The Self-Managing Database: Guided Application & SQL Tuning with Oracle Database 10*g* Release 2

An Oracle White Paper
June 2006

ORACLE

The Self-Managing Database: Guided Application & SQL Tuning with Oracle Database 10*g* Release 2

The Self-Managing Database: Guided Application & SQL Tuning with Oracle Database 10*g* Release 2[1]

INTRODUCTION

Over the past decade two clear trends have occurred: a) the database systems have been deployed in new areas, such as electronic commerce, with new set of database requirements, and b) the database applications have become increasingly complex with support for very large numbers of concurrent users. As a result, the performance of database systems has become highly visible and thus critical to the success of the businesses running these applications.

One important part of database system performance tuning is the tuning of SQL statements. SQL tuning involves three basic steps:

1. Identify high load or top SQL statements that are responsible for a large share of the application workload and system resources, by looking at the past SQL execution history available in the system (e.g., the cursor cache statistics stored in the V$SQL dynamic view),
2. Verify that the execution plans produced by the query optimizer for these statements perform reasonably well,
3. Take possible corrective actions to generate better execution plans for poorly performing SQL statements.

The three steps are repeated until the system performance reaches a satisfactory level or no more statements can be tuned. An expert, such as a DBA or an application developer, who has a deep knowledge of the application and database system usually performs this tuning process.

The corrective actions can be one or more of the following:

- Gather or refresh the statistics that are used by the query optimizer to build an execution plan. For example, create a histogram on a column that contains skewed data,
- Change the value of some configuration parameter that will affect how the optimizer builds an execution plan. For example, set the value of optimizer_mode to first_rows_10,

[1] Some of the features referenced below are part of separately licensed Diagnostic and Tuning packs. Please refer to the appendix at the end of the document for more details.

- Rewrite the SQL statement to use appropriate SQL constructs. For example, replace UNION by UNION-ALL operator when possible,
- Create or drop a data access structure on a table. For example, create an index or a materialized view,
- Add optimizer hints to the statement. For example, by using the INDEX hint on a table in order to replace a *full table scan* by an *index range scan*.

The manual SQL tuning process poses several challenges to the application developer. First, it requires a high level of expertise in several complex areas: Query optimization, Access design, and SQL design. Second, it is a time consuming process because each statement is unique and needs to be dealt with individually, and moreover, the number of statements can be very large, e.g., more than a thousand. Third, it requires an intimate knowledge of the schema structures (i.e., view definitions, indexes, table sizes, etc.) and the data usage model of the application. Finally, the SQL tuning activity is a continuous task because the SQL workload is always changing, e.g., when new application modules are deployed. Also, changes in data access structures (e.g., when an index or a materialized view is dropped or created) is very likely to cause changes in the execution plans, forcing the application developer to start over again. Figure 1 illustrates the manual SQL Tuning process.

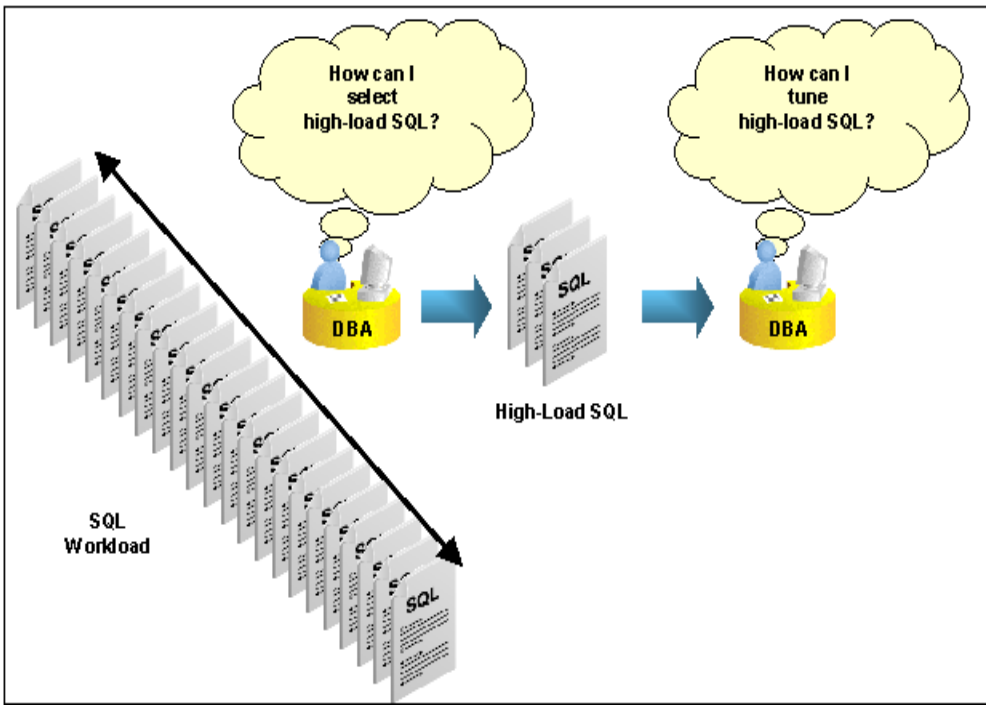


Figure 1: Manual SQL Tuning

To help the database administrator (DBA) and the application developer face those challenges, several software companies have developed performance monitoring and diagnostics tools that try to identify database system performance bottlenecks and suggest actions to fix them. Some of these tools do a very good job in most cases. However, these tools are not integrated with the system component they are targeting. For example, the query optimizer is the system component that is responsible for the SQL execution plans. Indeed, the query optimizer is simply a black box for these tools, and therefore, they have to interpret the optimization information outside of the database to perform the tuning. As a result, their tuning results are less robust and limited in scope. Moreover, the lack of integration makes the outside tools never keep up with the latest improvements and changes in the query optimizer.

In Oracle Database 10g, a great development effort and focus has been put into making the database system self-managing [see the paper "Oracle Database 10g: The Self-Managing Database"]. To enable automatic as well as on-demand system performance monitoring and tuning, a manageability feature called Automatic Workload Repository (AWR) has been introduced. The AWR looks at the system performance data at every 60 minutes (by default), and stores it persistently (for up to 7 days by default) as historical system workload. For example, among other things the AWR identifies top SQL statements that are resource intensive in terms of CPU consumption, buffer gets, disk reads, parse calls, shared memory, etc., in each time interval. Another manageability feature called Automatic Database Diagnostics Monitor (ADDM) is introduced that automates the task of continuous monitoring of system activity, identification of high system resource consumption, and performance bottlenecks [see the paper "The Self-Managing Database: Automatic Performance Diagnosis"]. In the area of SQL tuning, the ADDM identifies the set of high load SQL statements. Yet another manageability feature has been added, which automates the collection of data statistics in a maintenance window. The Automatic Statistics Collection is enabled by default for newly created Oracle Database 10g databases.

In Oracle Database 10g, the SQL tuning process has been automated by introducing a new manageability feature called the Automatic SQL Tuning. It is designed to work equally well for OLTP and Data Warehouse type applications. Automatic SQL Tuning is based on newly added automatic tuning capability of the query optimizer called the Automatic Tuning Optimizer. The Automatic SQL Tuning is exposed via an advisor called the SQL Tuning Advisor. The SQL Tuning Advisor takes one or more SQL statements, and produces well-tuned plans along with tuning advice. The SQL statements may have been identified by the ADDM, or by the AWR, or by a manual process. The manual process may include, for example, the testing of a set of SQL statements that are yet to be deployed, measuring individual performance and identifying the ones whose performance falls short of expectation. To cater to this need, in Oracle Database 10g, we have introduced a new persistent tuning object called the SQL

Tuning Set (STS), which is described in detail in section "SQL Tuning Sets". Figure 2 gives a high level view of how the Automatic SQL Tuning works in Oracle Database 10g.

