Performance Enhancement using SQL Statement Tuning Approach

Hitesh Kumar SHARMA[1], Mr. S.C. NELSON[2]

University of Petroleum & Energy Studies, Dehardun, Uttarakhand, India
hksharma@ddn.upes.ac.in, cnelson@ddn.upes.ac.in

Tuning your SQL statements may be one of the most important tasks you can do to improve the performance of your Oracle system. By tuning your SQL statements to be as efficient as possible, you use your system to its full potential. Some of the things you can do to improve the efficiency of your SQL statements may involve as little effort as rewriting the SQL to take advantage of some property of your database or perhaps even changing the structure of the database itself.

1 Introduction

Tuning SQL really falls into two separate categories:

*Tuning an existing application.*This approach involves less flexibility in terms of changing the structure of the application and the database, but may provide performance improvements anyway.

*Designing a new application.*With a new application, you have the flexibility to design the application and perhaps even the database itself. With this approach, you can take advantage of indexes, clustering, and hashing.

This paper emphasis at both of these categories. The amount of changes you can do and the flexibility you have in changing the design of the application and database depends on your particular situation. The more flexibility you have in designing the database along with the application, the better your overall results will be. Tuning an Existing Application

Tuning an existing application can be easier in some respects and harder in others. It is easier in terms of determining the data access patterns and the specific problem areas because the application is already running and can be profiled very easily with SQL Trace and EXPLAIN PLAN. However, fixing the problems can be a challenge because you may not have the flexibility to do so. With an existing application, you may or may not have the flexibility to fix the problem. For example, you may have system performance problems, but the system is still stable enough and performs well enough for the users to get their work done. It is hard to justify the downtime involved in reconfiguring the system when it is still in a somewhat functional state. In this type of situation, it is important to plan far in advance and take advantage of scheduled downtime to implement system enhancements and fix any current problems and anticipated additional growth in user activity. This may be done by changing Oracle or OS parameters or by adding more disk drives, memory, and so on. If you can afford to build in an additional 20 to 30 percent growth, you may save yourself some reconfiguration time down the road.

Over the years, I have found that you take advantage of additional capacity much faster than most people anticipate and plan for. I remember the old days of the PC industry, when it was inconceivable that anyone would ever need more than 64 megabytes of memory or even think of needing a gigabyte of disk space on a desktop PC. Today, high-end PC servers support more than a gigabyte of RAM and are getting close to supporting a terabyte of disk space.

2. Problem Analysis

To tackle the problem of tuning an existing Oracle application, I recommend using

some kind of methodology. Here are the steps that will hopefully lead you to the problem's resolution:

1. **Analyse the system.** Decide if there is a problem; if there is one, what is it?
2. **Determine the problem.** What do you think is causing the problem? Why?
3. **Determine a solution and set goals.** Decide what you want to try and what youthink will result from your changes. What is it you want to accomplish? Do you want more throughput? Faster response times? What?
4. **Test the solution.** Try it. See what happens.
5. **Analyse the results.** Did the solution meet the goals? If not, you may want to return to step 1, 2, or 3. By following a plan of this sort, you will find it much easier to resolve the problem—or determineif there is a problem. It is possible to spend a lot of time trying to solve a performance problem that may not even exist.

By looking at the performance of the system as it is and carefully examining its characteristics, you may be able to determine whether any action is necessary to fix the problem and how much effort is involved. We like to classify performance problems into oneof three categories:

- **It's broken.** Performance is severely handicapped because of a configuration error or an incorrect parameter in the OS or RDBMS. Problems that fall into this category cause a performance swing of 50 percent or more. Problems that fall into this category are usually oversights during the system build, such as incorrectly building an index or forgetting to enable asynchronous I/O. This category of problem may indicate a more serious problem such as insufficient disk drives or memory.
- **It's not optimized.** Performance is slightly degraded because of a small miscalculation in parameters or because system capacity is slightly exceeded. These types of problems are usually easily solved by fine-tuning the configurations.
- **Not a problem.** Don't forget that sometimes there isn't a problem; you are just at the capacity of the system. This "nonproblem" is easily solved by upgrading or adding more capacity. Not all problems can be solved with tuning.

In the first case, you may have to perform some drastic fix that probably involves rebuilding the database in some fashion. The solution may be as drastic as having to rebuild from scratch to add more disk drives; it may be as simple as adding an additional index.

In the second case, you may be able to tune the system with an OS or database configuration parameter. This is usually very easy to do and does not involve much risk. However, this solution does not usually result in a huge increase in performance.

The final case may involve adding an additional CPU (if you have an SMP or MPP machine) or upgrading to new hardware. Perhaps you will find that there really isn't a problem after all and that everyone is happy with the performance of the system. One thing to remember: You rarely see the end users if everything is going fine. It's only when there are performance problems that you hear from them.

3. Tuning the Application

When analysing the SQL statement, you should look for two things:

- **The SQL statement.** What does it do?
- **The effect of the SQL statement.** What is it doing it to? How does it fit into the big picture?

By looking at the SQL statement from these different angles, you may find a problem that you wouldn't find by just looking at it from one viewpoint.

