	8
Use Case Examples	8
Database Upgrade	8
Parameter Change	11
Guided Workflow	14
DBMS_SQLPA Package	15
Real-World Best Practices	18
SQL Workload Capture	18
Test System	18
Performance Comparison	18
Regression Remediation	18
Production Tuning	18
Conclusion	19

INTRODUCTION

Changes that affect SQL execution plans, such as upgrading a database or adding new indexes, can severely impact SQL performance. As a result, DBAs spend enormous amount of time and effort identifying and fixing SQL statements that have regressed due to the changes. SQL Performance Analyzer (SPA), a key feature of the Real Application Testing option introduced in Oracle Database 11g, can predict and prevent SQL execution performance problems caused by system changes.

SQL Performance Analyzer provides a granular view of the impact of changes on SQL execution plans and execution statistics by running the SQL statements in isolation before and after a change. SQL Performance Analyzer compares the SQL execution result, before and after the change, and generates a report outlining the net benefit on the workload due to the changes as well as the set of regressed SQL statements. For regressed SQL statements, appropriate executions plan details along with recommendations to remedy them are provided.

SQL Performance Analyzer is well integrated with existing SQL Tuning Set (STS), SQL Tuning Advisor and SQL Plan Management functionalities. SQL Performance Analyzer completely automates and simplifies the manual and time-consuming process of assessing the impact of change on extremely large SQL workloads (thousands of SQL statements). DBAs can use SQL plan baselines and SQL Tuning Advisor to remediate the regressed SQL statements in test environments and generate optimally performing execution plans. These plans are then exported back into production and used for future executions of SQL statements. Thus, using SQL Performance Analyzer, DBAs can validate with a high degree of confidence that a system change to a production environment in fact results in net positive improvement at a significantly lower cost.

COMMON USAGE SCENARIOS

SQL Performance Analyzer can be used to analyze the impact of any system change that can affect SQL execution plans and run-time statistics.

Examples of common system changes include:

- *Database upgrades including patch deployments:* Database upgrade installs a new version of the optimizer, which has direct effect on SQL performance. SQL

Performance Analyzer enables a DBA to compare SQL performance between the pre-change and post-change versions of Oracle database. In this way, DBA can proactively identify and tune SQL statements that may potentially regress after the database before the change is actually deployed on production system.

- *Database initialization parameter changes:* Changing the value of a database parameter may produce unexpected results. For example, a specific initialization parameter can be enabled to improve performance, but this change may produce unexpected results because the system constraints may have changed.
- *Schema changes:* Changes such as altering indexes or creating new ones, almost inevitably affects SQL performance.
- *Optimizer statistics refresh:* Gathering new statistics for objects whose statistics are stale or missing greatly influences the optimizer's costing algorithms, which can have considerable effects on what execution plans get generated. DBAs will need assurance that these plans do not hurt their system performance.
- *Changes to operating systems and hardware:* Changes, such as installing a new operating system, adding more CPUs or memory, or moving from a single instance database environment to Oracle Real Application Clusters may as well have a significant effect on SQL performance. SQL Performance Analyzer makes it easier to determine the improvement or deterioration to SQL performance when making these changes.
- *Implementation of tuning recommendations:* Accepting recommendations from an advisor (such as ADDM, SQL Tuning Advisor, or SQL Access Advisor) may require users to validate the effect of the tuning recommendations before implementing them. For example, SQL tuning advisor may recommend accepting a SQL profile for a particular SQL statement to improve its performance. But before accepting the recommended profile, the user can invoke SQL Performance Analyzer to measure the performance improvement that may be gained by the implementation of the profile and then determine whether to accept the recommendation.

USING SQL PERFORMANCE ANALYZER

As illustrated in Figure 1, SQL Performance Analyzer evaluates the impact of system changes on SQL performance through five main steps.

Capture SQL Workload

Before running SQL Performance Analyzer, users have to capture a set of SQL statements on the production system that represents the SQL workload that they intend to analyze. The more SQL statements are captured in the workload, the better the workload will represent the application or system and the more accurate

the prediction of performance changes will be. Therefore, users should capture as many SQL statements as possible for the interesting time period (for e.g., month-end close, quarter-end close, day-time OLTP, nightly batch, etc.) in order to get the best results from the performance analysis for the changes they intend to make. It would be ideal to capture all SQL statements that are issued by an application or run on a system.