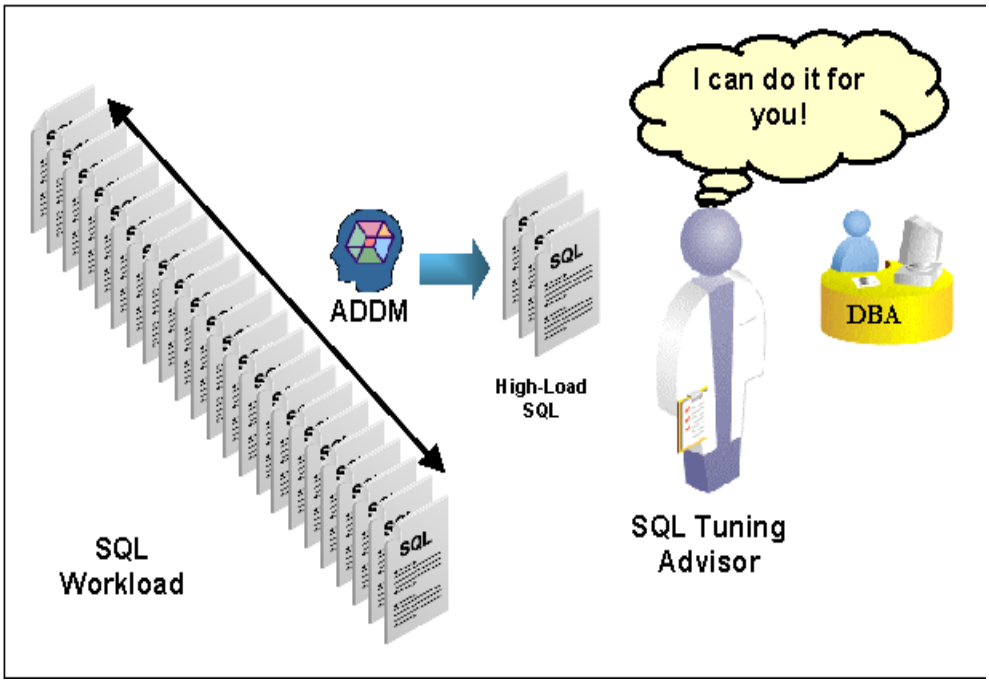


Figure 2: Automatic SQL Tuning

The rest of the paper is organized as follows. First, we explain how the Automatic SQL Tuning works, then give the details about the Automatic Tuning Optimizer. Next, we describe the SQL Tuning Sets that enable the user to create and tune custom SQL workloads. Then we introduce the primary interface to the Automatic SQL Tuning using the Enterprise Manager and illustrate it with examples. We follow this by describing the DBMS_SQLTUNE package, which groups SQL tuning procedures used for the SQL Tuning Advisor API, and the management of SQL Tuning Sets and SQL Profiles. Finally, we conclude this paper with a performance tuning comparison between Orcle9i and Oracle Database 10g.

AUTOMATIC SQL TUNING

The Automatic SQL Tuning is tightly integrated with the query optimizer. This gives several advantages: a) the tuning is done by the system component that is ultimately responsible for the execution plans and hence the SQL performance, b) the tuning process is fully cost-based and it naturally accounts for any changes and enhancements done to the query optimizer, c) the tuning process takes into account the past execution statistics of a SQL statement and

customizes the optimizer settings for that statement, and d) it collects auxiliary information in conjunction with the regular statistics based on what is considered useful by the query optimizer.

In fact, the Automatic SQL Tuning is based on an enhanced version of the query optimizer. The regular query optimizer has been enhanced in Oracle Database 10g to perform additional analyses during the process of building an execution plan for a SQL statement. We refer to this mode of the query optimizer as the Automatic Tuning Optimizer, to distinguish it from its regular operating mode. The query optimizer (under normal mode) has stringent constraints on the amount of time and system resources it can use to find a good execution plan for a given SQL statement. An acceptable optimization time is usually less than a second, and no more than few seconds. Because of this stringent requirement the optimizer performs limited plan search by using heuristics whenever and wherever it can to cut down on the optimization time. Hence, a query optimizer cannot perform time-consuming investigation and verification steps during the plan generation process.

In contrast, the Automatic Tuning Optimizer is typically given much more time, usually in minutes to perform necessary investigation and verification steps as part of the tuning process. Thus the Automatic Tuning Optimizer has a much higher probability of generating a well-tuned plan. The Automatic Tuning Optimizer uses dynamic sampling and partial execution (i.e. execution of fragments of a SQL statement) techniques to verify its own estimates of cost, selectivity, and cardinality. It also uses past execution history of the SQL statement to determine optimal settings (e.g., optimizer mode).

The functionality of the Automatic Tuning Optimizer is exposed via an advisor called the SQL Tuning Advisor. The SQL Tuning Advisor accepts a SQL statement and passes it over to the Automatic Tuning Optimizer along with other input parameters. The Automatic Tuning Optimizer then performs four types of analyses during the plan generation process.

- **Statistics Analysis**: The Automatic Tuning Optimizer checks each query object for missing or stale statistics, and makes recommendation to gather relevant statistics. It also collects auxiliary information to supply missing statistics or correct stale statistics in case recommendations are not implemented.
- **SQL Profiling**: The Automatic Tuning Optimizer verifies its own estimates and collects auxiliary information to remove estimation errors. It also collects auxiliary information in the form of customized optimizer settings (e.g., first rows vs. all rows) based on past execution history of the SQL statement. It builds a SQL Profile using the auxiliary information and makes a recommendation to create it. When a SQL Profile is created it enables the query optimizer (under normal mode) to generate a well-tuned plan.

- **Access Path Analysis**: The Automatic Tuning Optimizer explores whether a new index can be used to significantly improve access to each table in the query, and when appropriate makes recommendations to create such indexes.
- **SQL Structure Analysis**: Here the Automatic Tuning Optimizer tries to identify SQL statements that lend themselves to bad plans, and makes relevant suggestions to restructure them. The suggested restructurings can be syntactic as well as semantic changes to the SQL code.

Each of the analyses is explained in more detail in the following section "Automatic Tuning Optimizer".

The outputs generated by the Automatic Tuning Optimizer are relayed to the user via the SQL Tuning Advisor in the form of an advice. The advice consists of one or more recommendations, each with a rationale and an estimated benefit obtained when implemented. The user is given an option to accept the advice, thus completing the tuning of corresponding SQL statement.

The Automatic Tuning Optimizer together with the SQL Tuning Advisor constitutes the Automatic SQL Tuning component of the Oracle server. The Automatic SQL Tuning architecture as shown in figure 3 illustrates the functional relationship between the Automatic Tuning Optimizer and the SQL Tuning Advisor.

The Automatic SQL Tuning is part of Oracle's ongoing strategy of moving to an advice-driven database administration model. The model envisions that for critical database management functions the database server should give intelligent advise to users on how best to perform them. The database engine itself is now being made more intelligent so that Oracle users no longer need to rely on third-party tools and solutions to manage their databases in the most optimal way.

