Oracle® Tuning Pack

An Oracle Technical White Paper

November 1998

Enabling the Information Age®

Oracle Tuning Pack

OVERVIEW

Oracle performance tuning is an ongoing process that involves:

- Identifying Problem Areas
- Gathering and Analyzing Data
- Determining Options and Tradeoffs
- Implementing Changes
- Repeating the Cycle

Tuning must encompass a wide spectrum; the global level, such as the proper allocation of system resources; and the highly focused level, such as the tuning of individual SQL statements. Effective database tuning requires good methods and tools. Ideally, the tools should augment and promote the tuning methodology.

Oracle Tuning Pack is a set of database tuning applications that provide both in-depth tuning functionality, and an effective tuning methodology. Built on the Oracle® Enterprise Manager systems management platform, Oracle Tuning Pack applications provide comprehensive Oracle tuning, focused on the major Oracle performance opportunities:

- Identifying And Tuning High Impact Application SQL
- Optimizing Data Access Paths
- Proper Use And Maintenance Of Database Structures
- Optimal Use Of System Resources Through Well Planned Instance Parameters
- Taking Advantage Of New Oracle Performance Features

Oracle Tuning Pack provides this in-depth tuning functionality through three applications:

- *Oracle Expert*™, for Automated Database Tuning
- *Oracle SQL Analyze*, for Tuning Application SQL
- *Oracle Tablespace Manager*, for Correcting Data Storage Problems

This paper examines the functionality and use of the Oracle Tuning Pack applications.

ORACLE EXPERT

Monitoring the performance of production databases is basic to the survival of every database administrator (DBA). Unfortunately, this is where most performance management applications end, and the DBA is left to figure out the means to correct the problem that the monitor detected. Database tuning is an essential follow-through to performance monitoring. Tuning is the only way of ensuring that you are getting maximum database performance and utilization of the system resources supporting your database.

Unfortunately, because it is a complex, time-consuming process, most database tuning is reactive, rather than proactive. Performance problems arise unpredictably and must be dealt with immediately. Tuning requires time, specialized skills, and adherence to a structured methodology for monitoring performance, collecting data, and implementing changes.

To meet this need, Oracle Tuning Pack provides Oracle Expert, an application for automated performance tuning. Performance problems that Oracle® Diagnostics Pack and other Oracle monitoring applications detect can be analyzed and solved with Oracle Expert. Oracle Expert automates the process of collecting and analyzing data, and contains a rules-based inference engine that provides “expert” database tuning recommendations, implementation scripts, and reports.

Oracle Expert provides a structured, yet flexible approach to Oracle database tuning. The process is structured in that the product follows a logical, defined tuning process. It is flexible in that the user can select the tuning focus that best meets their needs. This flexibility allows Oracle Expert to be useful for a wide range of users; from the novice DBA to the expert consultant.

The Tuning Session

Oracle Expert organizes its functions within the concept of a “tuning session.” Each tuning session encapsulates all of the knowledge and data required to tune the database. The user specifies the focus for a given session, a global focus for overall database performance optimization, or a focused approach for tuning specific parts of the database. It is highly recommended that focused tuning be used, because it reduces the amount of data to be collected and analyzed. Equally important, a focused approach provides the user with a more manageable set of Oracle Expert tuning results, including extensive tuning reports and implementation scripts.

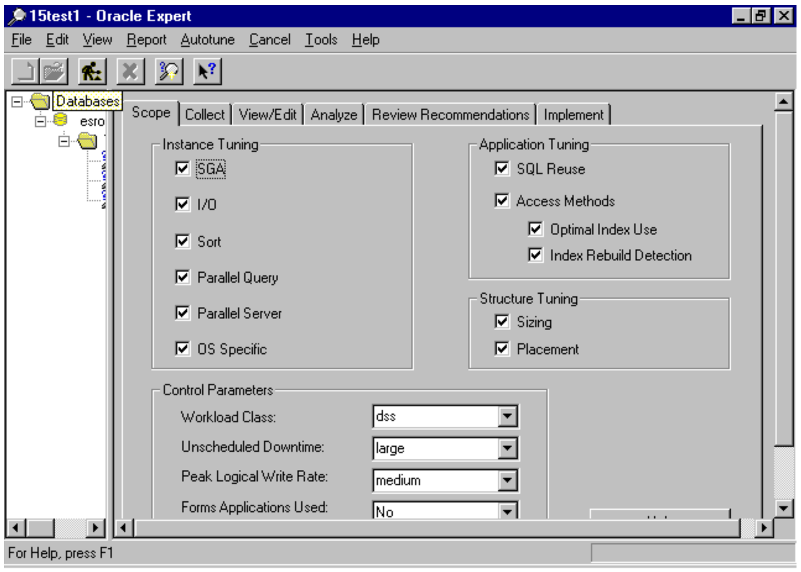
Oracle Expert monitors several factors in the database environment. It also provides tuning recommendations that take advantage of Oracle’s performance features, and optimizes the use of the systems resources. This can be broken down into three major categories:

- Database instance tuning for the shared pool, input/output (I/O) and sort operations, parallel query, and Oracle® Parallel Server.
- Application database access optimization, such as index tuning, index re-organization, and SQL reusability. Oracle Expert’s index tuning functionality is integrated with the Oracle Cost-based Optimizer, to ensure that recommended indexes will provide the most efficient, low cost access methods for the database workload.
- Database structure tuning, for storage sizing and placement of database objects, including index, rollback and temporary segments, as well as the identification of tables that should be rebuilt and recommendations for their proper storage attributes.

Oracle Expert also provides platform-specific Oracle tuning. The Oracle server is often optimized to take advantage of performance features available for specific platforms. Rules are being added to

Oracle Expert that tune Oracle platform-specific database parameters, such as the ASYNC_WRITE parameter. This platform-specific parameter enables asynchronous writer operations, allowing Oracle to continue executing without waiting for I/O requests to complete.

By tuning physical database considerations, parameter settings, and application access methods, Oracle Expert solves problems, such as poor resource utilization, excessive I/Os, and database contention. In addition, Oracle Expert will ensure that all applicable database performance features are used. DBAs can use Oracle Expert to tune several databases simultaneously, or to try several different tuning approaches for the same database.