For example, consider an application that does not used cached sequences to generate a primary key value. By itself, there is nothing wrong with this approach and the application is probably very efficient. But add a thousand users executing the same application and the problem is quite apparent: You have contention getting the value for the primary key value.

The SQL Statement

The best way to go about tuning the SQL statements of an existing application is to follow these few steps:

1. Familiarize yourself with the application. You should be familiar not only with the specific SQL statements but with the purpose of the application and what it does.
2. Use the SQL Trace facility to analyse what the particular SQL statements are doing, what features of the RDBMS are being used, and how well those features are being used.
3. Use EXPLAIN PLAN within SQL Trace to analyse how the optimizer is executing those SQL statements.

Now take a look at some specifics of these steps.

Familiarize Yourself with the Application

Look not only at the SQL statements themselves but at the effect of those statements. Make a chart of the different SQL statements and determine the number of accesses each SQL statement makes to each table in the database. This chart can give you an effective visual idea of which tables are being accessed most frequently. Consider the example shown in Figure 1. This chart is a good quick reference for which SQL statements are affecting which tables. You can take this a step further and split the chart into different types of statements such as SELECTs, INSERTs, UPDATEs, DELETEs, and so on. Depending on your system and whether your application is shrink-wrapped or developed in-house, this may be or may not be practical.

Use SQL Trace to Analyse the SQL Statements

By running SQL Trace on the SQL statements, you can gather much valuable information about the specific operation of each of the SQL statements. SQL Trace provides such valuable information as the following:

- Parse, execute, and fetch counts
- CPU and elapsed times
- Physical and logical reads
- Number of rows processed
- Library cache misses

You can use this information to determine which SQL statements are efficient and which ones are not. Look for the following indications of inefficient statements:

SQL Trace Output Comments

CPU and elapsed time. If the CPU or elapsed times are very high, this SQL statement is a good candidate for tuning. It doesn't make much sense to spend time tuning statements that don't use many resources. Executes Focus on SQL statements that are frequently executed. Don't spend time on SQL statements that are infrequently used.

SQL Statement # \ Table	Acct 1	Acct 2	Finance 1	Finance 2	History 1
1	R W	R W	R W	R W	W
2	R W	W		R W	W
3	R	R W		W	W
4	R	W	R	R	W
5	W	W		R	W

R = Read
W = Write

Fig. 1. An example of an SQL statement-analysis chart.

Rows Processed. An SQL statement with a high number of rows processed may not be using an index effectively.
Library Cache. A large number of library cache misses may indicate a need to tune the shared pool or to change the SQL statement to take advantage of the shared SQL area. These clues may point you in the direction of the SQL statements that need to be tuned. You may have to alter these SQL statements to improve their efficiency. By using EXPLAIN PLAN, you may find addition areas that can be improved.

Use EXPLAIN PLAN to Analyse Statement Execution
By running EXPLAIN PLAN as part of the SQL Trace report, you can get a better idea of how the SQL statement is actually being executed by Oracle. This information (and the information supplied by SQL Trace) helps you judge the efficiency of the SQL statement. Here is a list of some of the things to look for:
Are the table's indexes being used when they should be? If not, the statement may not be supplying the correct parameters in the WHERE clause.

- Are indexes being used when they should not be? In cases when you are selecting too much data, you may want to use the FULL hint to bypass the index.
- What is the cost of the SQL statement? This value is given in the *position* column of the first row of the table returned by EXPLAIN PLAN.
- What is the amount of overhead incurred from SELECT or UPDATE operations?
- Is the statement being parallelized? You may have to provide a hint to effectively take advantage of the Parallel Query option.

You should ask these questions and your own specific questions as you review the EXPLAIN PLAN output. By knowing what your application is supposed to do, you may find important information about the efficiency of your statements by looking at this information.

The Effect of the SQL Statement
In addition to looking at the SQL statement itself, you should also look at the effect of the SQL statements. In many cases, some detail that is unimportant by itself can become a problem when the application and SQL statements are run by hundreds or thousands of users at the same time. The effect of this can be a bottleneck on a specific table or even a specific row.
Here is a list of some things to look for when analysing the effect of the SQL statements:

- Is the SQL statement updating a specific row? If you update a specific row as a counter, it may cause a bottleneck.
- Where is the majority of the table activity? Is a specific table being heavily accessed? This could indicate an I/O bottleneck.
- Is there significant INSERT activity? Is it all to one table? This may indicate a contention problem on a certain table.
- How much activity is there? Can the system handle it? You may find that the SQL statements overload your particular system.

These are just a few of the things to consider when you are looking at the effects of the application on the system. I have seen cases in which an application, fully tested in the lab, moves into production and fails because it was tested with only one or two users. It is important to take into account the effect of hundreds or thousands of users simultaneously accessing the application.