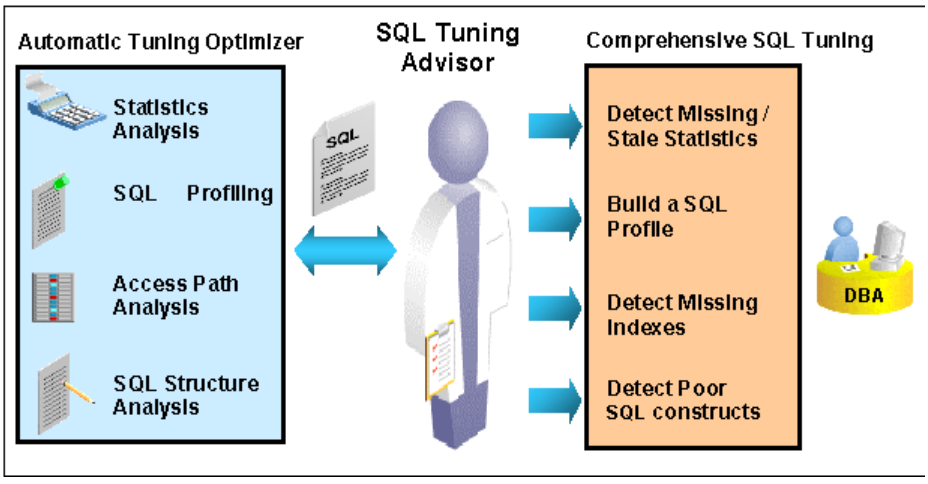


Figure 3: Automatic SQL Tuning Architecture

AUTOMATIC TUNING OPTIMIZER

It is important to note that the Automatic Tuning Optimizer is the query optimizer that runs under a special automatic tuning mode. Therefore, the Automatic Tuning Optimizer performs same functions as the regular query optimizer but with additional capability or functionality. The additional functionality includes the verification steps to validate its own internal estimates as well as the external information supplied (e.g., data statistics). The additional functionality also includes exploratory steps to investigate new access paths that can give significant performance speed-up, as well as explore the possibility of suggesting changes in the SQL constructs to enable efficient processing of the data.

The SQL Tuning advisor invokes the optimizer in automatic tuning mode to get advice on a SQL statement. The advice can consists of various recommendations from the Automatic Tuning Optimizer relating to the statistics, query plan, access paths (e.g. indexes), and SQL constructs. In addition to generating recommendations, it may also gather SQL statement specific information for its own use. This is the auxiliary information that will be used in conjunction with the regular database statistics. The Automatic Tuning Optimizer performs multiple analyses such as statistics analysis, SQL profiling, access path analysis, and SQL structure analysis. Each of these analyses may produce a separate recommendation for the SQL statement.

Statistics Analysis

The query optimizer relies on data statistics to function properly. It is important that the statistics needed to optimize a given SQL statement are collected and kept up to date. The goal of statistics analysis is to verify that these statistics are not missing or stale. The Automatic Tuning Optimizer logs the types of statistics that are actually used during the plan generation process, which are subsequently verified. For example, for equality predicate it will log the distinct column values statistic, whereas for a range predicate it will log the minimum and maximum column value statistics.

Once the logging of actually used statistics is completed, the Automatic Tuning Optimizer goes on to check if each of these statistics is available on associated query object (i.e. table or index or materialized view). If a statistic is available then it verifies its accuracy (i.e., freshness or staleness). To verify the accuracy of a statistic it will sample data from corresponding query object and use sampling result to measure the accuracy.

If a statistic is found to be missing it will generate auxiliary information to supply the missing statistic. If a statistic is available but it is found to be stale it will generate auxiliary information in the form of adjustment factors to the stale statistic so that the resulting statistic will reflect current state of the data.

Note that the statistics analysis produces two kinds of output: 1) recommendations to analyze the objects that are found to have either no statistics or stale statistics, and 2) auxiliary information to supply missing statistics or adjust stale statistics. It is a good idea to implement Analyze recommendations and re-run Automatic SQL Advisor. The auxiliary information is used in case Analyze recommendations are not implemented. The auxiliary information is stored into a SQL Profile, which is described in the following section.

SQL Profiling

The main verification step during the SQL profiling is the verification of the query optimizer's own estimates about cost, selectivity, and cardinality. There are many factors that can cause large errors in the estimates and lead the optimizer to generate a sub-optimal plan. Some of the important factors are: a) the use of internal default predicate selectivity when statistics are missing (e.g., table is not analyzed), b) when the predicate is complex so the query optimizer has no idea how much data will be filtered by this predicate, c) presence of data correlation between columns of a table which means the optimizer's general assumption of no correlation is wrong, d) skewed or sparse join relationship between tables which is very difficult to capture in the form of statistics, (e) existence of data correlation between columns of two or more tables (for example, join relationship between product, location, and sales tables wherein the #sales of snow shoes is very large in New York but it is very small (possibly zero) in Arizona). For the query optimizer to generate a good plan it is important that the estimates it makes do not contain large errors. Hence, it is very important for the Automatic Tuning Optimizer to verify the accuracy of its own estimates and remove or at least significantly reduce the error in them.

Another important source of sub-optimal plan generation is due to the incorrect settings for the optimizer. For example, if the user is going to fetch only few rows even when the size of the result is very large, it is important to optimize for minimizing the time to produce first few rows rather than optimize for all rows. Therefore, for such a SQL statement it is important to set the optimizer mode for first_rows rather than all_rows.

During SQL Profiling, the Automatic Tuning Optimizer performs verification steps to validate it's own estimates. The validation consists of taking a sample of data and applying appropriate predicates to the sample. A new estimate is produced from the data sample result. This new estimate will be accurate because the sample size is adjusted if necessary to guarantee a high level of accuracy. The new estimate is compared to the regular estimate, and if the difference is large enough a correction factor is produced to bring the regular estimate in line with the new estimate. Another method of estimate validation involves the execution of a fragment of the SQL statement (i.e., a partial execution). The partial execution method is more efficient than sampling

method when respective predicates provide efficient access paths. The Automatic Tuning Optimizer picks the appropriate estimate validation method.

The Automatic Tuning Optimizer also uses the past execution history of the SQL statement to determine correct settings. For example, if the execution history tells that a SQL statement is only partially executed majority of times then appropriate setting will for first_rows. This will be a customized setting for the SQL statement being tuned.

The Automatic Tuning Optimizer builds a SQL Profile if it has generated auxiliary information either during statistics analysis (i.e., statistics adjustment factors), or during SQL Profiling (i.e. estimate correction factors, customized optimizer settings). When a SQL Profile is built, it generates a user recommendation to create a SQL profile.

Because the estimate validation using the method of data sampling or partial execution can be expensive and time-consuming process, the SQL Profiling is not performed when the SQL Tuning Advisor is run under Limited mode. You need to use Comprehensive mode to let the Automatic Tuning Optimizer generate a SQL Profile.

SQL Profile

A SQL Profile is a collection of auxiliary information that is built during automatic tuning of a SQL statement. Thus, a SQL Profile is to a SQL statement what statistics is to a table or index object. Once created a SQL Profile is used in conjunction with the existing statistics by the query optimizer (in normal mode) to produce a well-tuned plan for the corresponding SQL statement.

A SQL Profile is created when it is stored persistently in the data dictionary. Once a SQL Profile is created, thereafter, every time the corresponding SQL statement is compiled (i.e., optimized) the query optimizer will use the SQL Profile to produce a well-tuned plan. A full set of functions is provided for the management of SQL Profiles (see section “DBMS_SQLTUNE Package”).

Figure 4 below shows the process flow of the creation and use of a SQL Profile. The process consists of two separate phases: SQL tuning phase, and regular optimization phase. During SQL tuning phase, a DBA selects a SQL statement for automatic tuning and runs the SQL Tuning Advisor either using the Enterprise Manager GUI interface (see section “SQL Tuning Interface”), or using the command line interface (see section “DBMS_SQLTUNE Package”). The SQL Tuning Advisor invokes the Automatic Tuning Optimizer to generate tuning recommendations possibly with a SQL Profile. If a SQL Profile is built the DBA can accept it. When it is accepted the SQL Profile is stored in the data dictionary. Then in the next phase, when an end-user issues same the SQL statement the query optimizer (under normal mode) utilizes the SQL Profile to build a well-tuned plan. The use of SQL Profile remains completely transparent to the end-user.

It is very important to note that the creation and use of a SQL Profile doesn't require changes to the application source code. This is key to tuning SQL statements that are issued by packaged applications.