Instance Tuning
- ✓ SGA
- ✓ I/O
- ✓ Sort
- ✓ Parallel Query
- ✓ Parallel Server
- ✓ OS Specific

Application Tuning
- ✓ SQL Reuse
- ✓ Access Methods
  - ✓ Optimal Index Use
  - ✓ Index Rebuild Detection

Structure Tuning
- ✓ Sizing
- ✓ Placement

Control Parameters

Workload Class:	dss
Unscheduled Downtime:	large
Peak Logical Write Rate:	medium
Forms Applications Used:	No

Figure 1: Oracle Tuning Pack Allows You to Easily Create Consultant Quality Tuning Recommendations Customized for Your Own Specific Operating Environment

Autotune Mode

Oracle Expert provides the option of running in “Autotune” mode. The Autotune feature allows Oracle Expert to operate in a continuous sampling/analysis mode for one or more databases, automatically searching for tuning opportunities and generating tuning recommendations. Autotune currently supports tuning several of the top initialization parameters.

Easy Index Tuning

Oracle Expert’s comprehensive index tuning capabilities can be easily accessed through the Index Tuning Wizard. This feature provides a quick launch point for performing index tuning that can be used from Oracle Expert and SQL Analyze. The Index Tuning Wizard guides the user through the process, analyzing targeted tables for index tuning, based upon the comprehensive database workload described below.

Oracle Expert Data Collection

The amount of data required to effectively tune a database is enormous. Data must be pulled from multiple sources, quickly and efficiently, with no influence on the production environment. The volume of data required, and the complexity of the collection process is often enough to discourage regular database tuning. Oracle Expert solves this dilemma by collecting the data required for database tuning. Data is collected in a systematic, non-intrusive way and stored in an Oracle Expert repository for future processing. Oracle Expert collects the following categories of performance tuning data:

- Database Schema and Instance Data
- Database Workload
- System Environment

The database schema and instance data is extracted from the Oracle data dictionary, V$ dynamic performance tables, and other sources. Instance data includes the current instance parameter settings and V$table statistics, which will be used to analyze the optimal settings for important instance parameters. Oracle Expert allows the user to decide how many instance data collections are to be performed for a given tuning session. As with most empirical analysis, the more data available for instance tuning, the better the results. Multiple data samples can be collected automatically by Oracle Expert over the course of several days. If Oracle Expert receives over 100 samples for a given tuning session, it has the intelligence to safely recommend de-allocating system resources for the database instance, if appropriate.

Database workload is a key factor in many tuning decisions, such as assessing the relative importance of a particular table, which is then used to determine the table's placement, and the type and number of indexes required. Database workload data includes the volume and type of application transactions that use the database. Oracle Expert provides the capability of collecting and compiling a comprehensive database workload profile. Workload data can be collected by periodically capturing all of the SQL statements in the SGA library cache. Optionally, a comprehensive profile of all workload transactions can be collected for Oracle Expert by Oracle Trace. Oracle Trace collects statistics on the volume of application transactions and requests for a given database, including the SQL statements that generated the workload transaction. Whether captured from the cache or through Oracle Trace, all of the workload collections are aggregated into a comprehensive database workload history, which can serve as the basis of highly effective index tuning. Moreover, this comprehensive workload capability is integrated with the Oracle SQL Analyze product, providing a shared workload basis for tuning SQL and database access methods.

System environment information collected by Oracle Expert includes system configuration data (such as number and capacity of disks), memory, CPU power, and so forth. A thorough picture of the power and various capacities of the system environment is essential to database tuning decisions, such as the allocation of memory to the various instance memory structures.

The Inference Engine and Tuning Rules

The heart of Oracle Expert is the inference rules engine. The rules engine contains a large set of database tuning rules that are applied to the data that Oracle Expert collects. Database performance experts from Oracle's development and consulting groups designed these rules. The rules engine evaluates the collected data and produces a set of recommend changes, including an explanation of each recommendation.

For example, as mentioned earlier, one of the primary factors in database performance is memory management. If the system's available memory is inadequate, or if the database does not take full advantage of available memory, then database performance will suffer. The database buffer cache hit ratio is one of the common metrics used to monitor effective memory management. The buffer cache hit ratio can be improved by increasing the number of database buffers. This is controlled by the INIT.ORA parameter, DB_BLOCK_BUFFERS. Increasing this parameter results in a reduction in I/Os, and increased performance. However, the DBA can not just arbitrarily increase this parameter; other dependencies must be considered. Information is required on available memory and SGA shared pool size. Oracle Expert will collect sample data on available memory, check SGA size, and will recommend changes to the DB_BLOCK_BUFFERS parameter, if possible. Oracle Expert will also document the logic and data supporting the recommendation.

This is one example of the type of analysis performed by the inference engine. Oracle Expert uses hundreds of rules and analytical algorithms to produce performance tuning recommendations, ranging from index tuning to physical database management. Future releases of Oracle Expert may enable users to define and add rules to the inference engine as well.