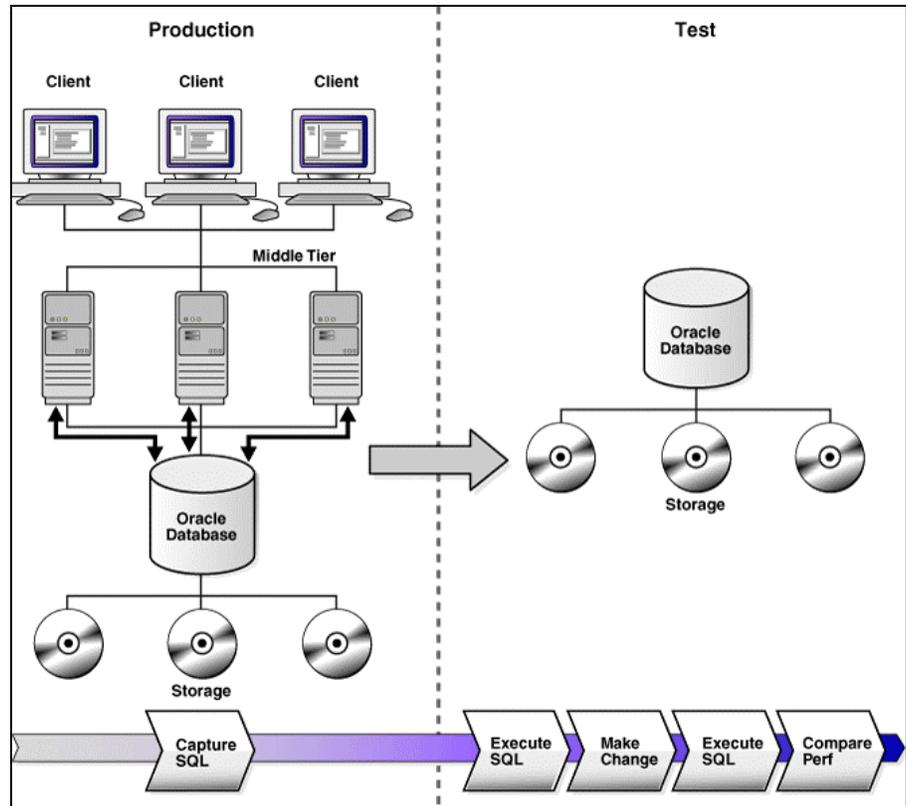


Figure1. SQL Performance Analyzer Workflow

SQL Tuning Set

The representative set of SQL statements to test will be captured and stored into a SQL Tuning Set. SQL Tuning Set is a database object that is used to manage SQL workloads. It can be used to capture and persistently store one or more SQL statements along with their complete execution context, including the text of the SQL, parsing schema under which the SQL statement can be compiled, bind values, execution counts, plans, and statistics. The complete execution context of the SQL statements makes it possible to re-execute them without the need to setup application and middle tier environments. This results in significant hardware and timesavings for customers.

SQL Tuning Sets can be populated from different SQL sources, including the cursor cache, Automatic Workload Repository (AWR), existing SQL Tuning Sets, or custom SQL provided by the user. Among the services SQL Tuning Set infrastructure provides to help users build SQL Tuning Sets from the above sources, incremental SQL workload capture is the most important one because it

enables the capture of the full system SQL workload. Incremental capture works by repeatedly polling the cursor cache over a period of time and updating the workload data stored in the SQL Tuning Set. It can be executed over as long a period as required to capture an entire system workload. Only statements that run during the specified period and meet specified criteria (for e.g., user, service, action, module, etc.) are captured. Incremental cursor capture imposes negligible performance overhead on the system. Extensive tests conducted internally revealed that the overhead of incremental capture is < 1%.

SQL Tuning Sets are also transportable across databases. A SQL Tuning Set captured on a database can be exported and imported into another, allowing for the transfer of SQL workload between databases, for example, for remote performance diagnostics and tuning.

Transport SQL Workload

SQL Performance Analyzer can be run on the production database or a test database. If a test database is used, then the test database environment has to be configured 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.

There are many ways to create a test database. For example, using the DUPLICATE command of Recovery Manager (RMAN), Oracle Data Pump, or transportable tablespaces. For best results, the test database should be as similar to the production system as possible.

After creating the SQL Tuning Set with the appropriate SQL workload, the SQL Tuning Set needs to be exported from the production system and imported into the test system where the system change under consideration will be tested. Oracle Enterprise Manager Grid Control 11g provides a simple interface to copy a specified SQL Tuning Set from one database to another. A SQL Tuning Set can also be copied manually using API. Please refer to the paper [“Optimizing the Optimizer: Essential SQL Tuning Tips and Techniques”](#) on OTN for more details.

Execute SQL Pre-Change

After the SQL workload is captured and the SQL Tuning Set transported to the test system, SQL Performance Analyzer can be used to build the pre-change performance data before making the system change. SQL Performance Analyzer executes the SQL statements captured in the SQL Tuning Set and generates execution plans and execution statistics for each statement. Only queries and the query part of DML statements are executed to avoid side effects on the database. SQL Performance Analyzer executes SQL statements sequentially and in isolation from each other without any respect to their initial order of execution and concurrency. However, the order in which SQL Performance Analyzer executes the SQL queries can be customized. For example, it can be configured to start with the most expensive SQL statements in terms of response time.

Depending on the size of the SQL workload and complexity of SQL statements, executing SQL statements can be resource-intensive and long running. SQL Performance Analyzer provides a mode to generate execution plans only (through the explain plan option) without actually executing the SQL statements. This technique reduces the execution time and resource system resource consumption, but the results of the comparison analysis may not be as accurate.

Make Change

After the pre-change performance data is built, the system change to test can be implemented. As explained earlier, this change can be any kind of change that might impact the performance of SQL statements such as a database upgrade, new index creation, initialization parameter changes, optimizer statistics refresh, and so on.

Execute SQL Post-Change

After implementing the planned change in the test system, SQL Performance Analyzer can be invoked again to re-execute the SQL statements and produce execution plans and execution statistics for each SQL statement, a second time. This execution result represents the post-change performance data that SQL Performance Analyzer uses to compare against the pre-change version of the performance data.