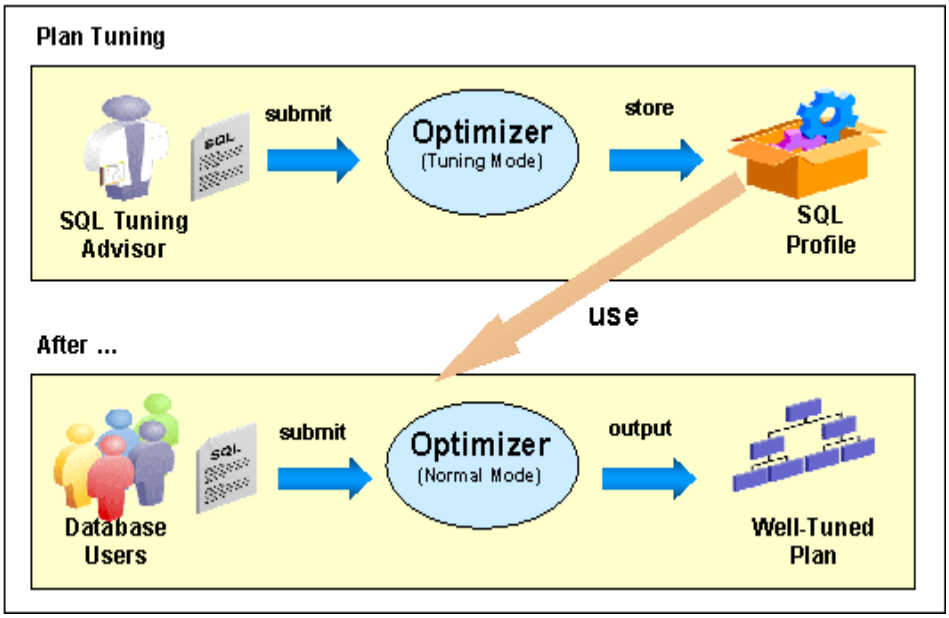


Figure 4: Creation and Use of SQL Profile

The auxiliary information contained in a SQL Profile is stored in such a way that it stays relevant in the face of database changes such as addition or removal of indexes, growth in the size of tables, and periodic collection of database statistics. However, a SQL Profile may not adapt to massive changes in the database, or changes that have been accumulating over a long period of time. In this case, a new SQL Profile needs to be built to replace the old one. For example, when a SQL Profile becomes outdated the performance of corresponding SQL statement may become noticeably worse. In such a case, the corresponding SQL statement may start showing up as high load or top SQL, thus becoming again a target for the Automatic SQL Tuning.

Access Path Analysis

The Automatic Tuning Optimizer also provides advice on indexes. Effective indexing is a well-known tuning technique that can significantly improve the performance of SQL statements by reducing the need for full data scans. Any index recommendations generated by the Automatic Tuning Optimizer are specific to the SQL statement being tuned, and therefore, it provides a quick fix to the performance problem associated with a single SQL statement.

Since the Automatic Tuning Optimizer does not do an analysis of how its index recommendation are going to affect the entire SQL workload, it also recommends running the Access Advisor on the SQL statement along with a

representative SQL workload. The Access Advisor collects advices given on each statement of a SQL workload, and consolidates them into a global advice for the entire SQL workload (see the paper "Turbocharge Your Database, Using the Oracle Database 10g SQL Access Advisor"").

SQL Structure Analysis

Often a SQL statement can be a high load statement simply because it is badly written. This usually happens when there are different (but not necessarily semantically equivalent) ways to write a statement to produce same result. For example, a SQL statement may produce same result when its UNION operator is replaced by UNION-ALL. The same result is possible if there is no possibility of producing duplicate rows making the duplicate elimination performed by the UNION operator redundant. If this is the case, it is prudent to replace with UNION-ALL thus eliminating an expensive duplicate elimination step from the execution plan. Another example is the use of NOT IN subquery while a NOT EXIST subquery could have produced same result much more efficiently.

Knowing which of these alternate forms is most efficient is a daunting task for the application developer because it requires both a deep knowledge about the data properties (e.g., there are no nulls in a column) as well as a very good understanding of the semantics (i.e., meaning) of SQL constructs.

Besides, during the development stage, developers are generally more focused on how to write SQL statements that produce desired result than improve the performance. Sometimes, a simple mistake can make a SQL statement perform very badly. A good example in this category is a type mismatch between the column and its predicate value, which essentially disables the use of an index even if one is available.

The Automatic Tuning Optimizer performs SQL structure analysis to detect poorly written SQL statements and recommends, subject to user verification of the result, possible alternative ways of rewriting the SQL statement to improve their performance. The developers can run SQL Tuning Advisor under Limited mode on a proactive basis to help them write better SQL statements.

Under SQL structure analysis mode the Automatic Tuning Optimizer generates extensive annotations and diagnostics during the plan building process and associates them to the execution plan. The annotations include the decisions made by the optimizer and the reasons for making those decisions. Using the reasons associated with costly operators in the execution plan the Automatic Tuning Optimizer gives recommendations either on how to rewrite the SQL statement or to make necessary schema changes to improve the performance.

There are various reasons related to the structure of the SQL statement that can cause poor performance. Some reasons are syntax-based, some are semantics-based, and some are purely design issues. We group these reasons into three categories:

Semantic-based Constructs: A construct such as NOT IN subquery, when replaced by a corresponding but not semantically equivalent NOT EXISTS subquery can result in a significant performance boost. However, this replacement is possible only if NULL values are not present in the related join columns, thus ensuring that same result is produced by either of these operators. Another example is the replacement of UNION with UNION ALL provided there is no possibility of getting duplicate rows in the result.

Syntax-based Constructs: Many of these are related to how predicates are specified in a SQL statement. For example, if a predicate such as col = :bnd is used with col and :bnd having different types, then such a predicate cannot be used as an index driver. Similarly, a predicate involving a function or expression (e.g. func(col) = :bnd, col + 1 = :bnd) will not be used as an index driver unless there is a functional index on the expression or function itself.

Design Issues: An accidental use of a Cartesian product, for example, is a common problem occurring when one of the tables is not joined to any of the other tables in a SQL statement. This can happen especially when the query involves a large number of tables.

SQL TUNING SETS

The ADDM enables automatic identification of high load SQL statements for the user to choose from and tune them. The AWR enables the selection of top SQL statements over a time interval. However, the user may like to tune a set of custom SQL statements in a user-specified priority order. A good example of this situation is when a developer is in the process of developing new SQL statements and testing them. Since this SQL is not yet deployed into the system the usual SQL sources (e.g., AWR) cannot be used. So there needs to be a mechanism for the user to create his own custom SQL workload and perform Automatic SQL Tuning on it. The answer is the SQL Tuning Set (STS). The support for STS has been introduced in Oracle Database 10g to make it easy for the user to manage a related set of SQL statements as a unit.

A SQL Tuning Set is a database object that stores one or more SQL statements along with their execution statistics and execution context, and possibly with user priority ranking. The SQL statements can be loaded into a SQL Tuning Set from different SQL sources. The SQL sources include the Automatic Workload Repository (AWR), the cursor cache, and the custom SQL provided by the user. Figure 5 shows a typical SQL Tuning Set usage model. A SQL Tuning Set is created and then loaded with SQL statements from one of the SQL sources. As illustrated, the SQL statements are selected, ranked and filtered before they are loaded into a SQL Tuning Set.