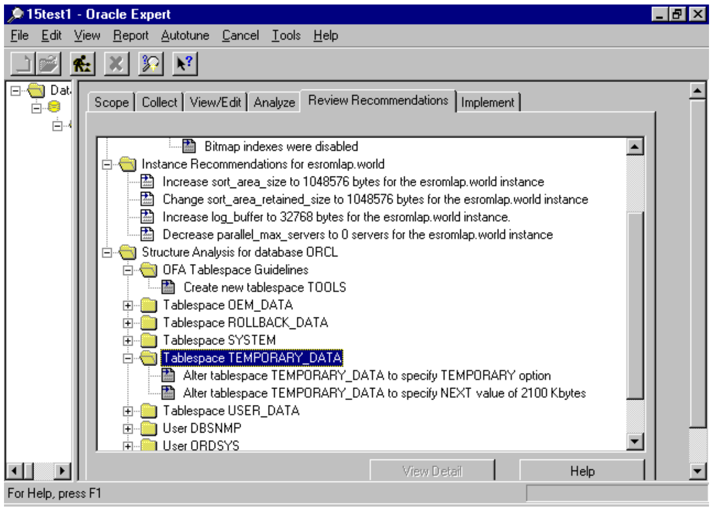
User Input and Validation

Oracle Expert provides the user with a flexible, interactive tuning environment. Users are able to graphically view and edit the collected tuning data. This helps them verify that all data necessary for a proper analysis was collected. However, this feature also allows the user to play a part in the analysis and recommendations that Oracle Expert produces. Users can customize the collected data, in order to test various "what-if" tuning scenarios, and to influence the final results. For example, to see how an increase or decrease in a parameter, such as OPEN_CURSORS, would effect Oracle Expert's analysis, the user can change the database initialization parameters that Oracle Expert collects. Or, in order to gauge the affect of an application workload change on the current access methods available to the application, a user can change the frequency of a particular SQL statement's occurrence.

User Defined Rules

Oracle Expert contains several hundred database tuning rules that the inference engine uses. The user can customize approximately one-third of these rules. Rules can be modified at the Oracle Expert user level such that they are used in every tuning session, or they can be customized for tuning a specific object, such as a table or index.

User-defined rules are available for every type of tuning that Oracle Expert performs. For example, in tuning the database access method, Oracle Expert evaluates the SQL workload and object statistics, to determine if any indexes require rebuilding. Oracle Expert evaluates the indexes targeted for tuning. If an index exceeds a height threshold, it may qualify for further rebuild evaluation. Oracle Expert provides a default height threshold value of "4." The user can modify this value to provide a lower or higher tolerance for index rebuild analysis. This value can be changed at the user level, or it can be changed for a specific sub-set of the database.



```
        Bitmap indexes were disabled
Instance Recommendations for esromlap.world
    Increase sort_area_size to 1048576 bytes for the esromlap.world instance
    Change sort_area_retained_size to 1048576 bytes for the esromlap.world instance
    Increase log_buffer to 32768 bytes for the esromlap.world instance.
    Decrease parallel_max_servers to 0 servers for the esromlap.world instance
Structure Analysis for database ORCL
    OFA Tablespace Guidelines
        Create new tablespace TOOLS
    Tablespace OEM_DATA
    Tablespace ROLLBACK_DATA
    Tablespace SYSTEM
    Tablespace TEMPORARY_DATA
        Alter tablespace TEMPORARY_DATA to specify TEMPORARY option
        Alter tablespace TEMPORARY_DATA to specify NEXT value of 2100 Kbytes
    Tablespace USER_DATA
    User DBSNMP
    User ORDSYS
```

Figure 2: Oracle Tuning Pack Supplies Recommendations Organized by Category. Reports can also be Generated with this Information

Reviewing Recommendations

After Oracle Expert has analyzed the data, the user can review the recommendations. Recommendations are listed in the order of tuning impact importance. The user can examine the details of each recommendation, including a summary of the data analysis and logic used in the recommendation, and the data and evidence that was collected by Oracle Expert. The user then validates the recommendations. Validation means accepting or declining a recommendation.

Recommendations are interdependent, therefore, if a recommendation is declined, the user must rerun the analysis prior to implementing the recommendations. A subsequent analysis will take into account the user's Implementation and Results.

After review and validation, the user can implement Oracle Expert's recommendations. Oracle Expert will produce a set of implementation files, optionally consisting of new instance parameter files and implementation scripts. The user has full control over the implementation process. The implementation files can be invoked when the user is ready, and will automatically implement the changes the user has accepted.

Oracle Expert will also produce a series of reports that document the data and analysis behind the recommendations. These reports provide extensive documentation for the database and tuning process. For the less experienced DBA, they can be a valuable education in the factors that drive database performance.

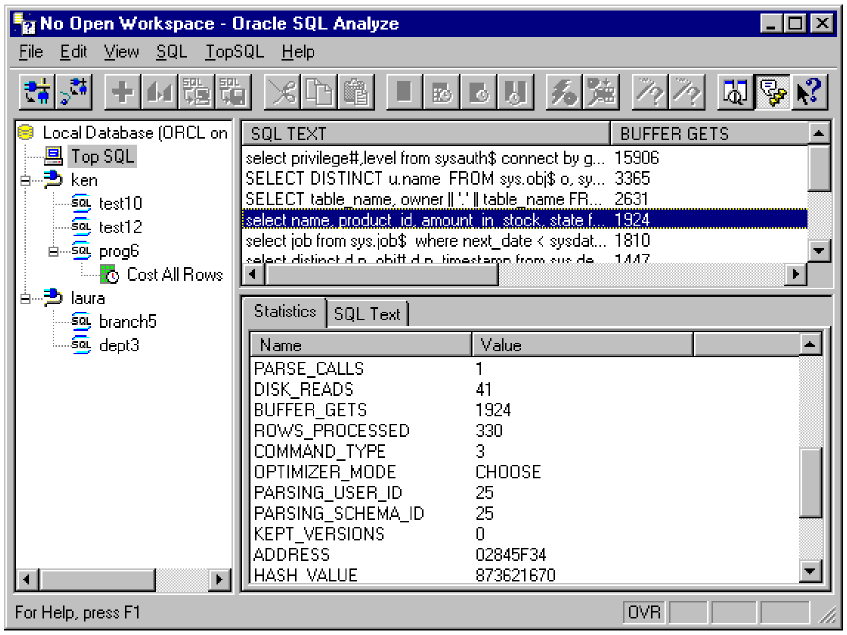
In summary, Oracle Expert provides the DBA with the technology and methodology for performing proactive Oracle database tuning on a database-wide level. However, every Oracle DBA knows that if the system is running a handful of badly designed SQL statements, even the best tuned database will produce unacceptable response times and excessive system load. This makes tuning SQL statements an absolute requirement in order to complete the database performance tuning cycle.