Compare SQL Performance

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

During performance analysis and comparison, SQL Performance Analyzer takes into account the number of executions of a SQL statement when measuring its change impact. This information is collected during the SQL workload capture and stored in the SQL Tuning Set for each SQL statement. SQL Performance analyzer uses the number of executions to determine the importance (weight) of each SQL in the SQL workload. A SQL statement that completes in seconds but is frequently executed may have an equal or higher impact on the system compared to a long running statement executed only once. SQL Performance Analyzer takes these factors into account when predicting overall performance improvements and regressions. By default, SQL Performance Analyzer uses elapsed time as the comparison metric but a user can choose an alternative metric such as disk reads, CPU time, buffer gets, etc. for analysis. If you're using SQL Performance Analyzer API, you have the ability to specify a user-defined function as comparison metric for analysis.

Comparison Report SQL

This SPA comparison report (in Figure 3) shows significant performance improvement of overall SQL workload after the proposed system change but with a few execution plan regressions. SQL Performance Analyzer takes into account the number of executions of a SQL statement when predicting overall performance improvements and regressions. If any regressions are encountered, SQL Performance Analyzer allows the user to remediate them using SQL Tuning Advisor or with SQL Plan Baselines. SQL Plan Baselines is a new feature introduced in Oracle Database 11g to address issues of plan stability. Please refer to the technical white paper "[SQL Plan Management in Oracle Database 11g](#)" on OTN for more information. Using SQL Plan Baselines to remediate regressed SQL results in regressed SQL reverting to previously known execution plans. Using SQL Tuning Advisor to remediate the regressed SQL can result in the optimizer picking alternate execution plans that help improve performance of regressed SQL statements. The "Database Upgrade" use case example in the following section describes how to interpret the key information in the SQL Performance Analyzer comparison report.

Re-iterate

If the performance comparison reveals regressed SQL statements, then you can make further 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 Tuning Set and comparing its performance to the first execution. Repeat these steps until you are satisfied with the outcome of the analysis.

USE CASE EXAMPLES

This section shows how to use Oracle Enterprise Manager to test the effect of common changes on the performance of an application SQL workload. The Oracle Enterprise Manager interface guides users in assessing the impact of changes on the performance of SQL workload through the following workflows:

- Optimizer Upgrade Simulation, which can be used to simulate a database upgrade and measure its effect on a SQL workload.
- Parameter Change, which makes it easier for users to determine how a database initialization parameter change will affect SQL performance.
- Guided Workflow, which can be used to compare the performance of SQL execution for a variety of changes.

The SQL Performance Analyzer functionality can be accessed through the Oracle Enterprise Manager interface under *Software and Support* tab, *Real Application Testing* category. Figure 2 illustrates the three workflows that are available in the Enterprise Manager interface.

Database Instance: v3f.us.oracle.com

Home Performance Availability Server Schema Data Movement Software and Support

Software

Configuration
Collection Status
Clone Oracle Home
Host Configuration
Oracle Home Inventory

Real Application Testing
Database Replay
SQL Performance Analyzer

SQL Performance Analyzer

Page Refreshed Nov 6, 2007 7:10:22 PM PST

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

SQL Performance Analyzer Workflows

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

- [Optimizer Upgrade Simulation](#) Test the effects of optimizer version changes on SQL Tuning Set performance.
- [Parameter Change](#) Test and compare an initialization parameter change on SQL Tuning Set performance.
- [Guided Workflow](#) Create a SQL Performance Analyzer Task and execute custom experiments using manually created replay trials.

SQL Performance Analyzer Tasks

Select Name	Owner/Description	Last Run Status	Created	Last Modified
TEST1111	SYS	■	Oct 30, 2007 3:40:33 PM	Oct 30, 2007 3:40:33 PM
GGG	SYS	✓	Oct 26, 2007 4:25:27 PM	Oct 26, 2007 4:27:52 PM

Figure 2: Enterprise Manager interface for SQL Performance Analyzer

Database Upgrade

SQL Performance Analyzer can automatically simulate the effect of a database upgrade on SQL performance. The database initialization parameter, OPTIMIZER_FEATURES_ENABLE enables simulation of the optimizer

component of the database upgrade. All SQL statements use the optimizer, which is a part of Oracle Database that determines the most efficient means of accessing the specified data. You could use SQL Performance Analyzer to compare the performance of SQL execution when using say 10.2.0.2 and 11.1.0.1 versions of the optimizer.

After you select a SQL Tuning Set and a comparison metric, SQL Performance Analyzer creates two replay trials. The first trial captures SQL performance by simulating the optimizer from the user-selected previous release, whereas the second trial uses the optimizer from the current release. The system-generated Replay Trial Comparison report evaluates SQL regression. If performance was degraded, then you can then use SQL Tuning Advisor to implement any SQL profile recommendations for regressed SQL. Figure 3 illustrates the “Optimizer Upgrade Simulation” use case.