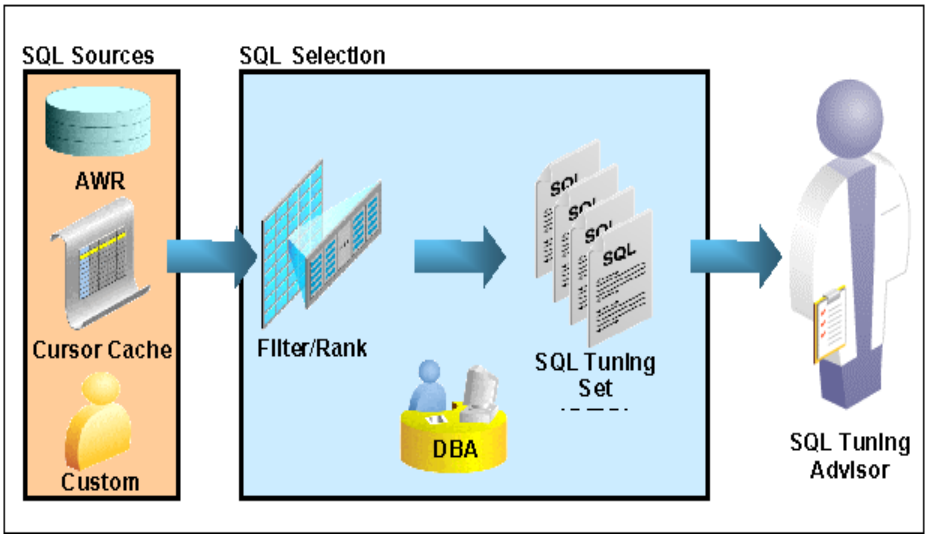


Figure 5: SQL Tuning Set Usage Model

The execution context stored with each SQL statement includes the user schema, application module name and action, list of bind values, and the cursor compilation environment. The execution statistics stored include elapsed time, CPU time, buffer gets, disk reads, rows processed, cursor fetches, the number of executions, the number of complete executions, optimizer cost, and the command type. The filtering of SQL statements can be done using the application module name and action, or any of the execution statistics. Following this, the ranking of the SQL statements can be performed based on any combination of execution statistics.

A SQL Tuning Set can be stored persistently in the database. The contents of a SQL Tuning Set can be updated, as well as the STS as an object can be manipulated through the use of DBMS_SQLTUNE procedures. Once loaded a SQL Tuning Set can be passed as an input to the SQL Tuning Advisor, which then performs automatic tuning of the SQL statements subject to other input parameters specified by the user. Other inputs include a time limit, the scope of tuning analysis, etc. Thus a SQL Tuning Set provides a powerful method of collecting and storing an interesting set of SQL statements along with lots of supporting information that aids in the tuning process

SQL TUNING INTERFACE

Enterprise Manager (EM) can be used to identify the high-load SQL statements. There are several places in EM from where the SQL Tuning Advisor can be launched with the identified SQL statement(s), or a SQL Tuning Set.

Tuning ADDM SQL

The following EM screen shows high load SQL statements identified by the Automatic Database Diagnostics Monitor (ADDM). Each of these high load SQL statements are known to consume a significant proportion of one or more system resources such as CPU time, buffer gets, disk reads, etc. This screen lets the user launch the SQL Tuning Advisor on a selected high load SQL statement.

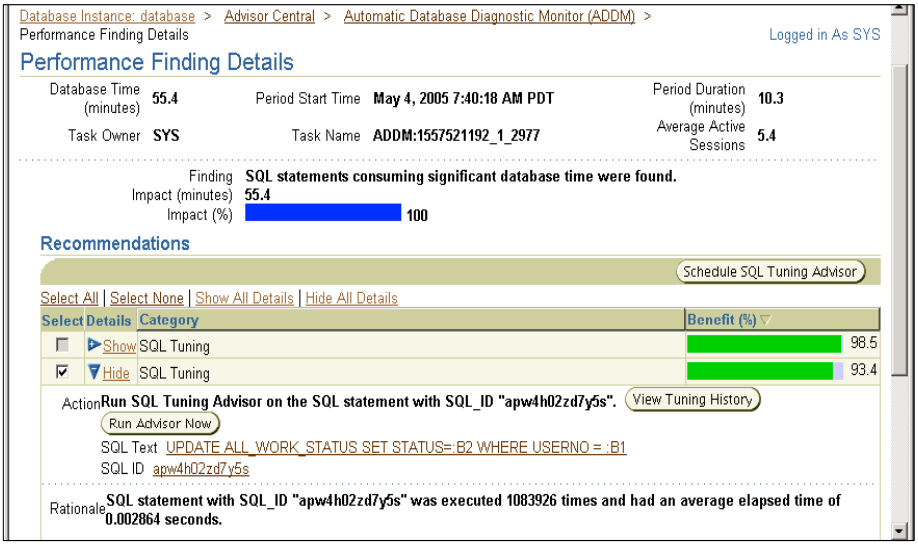
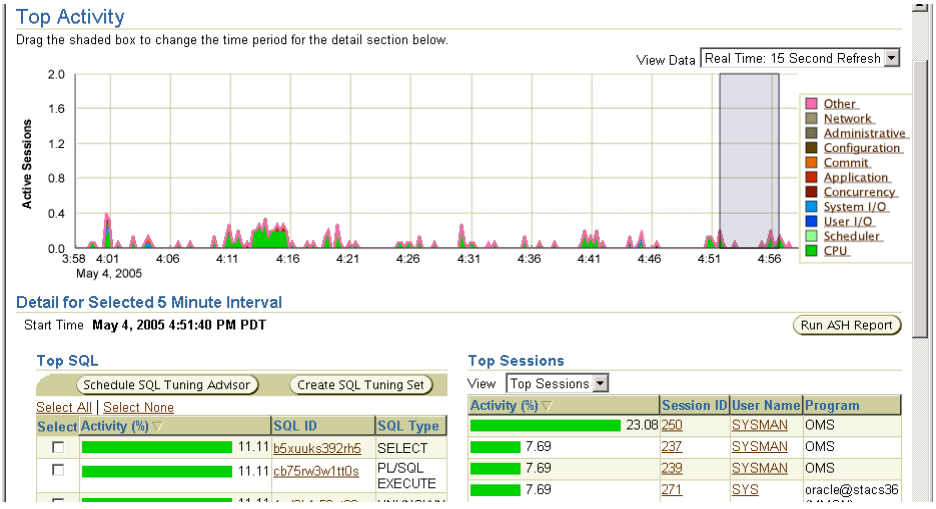


Figure 6: ADDM Finding High Load SQL

Tuning Top SQL

Another SQL source is the list of top SQL statements as shown by the Enterprise Manager screen in figure 7. The list of top SQL statements is identified by looking at their cumulative execution statistics based on a selected time window. The top SQL statements can be ranked on an associated statistics such as CPU consumption. The user can select one or more top SQL statements



identified by

Figure 7: Top SQL

their SQL ID and launch the SQL Tuning Advisor on them.

Tuning an STS

The Enterprise Manager also lets you look at various SQL Tuning Sets created by different users. An STS could have been created from a list of top SQL statements, or by selecting SQL statements from a range of snapshots created by Automatic Workload Repository (AWR), or by creating own customized SQL statements. Figure 8 shows how to launch the SQL Tuning Advisor on a selected STS in Enterprise Manager.