ORACLE SQL ANALYZE

As stated above, inefficient SQL statements are a major contributor to database performance problems. Yet, too often they go unchecked, due to the complexities of identifying and testing optional approaches for obtaining the desired result set. To address this issue, Oracle Tuning Pack provides SQL Analyze, an application for identifying and tuning problematic SQL statements. SQL Analyze can be used to:

- Detect resource intensive SQL statements.
- Examine a SQL statement's execution plan.
- Benchmark and compare various optimizer modes and versions of the statement.
- Generate alternative SQL that can improve application performance.

SQL Analyze is the natural complement to the physical database tuning that Oracle Expert performs.



SQL TEXT	BUFFER GETS
select privilege#,level from sysauth$ connect by g...	15906
SELECT DISTINCT u.name FROM sys.obj$ o, sy...	3365
SELECT table_name, owner || '.' || table_name FR...	2631
select name, product_id, amount_in_stock, state f...	1924
select job from sys.job$ where next_date < sysdat...	1810
select distinct d.p_obj#,d.p_timestamp from sys.de...	1447

Name	Value
PARSE_CALLS	1
DISK_READS	41
BUFFER_GETS	1924
ROWS_PROCESSED	330
COMMAND_TYPE	3
OPTIMIZER_MODE	CHOOSE
PARSING_USER_ID	25
PARSING_SCHEMA_ID	25
KEPT_VERSIONS	0
ADDRESS	02845F34
HASH_VALUE	873621670

Figure 3: Oracle Tuning Pack Identifies Problematic SQL Statements, and Displays Vital Execution Statistics for Each Statement. Oracle Tuning Pack also Creates New Statements on the Fly, as well as Modifies Existing SQL, which can then be Compared Against Other Statements.

Identifying Problem SQL

SQL is highly flexible. Identical results can be obtained by using several different SQL constructs. This flexibility is, of course, the major benefit of SQL. However, it is also the Achilles Heel of database application performance; a SQL statement that produces the desired results set will often be viewed as 'successful,' when in fact the throughput performance may be dismal in production. There is a very good chance that other SQL structures can be developed that will produce the same results set considerably faster. This is not to imply that we should pour over every one of the hundreds of SQL statements in our applications, looking for performance tuning opportunities. Rather, we need a method and mechanism for identifying high impact SQL, so that we can focus our efforts and maximize our results.

SQL Analyze includes a "TopSQL" feature that allows the user to easily identify the top SQL resource users. This feature evaluates the top SQL in use, and presents and sorts it in several ways, for example, by logical reads, disk reads, rows processed, and disk reads per execution.

Top SQL statements can be "dragged and dropped" from the database library cache to the SQL Analyze tuning window, for evaluation and tuning. In addition, SQL statements fromTopSQL can be added to the comprehensive database workload history, where it can be shared with Oracle Expert for integrated SQL and index tuning.

A SQL Tuning Workshop and Methodology

SQL Analyze is integrated with the Oracle® Enterprise Manager Console. All databases known to Enterprise Manager can be tuned from SQL Analyze. The SQL Analyze navigator presents a list of known databases, and allows the user to connect to multiple databases for SQL tuning. SQL statements that are targeted for tuning are imported into SQL Analyze as "SQL nodes." These SQL nodes are presented in a tree hierarchy for easy viewing and navigating. A SQL node contains a single specific syntactical version of a SQL statement. If the user wants to try a different version of the SQL statement, he creates a new SQL node. Users can easily create SQL node variants through a create/like feature. SQL statements that are imported or created from other SQL nodes are presented in a SQL Viewer window. The SQL Viewer allows the user to edit the SQL statement.

SQL "hints" can be added to a new SQL node, or existing hints can be edited using the Hint Wizard feature. The Hint Wizard will identify hints in a statement, and enable the user to present other hints to add to the statement. The Hint Wizard provides a description for a selected hint, and automatically generates a new SQL statement if a hint is added or deleted. Hints are discussed further below.

Generating the Explain Plan

Once you have settled on a SQL statement to start with, the next logical step is to examine the statement's execution path. The Oracle optimizer determines the optimal execution path for the SQL statement. A SQL statement that will retrieve data from multiple tables can use many variations of table join methods, join orders, and access paths to produce the same result set. The optimizer must figure out the optimal path for these operations, based upon a multitude of factors, such as:

- Available indexes
- The order of tables and columns in a SQL statement
- Statistics on the cardinality of objects referenced in the statement
- The use of hints, and so forth

These factors will vary, depending on the type of optimizer the statement uses.

Oracle provides two optimizers, a rule-based optimizer and a cost-based optimizer. The rule-based optimizer uses a fixed set of rules, or conditions, for determining the statement's execution path. The conditions are ranked from low to high in order of precedence, based upon which operation can be performed faster. The cost-based optimizer determines the execution path, based upon a quantifiable cost of the resources required to execute the statement, using various approaches, such as the join order of tables, the type of join used, and the use of available indexes. The cost is computed, based upon statistics for the objects contained in the statement, such as the size of tables and the number of rows for selected columns.

The cost-based optimizer can be run in two modes, Cost First Rows, and Cost All Rows. Cost First Rows provides the optimal cost associated with retrieving the first rows of the results set. Cost All Rows provides the optimal cost associated with retrieving the entire results set. The cost-based optimizer can also be influenced through the use of SQL hints, which can direct the optimizer to use a specific type of join, an index, and so forth.

The SQL statement's execution path can be displayed through an explain plan, which provides a list of the operations involved in the statement's execution. SQL Analyze provides a facility for generating explain plans that can be used to assess how a given SQL statement will perform under different optimizer modes. A SQL statement can run under each optimizer mode by selecting the SQL node, and then selecting the desired optimizer mode function. This will produce an explain plan for the statement, and the execution cost of the statement if the cost-based optimizer was used. It will also add a "child node" to the SQL node in the tree, one child node per selected optimizer mode.

Stepping through the Plan

After generating one or more explain plans for a SQL statement, a user can use SQL Analyze to examine the plan, the order of operations, and relevant statistics for objects used in the plan. Explain plans can be confusing to interpret. The hierarchical order of operations listed in the plan is not the order of execution.