Note that this workflow can also be leveraged for Oracle Database 10g upgrades, such as patchset upgrades (10.2.0.2 to 10.2.0.3) or minor upgrades (10.1.0.3 to 10.2.0.3). In this case, optimizer version1 will be the version of the database release one is currently running and optimizer version2 is the version you want to simulate. In order to leverage the 11g SQL Performance Analyzer functionality, it is assumed that one has upgraded a copy of the test system to 11g.

SQL Performance Analyzer Workflows
 Create and execute SQL Performance Analyzer Task experiments of different types using the following links.
[Optimizer Upgrade Simulation](#) Test the effects of optimizer version changes on SQL Tuning Set performance.
[Parameter Change](#) Test and compare an initialization parameter change on SQL Tuning Set performance.
[Guided Workflow](#)

Optimizer Upgrade Simulation
 Test the effects of optimizer version changes on SQL Tuning Set performance.

Task Information

* Task Name

* SQL Tuning Set

Description

Per-SQL Time Limit

TIP Time limit is on elapsed time of test execution of SQL. EXPLAIN ONLY generates plans without test execution.

Optimizer Versions

Version 1 Version 2

Evaluation

Comparison Metric

Schedule

Time Zone

Immediately

Figure 3: Enterprise Manager interface for “Optimizer Upgrade Simulation”

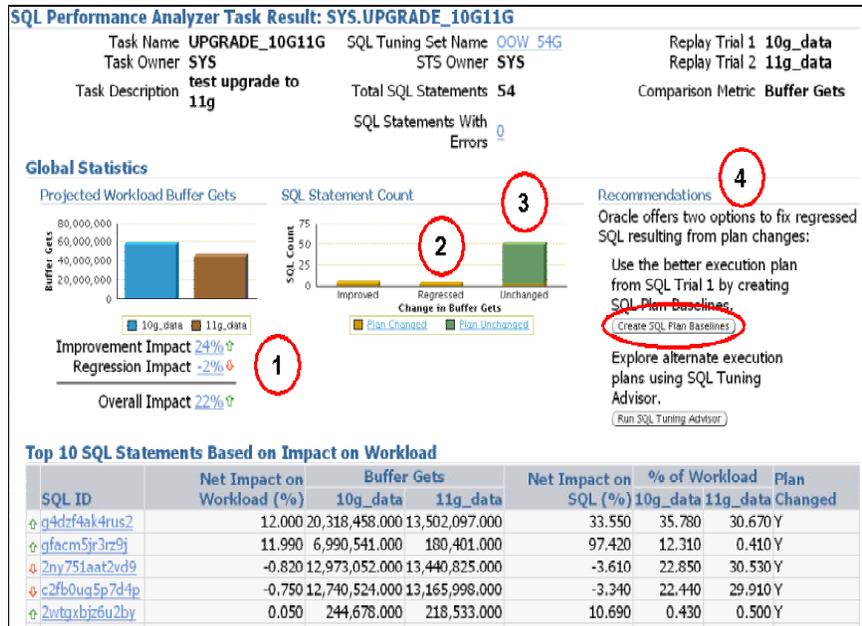


Figure 4: Optimizer Upgrade Simulation workflow: Comparison Report

The SQL Performance Analyzer report that is generated (see Figure 4) consists for three sections. The first section presents the task information, the SQL Tuning Set that was used, the replay trial names and if the SQL statements encountered any errors. The second section consists of global statistics based on the comparison metric used for analysis, the overall improvement on the workload and regression impact. While the workload has improved in performance, there are some regressed SQL statements. The SQL statement count gives a further break down on the regressed SQL. In order to remediate the regressed SQL two options are available, SQL Plan Baselines and SQL Tuning Advisor. SQL Tuning Advisor can be used to improve execution performance for regressed SQL, while SQL Plan Baselines help revert back to known execution plans. Any of the options can be used to remediate the regressed SQL statements. The third section gives the top 10 SQL statements impacting the workload with ability to drill down and get detailed statistics for the SQL.

In this use case, SQL Plan Baselines were used for remediating the regressed SQL and another SQL Performance Analyzer trial was created, followed by a comparison between the trial after the regressed SQL was fixed and the original 10g performance. As show in Figure 5, no regressions exist and one can now safely upgrade to production release.