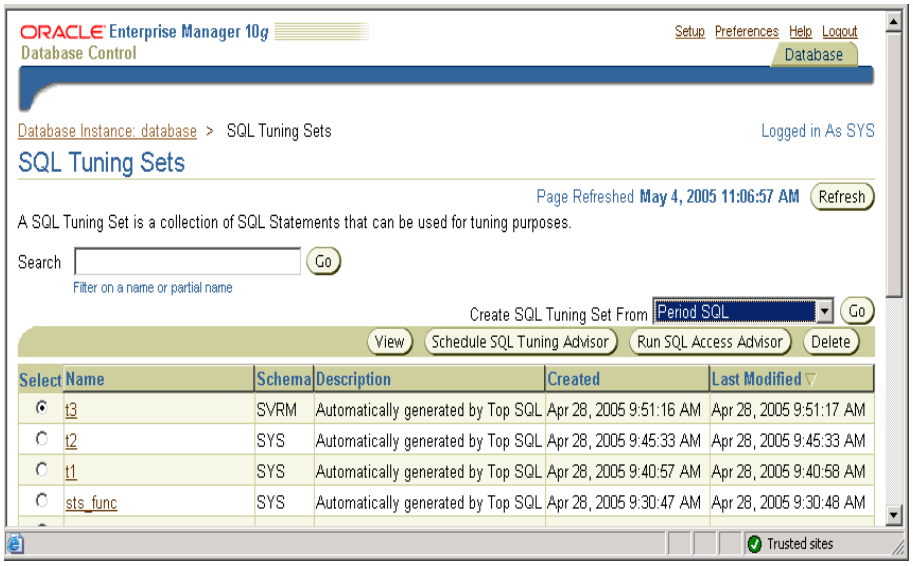


Figure 8: SQL Tuning Sets

Tuning Options

Once the SQL Tuning Advisor is launched, the Enterprise Manager will automatically create a tuning task provided the user has appropriate ADVISOR privilege to do so. The Enterprise Manager shows the tuning task with automatic defaults in the SQL Tuning Options screen shown in Figure 9. On this screen the user has the options to change the automatic defaults pertaining to a tuning task. One of the important options is to choose the scope of the tuning task. If you choose the Limited option then the SQL Tuning Advisor produces recommendations based on statistics check, access path analysis, and SQL structure analysis. No SQL Profile recommendation will be generated with Limited scope. If you choose Comprehensive option the SQL Tuning Advisor will do all recommendations under Limited scope plus it will invoke the optimizer under SQL profiling mode to build a SQL Profile if applicable. With Comprehensive option you can also specify a time limit for the tuning task, which by default is 60 minutes. Another useful option is to run the tuning task immediately or schedule it to be run at a later time.

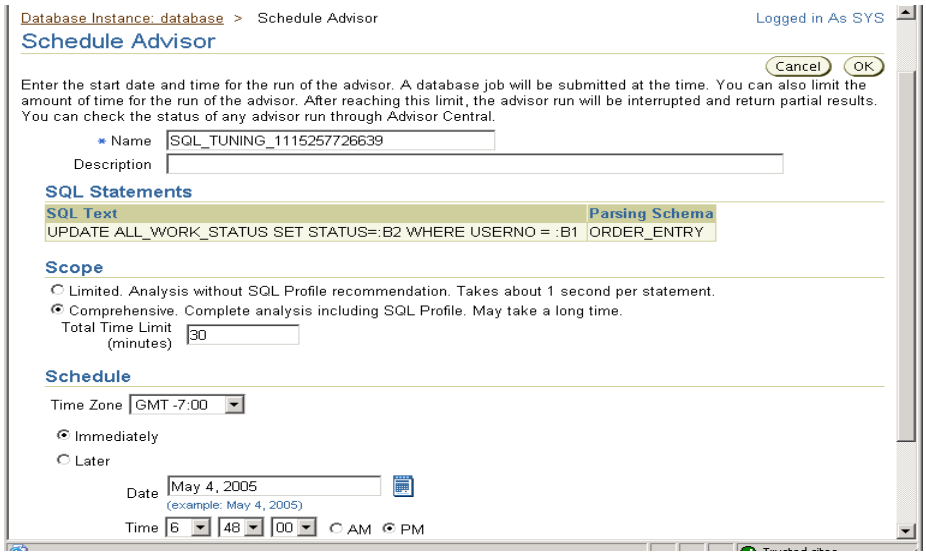


Figure 9: SQL Tuning Advisor Options Screen

Viewing SQL Tuning Recommendations

Once a tuning task completes, the recommendations generated by the SQL Tuning Advisor can be viewed. The Enterprise Manager shows both an overview of recommendations as well as recommendation details. The following screen shows an overview of the SQL Tuning Advisor recommendations for one or more SQL statements tuned. As shown below, there is only one recommendation, to create an index, for the SQL statement. If you select a SQL statement and click on View Recommendations button the Enterprise Manager will show you details on the recommendations.

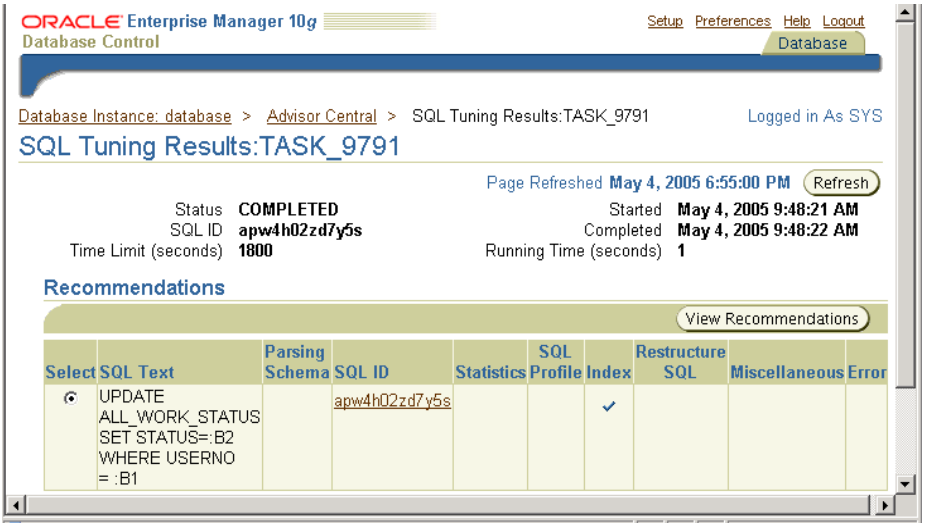


Figure 10: SQL Tuning Recommendations Overview

You can accept this recommendation by clicking on the Implement button to create the index.

Figure 11: SQL Tuning Recommendations Details

DBMS_SQLTUNE PACKAGE

While the primary interface for Automatic SQL Tuning is the Oracle Enterprise Manager, a command line interface to the DBMS_SQLTUNE package can also be used for tuning SQL statements. DBMS_SQLTUNE is a new package added in Oracle Database 10g, and it contains necessary APIs for using Automatic SQL Tuning features, including tasks to perform automatic tuning of statements, and management of SQL Profiles and SQL Tuning Sets.

Tuning Task Management

It is important to note that the SQL Tuning Advisor, like all other manageability advisors, is built on top of a common Advisor Framework. The Advisor Framework provides a common infrastructure support to build, store, and retrieve advice generated by various manageability features including the SQL Tuning Advisor. Therefore, all SQL tuning procedures operate with advisor task objects called tuning tasks. This means a tuning task needs to be created in order to perform automatic tuning. The use of SQL tuning procedures including the creation of tuning tasks requires the ADVISOR privilege.

To perform automatic SQL tuning using the DBMS_SQLTUNE package, the first step always is to create a tuning task by calling the *create_tuning_task* procedure. This procedure creates an advisor task and sets its corresponding parameters according to the user provided input arguments.

There are several flavors of the *create_tuning_task* procedure that can be used to create tuning tasks to tune either a single SQL statement or multiple statements stored in a SQL Tuning Set.

The following example illustrates one form of the *create_tuning_task* procedure that allows to directly pass a SQL statement text as an argument. In this example, the statement text is passed as a CLOB.

```
  create_tuning_task(sql_text    => 'select * from emp
where emp_id = :bnd',
                     bind_list   =>
sql_binds(anydata.ConvertNumber(100)),
                     user_name   => 'scott',
                     scope       => 'comprehensive',
                     time_limit  => 60,
                     task_name   =>
'my_sql_tuning_task',
                     description => 'task to tune a
query on a specified employee');
```

In the example the target statement uses a bind variable *bnd* whose value (100) is a number passed as a function argument of type SQL_BINDS. SQL_BINDS is a new object type introduced in Oracle Database 10g. The parameter *scott* represents the name of the schema where the statement is analyzed. The tuning scope of this task is passed as *comprehensive* to tell SQL Tuning Advisor to do complete analysis including SQL Profile generation. And finally, argument *60* is the time limit in seconds to tune the SQL statement.