For example, the following SQL statement:

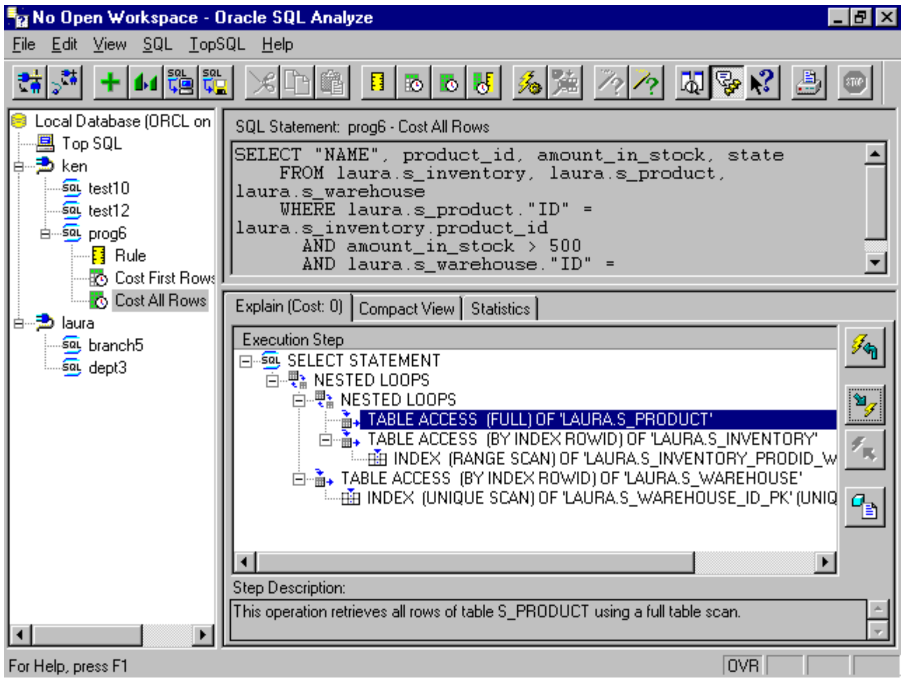
```
SELECT "name", product_id, amount_in_stock, state
FROM inventory, product, warehouse
WHERE product.id = inventory.product_id
AND amount_in_stock > 500
AND warehouse.id = inventory.warehouse_id;
```

is represented by the following explain plan, using the rule-based optimizer:

```
SELECT STATEMENT
   NESTED LOOPS
      NESTED LOOPS
         TABLE ACCESS (BY ROWID) OF 'INVENTORY'
            INDEX (RANGE SCAN) OF 'AMOUNT_IN_STOCK_PK'(NON-
UNIQUE)
         TABLE ACCESS (BY ROWID) OF 'WAREHOUSE'
            INDEX (UNIQUE SCAN) OF 'WAREHOUSE_ID_PK'(UNIQUE)
      TABLE ACCESS (BY ROWID) OF 'PRODUCT'
            INDEX (UNIQUE SCAN) OF 'PRODUCT_ID_PK'(UNIQUE)
```

In this case, the execution path uses INVENTORY as the driving table, and the first operation is the range scan of the AMOUNT_IN_STOCK_PK index for that table. After retrieving multiple ROWIDs from the index, it performs the second operation, by using these values to retrieve rows from the INVENTORY table. It then moves on to the third operation, which is to use the WAREHOUSE_ID_PK index to retrieve the ROWIDs. This allows it to then perform the fourth operation, access to the WAREHOUSE table by ROWID. It then performs the fifth operation, a NESTED LOOPS join of the sets returned from the two tables. Then on to the sixth and seventh operations, involving the PRODUCT table. The final operation is a NESTED LOOPS join of the set from the Product table, and the set resulting from the join on the INVENTORY and WAREHOUSE tables.

As you can see, the operations are performed in an order that is inverse to their position in the plan. SQL Analyze makes the execution order clear, by providing a graphical step-by-step ordering and explanation of the statement's operations. The user can walk through the plan, and immediately understand how the statement will execute and what step each operation is performing.



SQL Statement: prog6 - Cost All Rows

```
SELECT "NAME", product_id, amount_in_stock, state
    FROM laura.s_inventory, laura.s_product,
laura.s_warehouse
    WHERE laura.s_product."ID" =
laura.s_inventory.product_id
      AND amount_in_stock > 500
      AND laura.s_warehouse."ID" =
```

Execution Step

```
SELECT STATEMENT
  NESTED LOOPS
    NESTED LOOPS
      TABLE ACCESS (FULL) OF 'LAURA.S_PRODUCT'
      TABLE ACCESS (BY INDEX ROWID) OF 'LAURA.S_INVENTORY'
        INDEX (RANGE SCAN) OF 'LAURA.S_INVENTORY_PRODID_W
    TABLE ACCESS (BY INDEX ROWID) OF 'LAURA.S_WAREHOUSE'
      INDEX (UNIQUE SCAN) OF 'LAURA.S_WAREHOUSE_ID_PK' (UNIQ
```

Step Description:

This operation retrieves all rows of table S_PRODUCT using a full table scan.

Figure 4: Explain Plans can be Complex to Interpret. Oracle Tuning Pack Walks You through the Explain Plan, Providing a Description of each Step

Considering Alternative SQL Methods

After reviewing the explain plan, the user may want to check the statement for potential performance problems, or alternative methods that may improve performance. SQL Analyze provides a SQL Tuning Wizard that can be used automatically for tuning opportunities

Checking the "Rules of Thumb"

SQL Analyze can be used to check a statement for any violations of basic SQL "rules of thumb." This consists of checking the statement for syntax methods that are known to have a negative effect on performance. SQL Analyze contains a conservative set of rules that will evaluate the SQL statement, and provide an alternative statement, if possible.