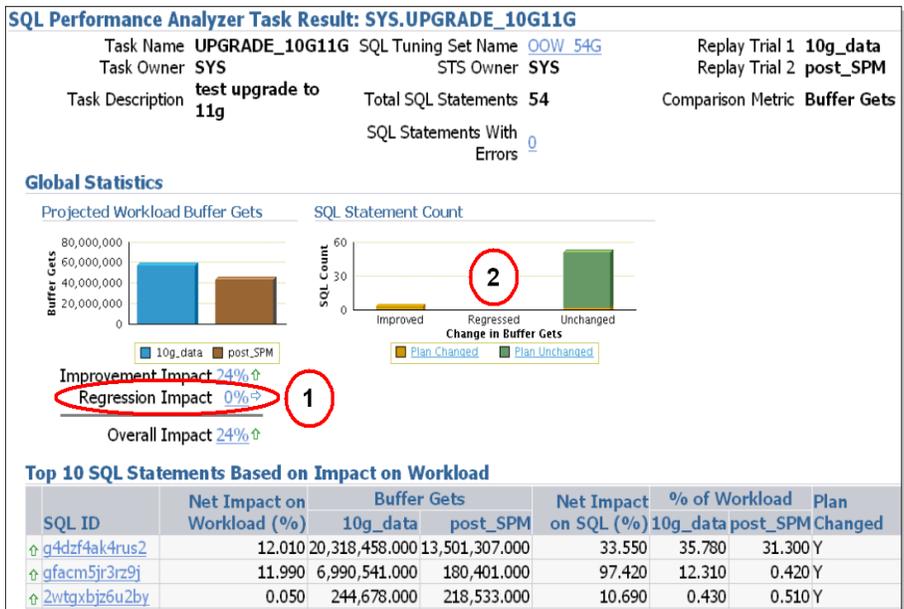


Figure 5: Optimizer Upgrade Simulation: Comparison Report after regressed SQL remediated through SQL Plan Baselines

Parameter Change

The Parameter Change workflow enables you to test the performance effect on a SQL Tuning Set when you vary a single database initialization parameter between two values. For example, you can compare SQL performance when the sort area size is increased from 1 MB to 2 MB. Figure 6 illustrates a use case where a customer wants to migrate his/her legacy application from Rule-Based Optimizer to Cost-Based Optimizer (ALL_ROWS).

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. The Analyzer then performs a second trial with the parameter set to the new value. The system-generated Replay Trial Comparison report evaluates the regression.

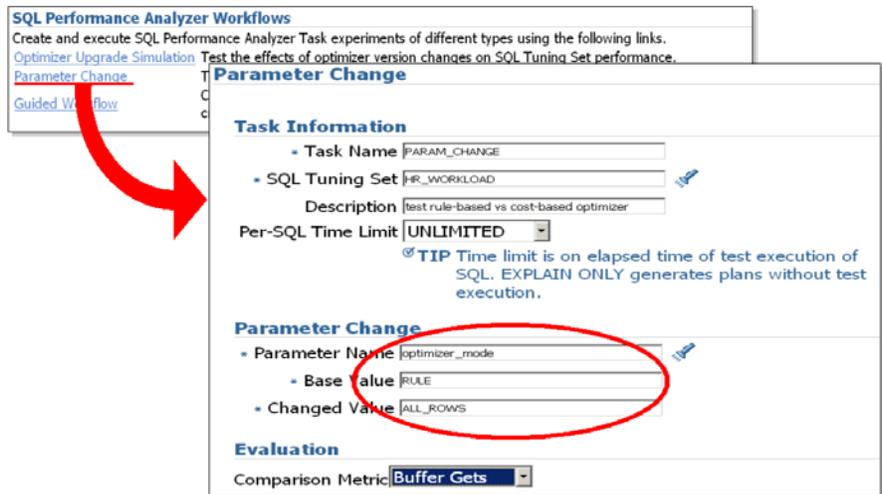


Figure 6: Enterprise Manager interface for “Parameter Change”

Figure 7 illustrates the SQL Performance Analyzer comparison report for the parameter change use case. It shows overall impact to be +24% on the workload; however, a few regressed SQL statements exist. One can use either SQL Plan Baselines or SQL Tuning Advisor for remediating regressed SQL statements, for illustrative purposes, we select the latter. One can click on “Run SQL Tuning Advisor” button to explore any alternate plans that can improve workload performance. In this case (see Figure 8), SQL Tuning Advisor recommends SQL Profiles for four of the regressed SQL statements and one can implement them to tune the regressed SQL Statements. After the SQL Profiles are implemented, another SQL replay trial can be created to evaluate the performance post SQL Profiles. Figure 9 illustrates no performance regression and in fact the overall impact has increased from 24% to 33%, which is a gain of 9% due to SQL Profiles improving the performance of the queries. A DBA can now safely implement the parameter change in production along with SQL Profiles knowing that it will improve the performance of the workload.