There are two other forms of this procedure available to target a particular SQL statement selected from either the cursor cache or the Automatic Workload Repository (AWR). In either case, the statement will be identified by passing its SQL_ID in place of the SQL text.

Once the tuning task is successfully created, it is at an initial state. The task then needs to be executed in order to start the tuning process. This is achieved by invoking the *execute_tuning_task* procedure as follows:

```
execute_tuning_task(task_name => my_sql_tuning_task');
```

At any time after the execution has begun, the user can use an appropriate advisor procedure to cancel, interrupt or reset the task. The user can also check the status of the task by reviewing the information placed in DBA_ADVISOR_LOG performance view, or can query the V$SESSIO_LONGOPS view to display information about the task execution progression made so far. This information includes remaining execution time, number of findings, benefit, and number of statements processed in case of tuning a SQL Tuning Set.

When the tuning task is completed, the tuning results can be visualized by calling the *report_tuning_task* procedure as shown below.

```
  set long 10000;
 select report_tuning_task(task_name =>
'my_sql_tuning_task')from dual;
```

The above SELECT produces a textual report (of type CLOB) of all findings and recommendations based on automatic tuning of the SQL statement, along with rationale and benefit for each proposed recommendation and the SQL commands to implement each recommendation. The User can accept a recommendation by simply executing the command associated with that recommendation.

Automatic SQL Tuning results can also be viewed using the DBA advisor framework views such as DBA_ADVISOR_TASKS, DBA_ADVISOR_FINDINGS, DBA_ADVISOR_RECOMMENDATIONS, DBA_ADVISOR_RATIONALE, etc. Besides these views, which are common to all advisors in Oracle Database 10g, Automatic SQL Tuning extends the framework by adding new views that can display specific SQL tuning information such as SQL statistics, associated binds and execution plans. To check such results the user can query DBA_SQLTUNE_STATISTICS, DBA_SQLTUNE_BINDS, and DBA_SQLTUNE_PLANS views.

SQL Profile Management

The procedures for managing SQL Profiles are also part of the DBMS_SQLTUNE package. When a SQL Profile is recommended by the SQL Tuning Advisor the SQL Profile can be created by calling *accept_sql_profile* procedure which will store it in the data dictionary. To create a SQL Profile requires CREATE ANY SQL PROFILE privilege. Once

created, the SQL Profile will be automatically applied on the next execution of the same SQL statement. For example, the following procedure call stores a SQL Profile produced by automatic tuning of a SQL statement associated with the tuning task *my_sql_tuning_task*. In this example, the SQL Profile is given a name *my_sql_profile*.

```
accept_sql_profile(task_name => 'my_sql_tuning_task',
                   name      => 'my_sql_profile');
```

The information about SQL Profiles can be viewed using DBA_SQL_PROFILES view. The user can also alter the attributes of an existing SQL Profile by executing the `alter_sql_profile` procedure. To do this, it requires ALTER ANY SQL PROFILE privilege.

In the following example, the status attribute of SQL Profile *my_sql_profile* is changed to *disabled*, which means the SQL Profile will no longer be used in the generation of execution plan of corresponding SQL statement.

```
alter_sql_profile(name           => 'my_sql_profile',
                  attribute_name => 'status',
                  value          => 'disabled');
```

The other SQL Profile attributes that can be altered are name, description, and category. Finally, a SQL Profile can be removed from the data dictionary by using *drop_sql_profile* procedure. This requires DROP ANY SQL PROFILE privilege.

SQL Tuning Set Management

A typical usage scenario of using the DBMS_SQLTUNE package for a SQL Tuning Set (or simply sqlset) will involve creating a new SQL Tuning Set, load it with a set of high load SQL statements, select and browse its content for manual analysis and further refining and selection, then run SQL Tuning Advisor to automatically tune all statements in the SQL Tuning Set, and finally drop the SQL Tuning Set after implementing the SQL Tuning Advisor recommendations.

In the following example the `create_sqlset` procedure creates a SQL Tuning Set called `my_sql_tuning_set` that can be used to load the I/O intensive SQL statements collected during a specific time period.

```
create_sqlset(sqlset_name => 'my_sql_tuning_set',
              description => 'I/O intensive
workload');
```

This procedure creates an empty SQL Tuning Set in the database. Notice that to execute SQL Tuning Set procedures, a user must have been granted the

ADMINISTER SQL TUNING SET, or ADMINISTER ANY SQL TUNING SET privilege.

After creating the SQL Tuning Set, the procedure *load_sqlset* can be used to populate it with selected SQL statements. The standard sources for populating a SQL Tuning Set are the Automatic Workload Repository (AWR), the cursor cache, or another SQL Tuning Set that was created and loaded earlier. For each of these sources, there are predefined table functions that can be used to extract and filter the source content before loading it into a new SQL Tuning Set.

For example, the following procedure calls are used to load *my_sql_tuning_set* from an AWR baseline called "*peak baseline*", by choosing those SQL statements that have been executed at least 10 times and they have a (disk-reads/buffer-gets) ratio of greater than 50% during the baseline period. The SQL statements are ordered by (disk-reads/buffer-gets) ratio and only the top 30 SQL statement are chosen.

```
  -- open a ref cursor to select from the specified
baseline
  open baseline_ref_cursor for
    select value(p)
    from table (dbms_sqltune.select_baseline(
                  'peak baseline',
                  'executions >= 10 and
disk_reads/buffer_gets >= 0.5',
                   null,
                   disk_reads/buffer_gets,
                   null, null, null,
                   30)) p;
  -- load statements and their stats from the baseline
into the STS
  dbms_sqltune.load_sqlset(sqlset_name     =>
'my_sql_tuning_set',
                           populate_cursor =>
baseline_cur);
```

Now that the SQL Tuning Set has been created and populated, the DBA can browse through the SQL statements in the SQL Tuning Set using the *select_sqlset* procedure as follows:

```
  SELECT * from TABLE(select_sqlset('my_sql_tuning_set',

'(disk_reads/buffer_gets) >= 0.75'));
```

In this example, only the SQL statements with (disk reads/buffer gets) ratio > 75% in the SQL Tuning Set are displayed.

The details of the SQL Tuning Sets that have been created and loaded can also be displayed using the DBA views such as DBA_SQLSET, DBA_SQLSET_STATEMENTS and DBA_SQLSET_BINDS.

SQL statements can also be updated and deleted from a SQL Tuning Set based on a search condition. For example, the following *delete_sqlset* procedure

will delete SQL statements from *my_sql_tuning_set* that have been executed less than fifty times.

```
delete_sqlset(sqlset_name  => 'my_sql_tuning_set',
              basic_filter => 'executions < 50');
```

Finally, when a SQL Tuning Set is no longer required (for example, after tuning all the statements it contains and implementing necessary recommendations) it can be dropped using the *drop_sqlset* procedure as follow:

```
drop_sqlset(sqlset_name => 'my_sql_tuning_set');
```

CONCLUSION

In this paper, we have described the Automatic SQL Tuning manageability component that has been introduced in Oracle Database 10g. The Automatic SQL Tuning provides automatic tuning of SQL statements in the form of a set of comprehensive tuning recommendations. It is tightly integrated with the query optimizer. In fact, the query optimizer runs under a special automatic tuning mode and generates tuning recommendations. In addition to recommendations it may also build a SQL Profile when it is appropriate to do so. The user can choose to implement the recommendations including the SQL profile. Once a SQL Profile is created, the query optimizer will use it to generate a well-tuned plan for the corresponding SQL statement. A tuning object called the SQL Tuning Set is also introduced that enables a user to create custom SQL workload for tuning purpose. The interface to the Automatic SQL Tuning has been created using the Enterprise Manager with options to select from different SQL sources, and tune SQL statements with different tuning options.