A simple example of a "rule of thumb" involves the inadvertent disabling of an index for a SQL statement, by performing a function on the indexed column, as in the following example:

```
    SELECT division_name, company_name
```

```
      FROM division, (SELECT sum(profits) sumprofits FROM division)
      total
      WHERE (division.profits / total.sumprofits) * 100 > :pct
```

In this case, division.profits is an indexed column. However, the execution plan for this statement does not use the index, choosing a full table scan of the division table instead:

```
      SELECT STATEMENT
        NESTED LOOPS
          VIEW
            SORT (AGGREGATE)
              INDEX (FULL SCAN) OF 'IDX_DIVISION_PROFITS'
          TABLE ACCESS (FULL) OF 'DIVISION'
```

SQL Analyze will recognize this 'rule of thumb' violation, and propose an alternative SQL statement that allows the index to be used, as follows:

```
      SELECT division_name, company_name
      FROM division, (SELECT sum(profits) sumprofits FROM division)
      total
      WHERE division.profits > (:pct / 100) * total.sumprofits
```

This removes the operation on the indexed column, and retrieves the desired results set much more quickly with the following execution path:

```
      SELECT STATEMENT
        NESTED LOOPS
          VIEW
            SORT (AGGREGATE)
              INDEX (FULL SCAN) OF 'IDX_DIVISION_PROFITS'
          TABLE ACCESS (BY INDEX ROWID) OF 'DIVISION'
            INDEX (RANGE SCAN) OF 'IDX_DIVISION_PROFITS'
```

SQL Analyze will provide a growing body of such 'rules of thumb' that can be used to check statements for basic violations that might otherwise go unchecked.

Evaluating SQL Join Methods and Join Orders

Most SQL queries involve selecting data from multiple tables. In these operations, data from several tables is "joined" from multiple tables, in order to produce the desired results set. The type of join method used, and the order of table joining, is driven by several factors, such as the presence of indexes, and the cardinality of columns involved in the selection process. If indexes are present on one or more of the tables involved in the query, then specific rows can be selected from each table. These sets of rows can then be compared and joined. In this case, the join operation will be a "nested loop," where the data obtained from the "driving" table will be used to select rows from the "driven" table. This is the case in the explain plan example above.

Given that the data obtained from one table is used as criteria for selecting data from another table, the order of table access operations is the major determinant of performance for a multi-table query. Determining the proper join order becomes more complex as the number of joined tables increases. The potential join orders increases by n! for n tables, where a join involving 6 tables has 720 possible join orders.

Both the cost and rule based optimizers utilize specific rules, data, and strategies to determine the join order. An informed developer can influence the behavior of both optimizers. Using SQL Hints, the developer can assist the cost-based optimizer and control the execution plan. This can be valuable for specific queries, where the developer is aware of details that may not be available to the optimizer, such as the performance goal of a particular query (throughput vs. response time), outdated or non-existing statistics for certain objects, or bind-variables that may effect a filter condition.

Using hints to control the join method and order for specific queries can be complex and risky. To assist the developer with this, SQL Analyze provides an automated methodology that can be used to evaluate alternative join strategies, where appropriate. In the same manner as Oracle's internal cost-based optimizer, SQL Analyze will estimate the cost for executing a statement in different ways. SQL Analyze will estimate the object statistics, and collect typical values for bind variables that can be used to fully evaluate the optimal join order. If an alternative join order can be used, it will rewrite the statement, with the necessary hints or reordering of objects. The Tuning Wizard will also provide a before and after comparison of the statement's performance results.

Reviewing Object Statistics

The performance of the SQL statement is also effected by the space usage of the objects accessed. Factors, such as the existence of chained rows in a table, can increase the number of I/Os required to retrieve the set. SQL Analyze provides a quick overview of pertinent space usage statistics for objects. Table statistics include important metrics, such as the number of extents, used and empty blocks, chained rows and average free space per block, and a list of the distinct values count for each of the columns. Index statistics include metrics that can be used to gauge how effectively space is being used within the index segment, such as the tree depth, and the average leaf blocks per distinct key.

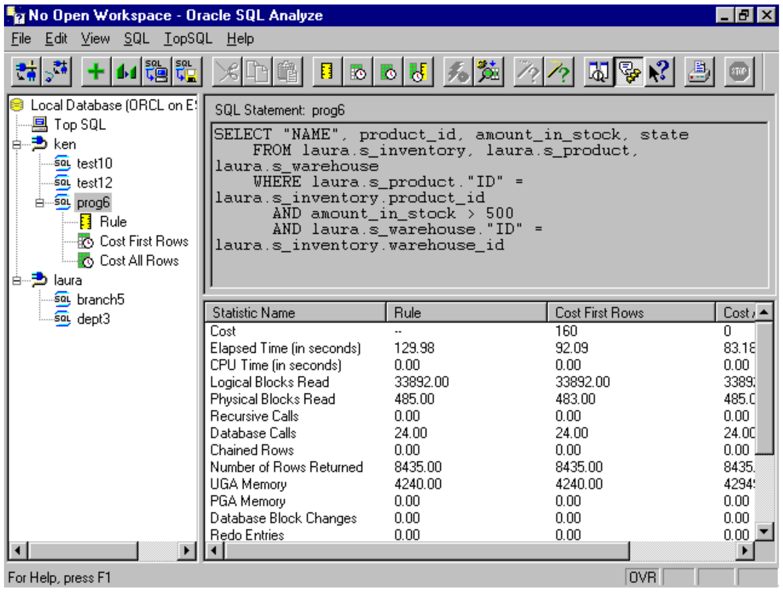
Measuring and Comparing SQL Performance

Having reviewed and interpreted the explain plan, the user can move on to the next logical step, testing the performance of the statement under various optimizer modes. SQL Analyze allows the user to measure performance, by executing the statement one or more times. An easy measure of relative performance is the elapsed time that it takes to perform the query. CPU time can also be used, as well as the number of logical and physical reads. Another important metric is the occurrence of sorts performed on disk, where the query has to swap out part of the sort to a temporary segment, resulting in expensive I/Os. SQL Analyze provides these and other key metrics for measuring a statement's performance. Execution statistics are presented for the last test execution, and averaged over all test executions for each optimizer mode tested for a given SQL node.

In addition, all of these key SQL metrics can be displayed for comparison, by simply highlighting the desired SQL object in the navigator. A table of performance metrics for each optimizer mode is provided for easy comparison.