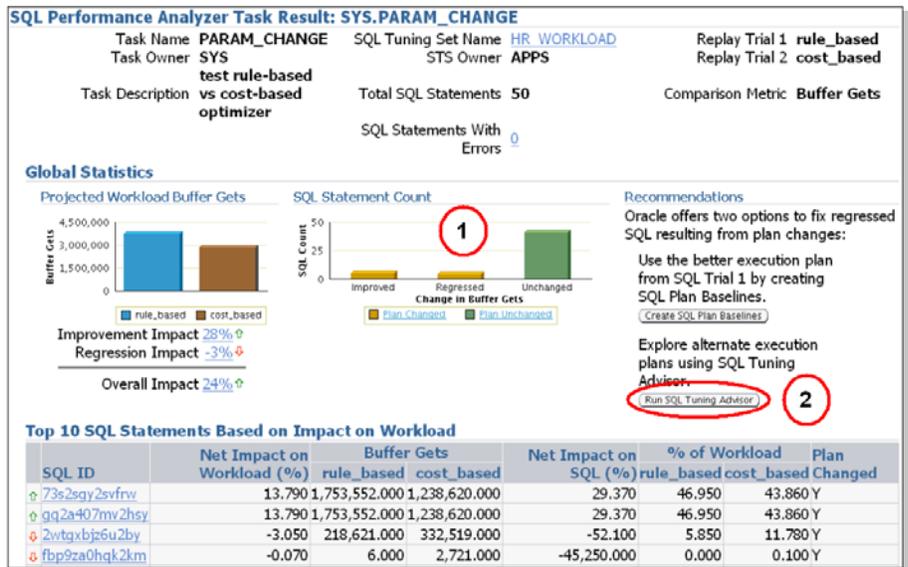


Figure 7: Parameter Change workflow: Comparison Report after parameter change

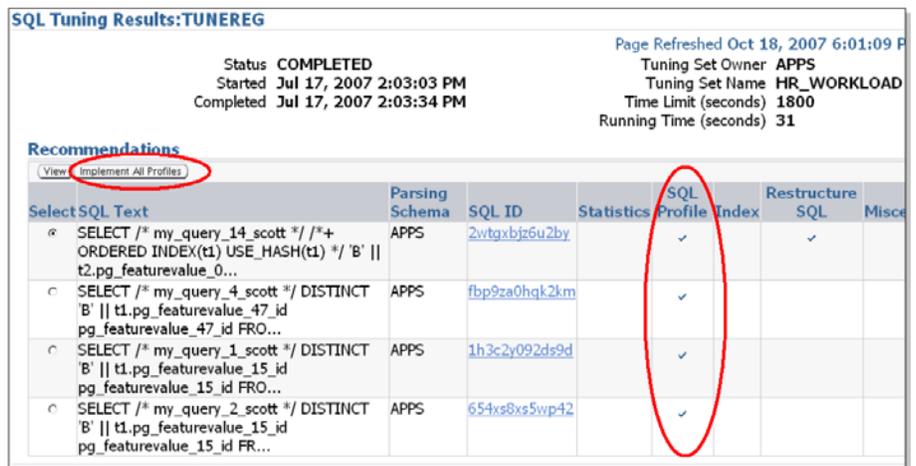


Figure 8: Parameter Change workflow: SQL Tuning Advisor recommendations for regressed SQL

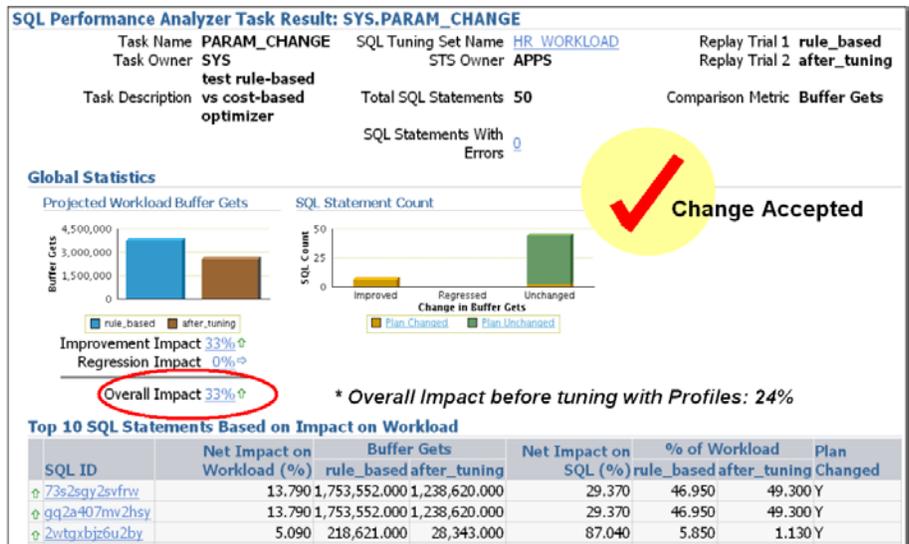


Figure 9: Parameter Change workflow: Comparison Report after remediating regressed SQL through SQL Profiles

Guided Workflow

You can use the “Guided Workflow” link to compare the performance of SQL statements before and after a variety of system changes that can impact the performance of the SQL workload.

In the example used here for guided workflow, the DBA thinks that some key indexes are missing and would like to create them on production database. But before doing so, he/she would like to validate that the indexes created will not be detrimental to other SQL statements.

Figure 10 illustrate the Guided Workflow in the Oracle Enterprise Manager interface.

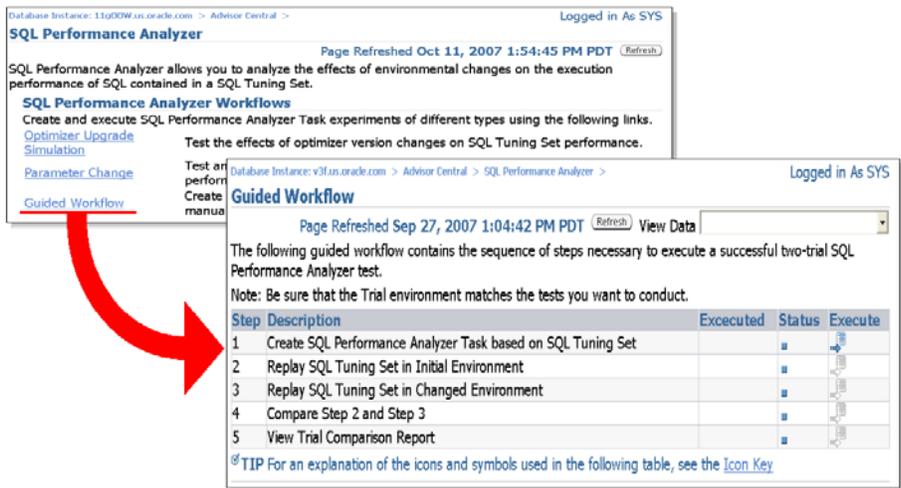


Figure 10: Enterprise Manager interface for "Guided Workflow"

Figure 11 illustrate the indexes that were created do not affect other SQL statements and there is an overall impact of +8% on the workload. So this change can be safely be deployed on the production system.