We conclude this paper by showing you how the Automatic SQL Tuning greatly simplifies the tuning process. A common problem observed is the degradation in performance of a SQL statement as the data grows over time. For a SQL statement embedded in a packaged application the following table compares typical SQL tuning steps performed in Oracle9i and Oracle Database 10g.

Steps	Oracle9*i*	Oracle Database 10g
1	Get explain plan	Run SQL Tuning Advisor
2	Examine query objects and their sizes	Implement recommendation
3	Review and compare explain plan statistics with execution statistics (stored in V$SQL view)	
4	Identify that it is a “first rows” issue because only recent data is ever displayed despite large history being queried	
5	Contact application vendor	
6	Produce test case for vendor	
7	Get a patch with “first rows” hint from the vendor	
8	Install the patch in next maintenance cycle	

As the table above shows, the effort and time spent by a tuning expert in this fairly common task is considerably greater in Oracle9i compared to Oracle Database 10g. Furthermore, in Oracle9i the customer has to wait for the application vendor to provide the patch, which can take weeks or even months, whereas in Oracle Database 10g the problem resolution is immediate. Automatic SQL Tuning offers a comprehensive yet easy to use solution for application tuning, which can be used equally effectively by novice and expert users.

APPENDIX: ORACLE DIAGNOSTIC AND TUNING PACKS

The sections below describe the Oracle Diagnostic and Tuning packs. These packs can be purchased only with Enterprise Edition. The features in these packs are accessible through Oracle Enterprise Manager Database Control, Oracle Enterprise Manager Grid Control, and APIs provided with Oracle Database software.

Oracle Diagnostic Pack

The Oracle Diagnostic Pack provides automatic performance diagnostic and advanced system monitoring functionality. The Diagnostic Pack includes the following features:

- Automatic Workload Repository
- Automatic Database Diagnostic Monitor (ADDM)
- Performance monitoring (database and host)
- Event notifications: notification methods, rules, and schedules
- Event history and metric history (database and host)
- Blackouts
- Dynamic metric baselines
- Monitoring templates

In order to use the features listed above, you must purchase licenses for the Diagnostic Pack. The Diagnostics Pack functionality can be accessed by Enterprise Manager links as well as through the database server command-line APIs. The use of either interface requires a Diagnostic Pack license.

Enterprise Manager

To determine which links in Enterprise Manager Grid Control and Database Control are part of the Diagnostics Packs, click the Setup link on the top right-hand part of the Enterprise Manager Home page.

- When you click the Setup link, the navigation bar contains the Management Pack Access link. Click this link.
- This will take you to the Management Pack Access page, which allows you to grant and remove access from all the management packs.
    - For Enterprise Manager Database Control, click the Remove Access radio button for the Diagnostic Pack and click Apply.
    - For Enterprise Manager Grid Control, click the appropriate check box for the Diagnostic Pack and click Apply.

This will disable all the links and tabs associated with the Diagnostics Pack in Enterprise Manager. All the disabled links and tabs are part of the Diagnostics Pack and therefore require pack license.

Command-Line APIs

Diagnostics Pack features can also be accessed by way of database server APIs and command-line interfaces:

- The DBMS_WORKLOAD_REPOSITORY package is part of this pack.
- The DBMS_ADVISOR package is part of this pack if you specify ADDM as the value of the advisor_name parameter, or if you specify for the value of the task_name parameter any value starting with the ADDM prefix.
- The V$ACTIVE_SESSION_HISTORY dynamic performance view is part of this pack.
- All data dictionary views beginning with the prefix DBA_HIST_ are part of this pack, along with their underlying tables.
- All data dictionary views with the prefix DBA_ADVISOR_ are part of this pack if queries to these views return rows with the value ADDM in the ADVISOR_NAME column or a value of ADDM* in the TASK_NAME column or the corresponding TASK_ID.
- The following reports found in the /rdbms/admin/ directory of the Oracle home directory are part of this pack: awrrpt.sql, awrrpti.sql, addmrtp.sql, addmrpti.sql, awrrpt.sql, awrrpti.sql, addmrpt.sql, addmrpti.sql, ashrpt.sql, ashrpti.sql, awrddrpt.sql, awrddrpi.sql, awrsqrpi.sql, awrsqrpt.sql.

Oracle Diagnostics Pack Enterprise Manager Repository Views

- Monitoring Views
  - MGMT$BLACKOUT_HISTORY
  - MGMT$BLACKOUTS
  - MGMT$ALERT_ANNOTATIONS
  - MGMT$ALERT_NOTIF_LOG
  - MGMT$TARGET_METRIC_COLLECTIONS
  - MGMT$METRIC_COLLECTIONS
  - MGMT$TARGET_METRIC_SETTINGS
  - MGMT$AVAILABILITY_CURRENT
  - MGMT$AVAILABILITY_HISTORY

    - MGMT$ALERT_CURRENT
    - MGMT$ALERT_HISTORY
    - MGMT$METRIC_DETAILS
    - MGMT$METRIC_CURRENT
    - MGMT$METRIC_HOURLY
    - MGMT$METRIC_DAILY
  - Template Views
    - MGMT$METRIC_DAILY
    - MGMT$TEMPLATES
    - MGMT$TEMPLATE_POLICY_SETTINGS
    - MGMT$TEMPLATE_METRIC_COLLECTION
    - MGMT$TEMPLATE_METRIC_SETTINGS

Oracle Tuning Pack

The Oracle Tuning Pack provides database administrators with expert performance management for the Oracle environment, including SQL tuning and storage optimizations. The Oracle Diagnostic Pack is a prerequisite product to the Oracle Tuning Pack. Therefore, to use the Tuning Pack, you must also have a Diagnostic Pack. The Tuning Pack includes the following features:

- SQL Access Advisor
- SQL Tuning Advisor
- SQL Tuning Sets
- Reorganize objects

In order to use the features listed above, you must purchase licenses for the Tuning Pack. The Tuning Pack functionality can be accessed by the Enterprise Manager links as well as through the database server command-line APIs. The use of either interface requires licensing of the Tuning Pack.

Enterprise Manager

To determine which links in Enterprise Manager Grid Control and Database Control are part of the Diagnostics Packs, click the Setup link on the top right-hand part of the Enterprise Manager Home page.

- When you click the Setup link, the navigation bar contains the Management Pack Access link. Click this link.

- This will take you to the Management Pack Access page, which allows you to grant and remove access from all the management packs.
    - For Enterprise Manager Database Control, click the Remove Access radio button for the Tuning Pack and click Apply.
    - For Enterprise Manager Grid Control, click the appropriate check box for the Tuning Pack and click Apply.

This will disable all the links and tabs associated with the Tuning Pack in Enterprise Manager. All the disabled links and tabs are part of the Tuning Pack and therefore require pack license.

Command-Line APIs

Tuning Pack features can also be accessed by way of database server APIs. Use of the following PL/SQL packages requires a license for the Oracle Tuning Pack:

- `DBMS_SQLTUNE`
- `DBMS_ADVISOR`, when the value of the advisor_name parameter is either `SQL Tuning Advisor` or `SQL Access Advisor`.

The following report found in the /rdbms/admin/ directory of the Oracle home directory is part of this pack: sqltrpt.sql.

White Paper Title: The Self-Managing Database: Guided Application & SQL Tuning with Oracle Database 10g Release 2
May 2005
Author:Mohammed Ziauddin, Mohammed Zait, Mughees Minhas, Khaled Yagoub
Contributing Authors:Benoit Dageville

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
www.oracle.com

Oracle Corporation provides the software
that powers the internet.