SQL Analyze also takes advantage of the new query progress monitoring capability in Oracle8*i*, enabling the user to track the progress of a SQL statement being executed from SQL Analyze.

SQL Analyze provides a "split view" screen option that allows a side-by-side comparison of two different SQL statements, or optimizer modes for the same statement. A user can easily compare explain plans and performance statistics throughout the tuning cycle. This allows the user to perform, and quickly gauge the relative performance of several different approaches for obtaining the same result set. The user can also view the result of the query at any point after execution, to ensure that the statement is producing the desired data.



```
SELECT "NAME", product_id, amount_in_stock, state
    FROM laura.s_inventory, laura.s_product,
laura.s_warehouse
    WHERE laura.s_product."ID" =
laura.s_inventory.product_id
      AND amount_in_stock > 500
      AND laura.s_warehouse."ID" =
laura.s_inventory.warehouse_id
```

Statistic Name	Rule	Cost First Rows	Cost
Cost	--	160	0
Elapsed Time (in seconds)	129.98	92.09	83.18
CPU Time (in seconds)	0.00	0.00	0.00
Logical Blocks Read	33892.00	33892.00	3389
Physical Blocks Read	485.00	483.00	485.0
Recursive Calls	0.00	0.00	0.00
Database Calls	24.00	24.00	24.0
Chained Rows	0.00	0.00	0.00
Number of Rows Returned	8435.00	8435.00	8435.
UGA Memory	4240.00	4240.00	4294
PGA Memory	0.00	0.00	0.00
Database Block Changes	0.00	0.00	0.00
Redo Entries	0.00	0.00	0.00

Figure 5: Vital Execution Statistics are Displayed for a Complete Execution Picture. When Tuning SQL, Comparing Execution Mode Results in this Manner is Quite Useful for Determining a Statement's Ideal Optimizer Mode

Integrated Index Tuning

Adding an index to improve the performance of a SQL statement should be done in the context of the entire database workload for the target table. SQL Analyze provides this capability through its use of the comprehensive database workload and Index Tuning Wizard, referenced earlier in this paper. SQL Statements targeted for tuning from the TopSQL view can be added to the workload, and the Index Tuning Wizard can be focused on the objects used in the SQL statements being tuned. This provides a complete SQL tuning effort, driven through SQL Analyze.

In summary, SQL Analyze provides a robust workbench for identifying and tuning high impact SQL statements. However, getting the best performance from the database's application SQL requires more than tuning individual SQL statements. It also requires identifying and eliminating database storage problems, and ensuring that Oracle's advanced Cost-based Optimizer has up-to-date object statistics. To meet this need, we now turn to the third application in Oracle Tuning Pack, known as Oracle Tablespace Manager.

ORACLE TABLESPACE MANAGER

Effective management of database space usage is required to ensure high database performance. Storage space required by various database structures, such as tables and indexes, will change as the database changes and grows. An Oracle database will grow dynamically, meaning that additional units of data storage will be added as space requirements increase, ensuring continuous operation. Likewise, as data is changed or deleted from the database, space may be reclaimed for use by other database objects. Careful planning and use of storage settings will make this dynamic process as transparent as possible to administrators, without requiring frequent monitoring and adjustment.

When Oracle space management is required, it generally involves reorganizing tables and indexes to increase performance or reclaim unused space. For example, a database serving a transaction or batch processing application will undergo frequent updates, insertions, and deletions. Indexes affected by these transactions may experience “index stagnation,” a situation where the amount of unused space in an index segment can grow, resulting in less efficient index scans. Indexes can be rebuilt to minimize unused space and improve performance.

Another example of how database reorganization may be necessary for improved performance is when segments have become fragmented, due to poorly planned storage parameters. If extent size is allowed to grow with each new extent for the segment (where the PCTINCREASE parameter for the segment is > 0), then new extent allocations will grow geometrically, and the extents will vary in size. Sizing a segment’s extents uniformly, and as a multiple of the number of blocks that can be read during each multi-block read (as determined by the DB_FILE_MULTIBLOCK_READ_COUNT parameter), will neutralize any performance impact of having multiple extents.

Additionally, properly planning space requirements and settings, such as PCTFREE and PCTUSED, can reduce the incidence and performance impact of disorganized segments and unused space. For example, if the application is performing frequent updates that increase row size, then PCTFREE should be increased to handle such growth. If these settings are not planned properly, then row chaining and excessive unused space can occur.

A DBA that suspects database performance problems due to any of the above space management issues can use Oracle Tablespace Manager to further investigate and correct space problems. Oracle Tablespace Manager has four major features:

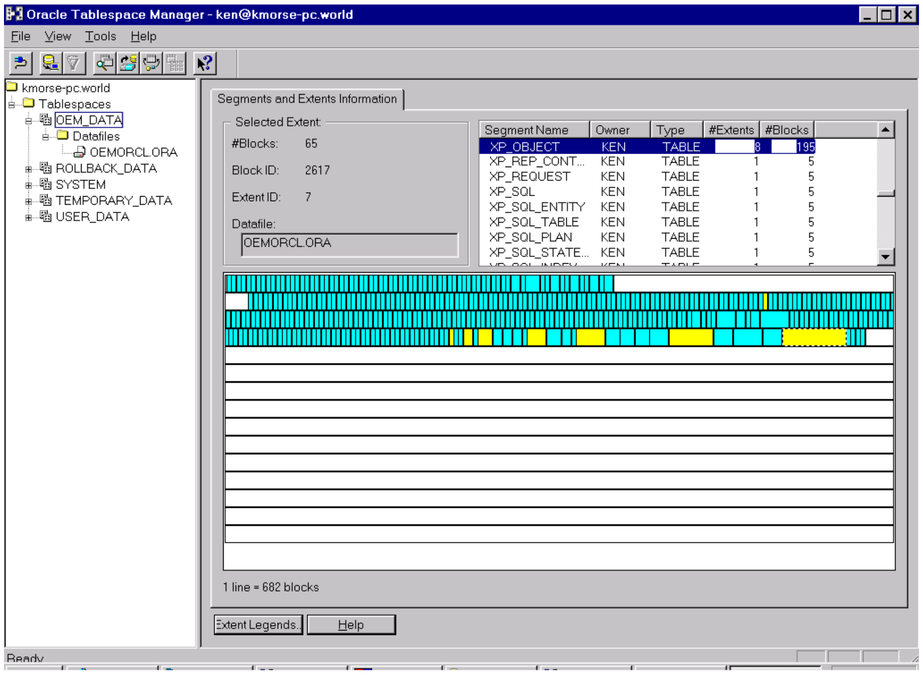
- Tablespace Viewer
- Proactive Problem Detection Wizard
- Tablespace Reorganization Wizard
- Tablespace Analyzer Wizard