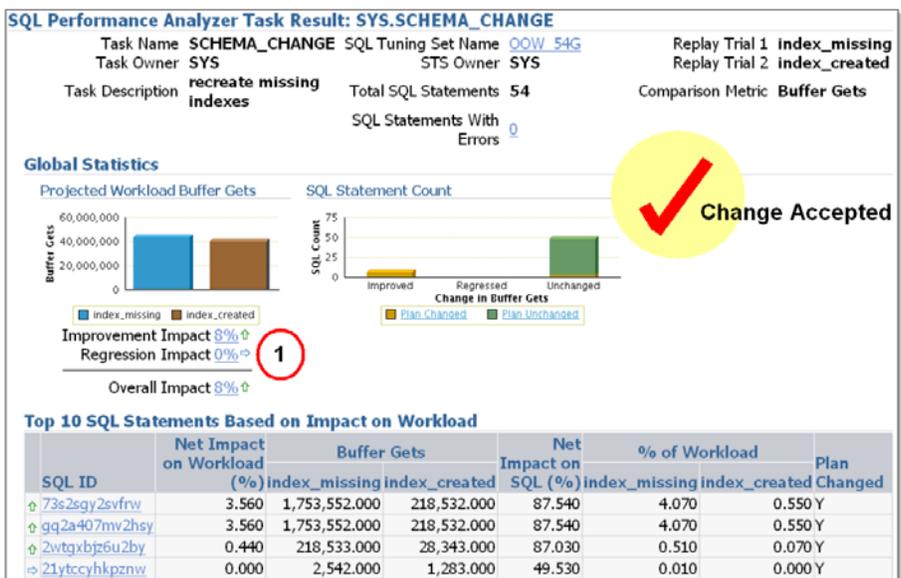


Figure 11: Guided Workflow: Comparison Report after schema change

DBMS_SQLPA PACKAGE

While the primary interface for SQL Performance Analyzer is the Oracle Enterprise Manager, a command line interface to the DBMS_SQLSPA package can also be used to test the impact of system changes on SQL performance. DBMS_SQLPA is

a new package added in Oracle 11g containing necessary APIs for using SQL Performance Analyzer, including functionalities such as, for generating execution plans and statistics for SQL statements and comparing their performance differences.

It is important to note that the SQL Performance Analyzer, like all other manageability advisors, is built on top of a common Advisory Framework. The Advisory Framework provides a common infrastructure support to build, store, and retrieve advice generated by various manageability features. Therefore, all SQL Performance Analyzer procedures operate with advisor task objects called analysis tasks. A SQL Performance Analyzer analysis task is used as a container for its execution inputs and results.

To interact with SQL Performance Analyzer using the DBMS_SQLSPA package, the first step always is to create an analysis task by calling the *create_analysis_task* procedure. This procedure creates an advisor task and sets its corresponding parameters according to the user provided input arguments.

The following example illustrates a call of this procedure:

```
dbms.sqlpa.create_analysis_task(sqlset_name => 'my_sts',
                              task_name   => 'my_spa_task',
                              description => 'test upgrade from 10g to 11g');
```

In the example, we assume that, before creating the task, the SQL workload to use for performance analysis is available on the system in the form of a SQL Tuning Set called 'my_sts'. The name of the created analysis task is 'my_spa_task'.

Once the analysis task is successfully created, it is at an initial state. The task then needs to be executed twice. The first time is to build the SQL performance data before making the intended change to test and the second time to produce the corresponding SQL performance data after the change. This is achieved by invoking the *execute_analysis_task* procedure as follows:

```
dbms_sqlpa.execute_analysis_task(task_name => 'my_spa_task',
                                execution_type => 'test execute',
                                execution_name => 'before_change');
```

The above example illustrates the procedure call before the system change. The procedure is invoked with the *execution_type* argument set to 'test execute' which requests SQL Performance Analyzer to execute all SQL statements in the SQL tuning set in order to generate their execution plans as well as their execution statistics. Setting the *execution_type* argument to 'explain plan' makes the Analyzer produce execution plans only. The resulting plans and statistics will be stored within the analysis task in a container called "before_change", which is the name given to that particular task execution. In Enterprise Manager, we refer to this task execution container as SQL trial.

After making the change, the same procedure is called again using the same arguments, but with a different name for the task execution, e.g., 'after_change', as follows:

```
dbms_sqlpa.execute_analysis_task(task_name => 'my_spa_task',
                                execution_type => 'test execute',
                                execution_name => 'after_change');
```

At any time after a task execution has begun, the user can use an appropriate DBMS_SQLPA procedure to cancel, interrupt or reset the task. The user can also check the status of the task execution by querying V\$ADVISOR_PROGRESS view. This view displays information about the task execution progression made so far. This information includes remaining task execution time, number of findings, and number of statements processed in the SQL Tuning Set.

After the 'after_change' SQL performance data is built, the user can compare it to the 'before-change' version of performance data by calling procedure *execution_analysis_task* a third time as follows:

```
dbms_sqlpa.execute_analysis_task(
    task_name => 'my_spa_task',
    execution_type => 'compare performance',
    execution_name => 'analysis_results',
    execution_params => dbms_advisor.arglist('execution_name1', 'before_change',
                                             'execution_name2', 'after_change',
                                             'comparison_metric', 'buffer_gets'));
```

In the above procedure call, "*execution type*" is set to 'compare performance' to input SQL Performance Analyzer to conduct a comparison of performance. Also, the procedure call uses argument *execution_params* to set some task parameters that are specific for the comparison. The *execution_params* parameters are specified as (name, value) pairs for the specified task execution. In this example, parameter *execution_name1* is set to the name of the 'before_change' execution, which constitutes the baseline of the SQL workload performance data, parameter *execution_name2* refers to the performance data stored within execution 'after_change', and parameter 'comparison_metric' to 'buffer_gets' which determines the metric SQL Performance Analyzer has to use to compare SQL performance. Possible values for this last parameter are: elapsed_time (default), cpu_time, buffer_gets, disk_reads, direct_writes, and optimizer_cost.

And finally, when the analysis task execution is completed, the comparison results can be generated by calling the *report_analysis_task* procedure as shown below.

```
var rep clob;
set long 10000
:rep := dbms_sqlpa.report_analysis_task(task_name => 'my_spa_task');
print :rep
```

By default, *report_analysis_task* produces a textual report (of type CLOB) of the last task execution, which in this example is the one used for performance comparison. The report contains a result summary and details of the top 100 SQL statements whose performance change impacts the workload the most, regardless of whether the SQL statements improved or regressed. The report also shows analysis findings and tuning recommendations for regressed SQL statements.

REAL-WORLD BEST PRACTICES

SQL Workload Capture

Use the “Incremental STS capture” method to accurately capture representative SQL workload. The overhead incurred by incremental STS capture is negligible (<1%) and therefore can be turned on production systems without concern.

Test System

SQL Performance Analyzer analysis can be resource intensive and hence it is recommended to run it on a test system. If run on production system, make sure to run in the maintenance window time period and the system has spare system resources to handle SQL Performance Analyzer task load. It is also important to ensure that if a test system is used, it has similar configuration and comparable optimizer statistics as production.

Performance Comparison

It is recommended to use several different metrics, e.g., elapsed time, CPU time, etc., to compare pre-and post-change performance for reliable results.

Regression Remediation

Both SQL Tuning Advisor and SQL Plan Baselines can be used in remediating SQL regressions. SQL Tuning Advisor can further improve performance by exploring alternate execution plans while SQL Plan Baselines functionality conservatively reverts to previously known plans.

Production Tuning

It is recommended to use session or private testing to limit visibility in the validation phase of implementing tuning recommendations. SQL Profiles can be tested in a private session by using a named “*sqltune_category*” session attribute. See below how this can be accomplished through the alter session command. After the validation phase shows that the SQL Profile in fact benefits the regressed SQL, one can change the ‘*sqltune_category*’ to ‘DEFAULT’ that means all users can use the implemented SQL Profile.

```
alter session set sqltune_category='TEST';
```

CONCLUSION

In this paper, we have described SQL Performance Analyzer functionality that has been introduced in Oracle Database 11g. SQL Performance Analyzer helps users predict the impact of system changes on the performance of SQL statements and fix any potential regressions. SQL Performance Analyzer is well integrated with SQL Tuning Set, SQL Plan Management and SQL Tuning Advisor functionality to provide users a comprehensive solution for managing SQL workload performance. It uses SQL Tuning Set as input for SQL workloads; helps build and compare different versions of SQL performance data, and then generate tuning recommendations to overcome potential performance problems. To fix regressed SQL statements, users can choose to create SQL Plan Baselines or implement SQL Profiles recommended by the SQL Tuning Advisor.

The table below summarizes how SQL Performance Analyzer greatly simplifies the testing process of changes that have impact on SQL performance.

Oracle pre-11g	Oracle 11g
Manual capture of application SQL workload	Automated capture of application SQL workload
Partial SQL workload	Complete/entire SQL workload
Manual execution of SQL to produce execution plans and execution statistics	Automatic generation of SQL execution plans and execution statistics
Months of manual analysis	Automated analysis in minutes
Manual SQL regression remediation	Automatic SQL regression remediation
High risk, high cost	Low risk, low cost

As the table above shows, the effort and time spent by a DBA in this fairly common task is considerably greater in an Oracle pre-11g release compared to Oracle11g using SQL Performance Analyzer. SQL Performance Analyzer offers a comprehensive yet easy to use solution for application change testing, which can be used equally effectively by novice as well as expert users.



SQL Performance Analyzer

November 2007

Author: Khaled Yagoub and Prabhaker Gongloor (GP)

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:

Phone: +1.650.506.7000

Fax: +1.650.506.7200

oracle.com

Copyright © 2007, Oracle. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice.

This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission. Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.