Graphical Tablespace Viewer

The Tablespace Viewer provides the administrator with a complete picture of the characteristics of all tablespaces associated with a particular Oracle instance, including:

- Tablespace Datafiles and Segments
- Total Data Blocks
- Free Data Blocks
- Percentage of Free Blocks Available in the Tablespace’s Current Storage Allocation

The DBA has the option of displaying either all segments for a tablespace, or all segments for a datafile. The Tablespace Viewer presents key performance metrics for the selected segment, such as average free space per blocks, chained rows, and the last date that the object was analyzed.



Selected Extent:
#Blocks: 65
Block ID: 2617
Extent ID: 7
Datafile: OEMORCL.ORA

Segment Name	Owner	Type	#Extents	#Blocks
XP_OBJECT	KEN	TABLE	8	195
XP_REP_CONT...	KEN	TABLE	1	5
XP_REQUEST	KEN	TABLE	1	5
XP_SQL	KEN	TABLE	1	5
XP_SQL_ENTITY	KEN	TABLE	1	5
XP_SQL_TABLE	KEN	TABLE	1	5
XP_SQL_PLAN	KEN	TABLE	1	5
XP_SQL_STATE...	KEN	TABLE	1	5

1 line = 682 blocks

Figure 6: Oracle Tablespace Manager's "Map View" Shows the Physical Location of Objects and Available Free Space, Including any Fragmentation Problem.

The Tablespace Viewer also provides a map of the organization of a tablespace's segments. This map graphically displays the sequential allocation of space for segment extents within a selectedtablespace or datafile. For example, a table segment may consist of three extents, all of which are physically separated by other segment extents. The map will highlight the locations of the three extents within the tablespace or datafile. It will also show the amount of free space available for each segment. In this way, the Tablespace Viewer map provides the DBA with an easy birds-eye view of tablespace fragmentation.

Proactive Problem Detection

Space management problems can stem from many sources. Detecting potential problems before they start to influence performance can save unnecessary downtime and expensive delays. TheTablespace Manager provides a mechanism to automatically analyzetablespace objects, and identify potential space management problems. The Proactive Problem Detection Wizard will evaluate the target objects for the following potential problems:

- Migrated or Continued Rows
- Stagnated Indexes
- Non-Linear Extent Growth

- Inefficient Extent Size Settings for Linearly Scanned Segments
- Objects Approaching Maximum Extent Settings

The wizard will detect and report on problems, and will allow the user to initiate space reorganization if required.

Tablespace Reorganization

When space reorganization is necessary, the DBA can use the Oracle Tablespace Manager Reorganization Wizard to automatically correct the problem. The DBA can select a single segment, multiple segments, or the entire tablespace for defragmentation, including tables, indexes, and cluster segments.

The reorganization process uses the Oracle export/import functions on the target table, and ensures that all rows, indexes, constraints, and grants will remain intact. It also checks to ensure that sufficient space exists to perform the reorganization tasks. Before the table export occurs, the DBA is presented with dialog boxes for modifying the segment's storage parameters, in order to avoid future space problems. The new parameters will then be used in the re-creation of the re-organized objects. Segments can also be moved between tablespaces.

Automated ANALYZE

If an object has not been ANALYZED for cost-based optimizer statistics, or if ANALYZE statistics are out-dated, then the cost-based optimizer may be hampered, or unavailable, and SQL performance tuning will be difficult. In addition, proper object metrics will not be available for use in troubleshooting performance issues related to space management problems.

The Tablespace Manager provides a simple solution for keeping object statistics up-to-date, the Tablespace Analyzer. This feature provides automated ANALYZE of one or more objects for a selected schema, or for the entire database. ANALYZE can be run immediately, or scheduled to run at a later time. ANALYZE can also be scheduled to run on a periodic basis. The Tablespace Analyzer is built on the Oracle Enterprise Manager job system, and uses the Oracle Intelligent Agent to control job scheduling, allowing lights-out re-ANALYZE of specific database objects.

ORACLE ENTERPRISE MANAGER PRODUCT FAMILY

Also part of the Oracle Enterprise Manager product family are the Oracle® Diagnostics Pack, Oracle® Change Management Pack, Oracle® Management Pack for Oracle8*i*, Oracle® Management Pack for Oracle Applications and Oracle® Management Pack for SAP/R3. All are fully integrated into the Oracle Enterprise Manager Console and framework, and provide a unified system management framework for end-to-end management of your Oracle environment.

Built upon open Internet standards, such as Java, CORBA, and IIOP, these products provide the first management framework designed to support Internet computing. All applications can be accessed from anywhere a browser is available. A reliable and scalable multi-administrator repository leverages your administrative staff, by providing cooperative management. Using the Oracle Enterprise Manager product family, administrators and IT managers can insure higher-productivity, deliver better services, and reduce the overall cost of their information systems.

CONCLUSION

Oracle Tuning Pack provides the DBA with a useful set of sophisticated tools for database performance tuning, covering key performance tuning areas, such as:

- Data Access Paths.
- Proper Database Structures.
- Optimized Instance Settings and Application SQL.
- Tuned Database Storage.

Oracle Tuning Pack is the natural complement to Oracle Diagnostics Pack, which provides comprehensive database performance monitoring and planning. By using the collective set of Diagnostic and Tuning tools provided in these two management packs, the Oracle DBA can make significant improvements in productivity and database performance.