Review of How to Tune an Existing Application
Tuning an existing application can be quite a challenge. Determining whether the system is in need of optimization and figuring out how to do it is not always

easy. The task may be easier if you take a methodical approach like this one:

1. **Analyse the situation.** It may be that your system is not in need of adjustment. I do not recommend making any changes to a stable system unless you have to.
2. **Familiarize yourself with the application.** Look at the SQL statements as well as the overall application. Understand the purpose of the application.
3. **Make an analysis chart.** Look at the table accesses being generated by the application.
4. **Run SQL Trace with** EXPLAIN PLAN**.** See what the SQL statements are really doing. Choose the statements to focus on based on how often they are used and how many resources they use.
5. **Understand how these SQL statements affect the server system.** Look at Oracle and the OS. Determine which disks may be overused and where contention could occur when many users run the application. With an existing application, you may or may not have the flexibility to fix the problem. I do not recommend making any changes to an existing application or a functioning system unless some specific performance problems are affecting users or limiting the capacity of the system. Of course, if you have the flexibility to make changes and there is a need, any of the design and application changes described in the following section, "Designing a New Application," also apply to an existing application.

Packages, Procedures, and Functions

Another way to improve performance of your SQL statements is by using packages, procedures, and functions. *Packages* can help improve performance by storing together procedures and functions that are often used together. By storing these elements together, you can reduce the I/O required to bring them into memory from disk. Because these elements are often used together, they can also be loaded from disk together. By using *stored procedures*, you benefit in several ways. Stored procedures allow you to reduce the amount of data sent across the network. The stored procedure requires fewer instructions to be sent to the server; in many cases, less data must be sent back to the client from the server.

A second benefit of a stored procedure is the increased chance that the SQL statement can be used by other processes. Because the SQL statement is defined and used by many processes, chances are good that the SQL statement will already be parsed in the shared SQL area and available to other users.

Optimization Approaches

Oracle offers several options for optimization techniques. Among these are a cost-based approach and a rule-based approach. The approach you take depends both on your application and your data. In most cases, the cost-based approach is recommended because it determines an execution plan that is as good or better than the rule-based approach. The following sections look at the optimization approaches available from Oracle and when each approach is appropriate. Remember that you can use hints to specifically tell Oracle how you want the SQL statement to be executed. There are several ways you can indicate your preference, as described later in this chapter.

Rule-Based Approach

The rule-based approach to Oracle optimization is straightforward and consistent. In the rule-based approach, the execution plan is derived by examining the available paths and ranking them against a list of predetermined values for these paths (see Figure 2).

Rank	Access Path
1	Single row by ROWID
2	Single row by cluster join
3	Single row by hash cluster key with unique or primary key
4	Single row by unique or primary key
5	Cluster join
6	Hash cluster key
7	Indexed cluster key
8	Composite key
9	Single-column indexes
10	Bounded range search on indexed columns
11	Unbounded range search on indexed columns
12	Sort-merge join
13	MAX or MIN of indexed column
14	ORDER BY on indexed columns
15	Full-table scan

Fig. 2. The rule-based optimization rankings.

With the rule-based optimization approach, the optimizer determines the ways to execute the SQL statement. If there is more than one way to execute the SQL statement, the table in Figure is used to choose the approach with the lowest ranking.

Cost-Based Approach

The cost-based approach to optimization uses existing knowledge of the database to choose the most efficient execution plan. During the normal operation of the RDBMS, statistics are gathered on the data distribution and storage characteristics. The optimizer uses this information to determine the most optimal execution plan. This optimization approach takes three steps:

1. The optimizer generates a set of possible execution plans, just as it does with the rule-based optimization approach.
2. The cost of each plan is determined based on statistics gathered about the database. This cost is based on CPU time and the I/O and memory necessary to execute the plan.
3. The optimizer compares the costs and chooses the execution plan with the lowest cost. The cost-based approach is usually preferred. In some cases, the rule-based approach may be more appropriate.

Discrete Transactions

Discrete transactions are an optional way of processing SQL statements. They may help performance in certain situations. Discrete transactions can be used only in a limited set of circumstances, but if you can take advantage of them, they can help performance. Discrete transactions work with small, non-distributed transactions and improve performance by deferring the writing of redo information until the transaction has been committed. Discrete transactions cannot and should not be used for all transactions.

How Do Discrete Transactions Work?

With discrete transactions, all changes made to data are deferred until the transaction has been committed. Even though redo information is saved, it is not written to the redo log until the transaction has been committed. Until the commit, the redo information is stored in another area of memory. When the transaction is committed, the redo information is written to the redo log and the changes are made to the data block. This arrangement causes the rollback segments to be bypassed. Because the changes are deferred until the commit, the undo information does not have to be saved. With normal transactions, the undo information must be saved in the rollback segments because the data blocks have already been changed. With discrete transactions, because the data blocks have not been changed until the commit, it is unnecessary to save that information. Discrete transactions should not be used on data accessed with long-running queries because those queries will not be able to access any undo information.

When Should Discrete Transactions Be Used?

Discrete transactions should be used only under certain circumstances and with certain transaction types. Here are some guidelines for when to use discrete transactions:

- Discrete transactions should be used to modify only a few data blocks. The amount of data and the amount of time it takes to perform the transactions should be as small as possible.
- Discrete transactions should not be used on data that might be accessed by long-running queries. Those queries will not be able to access any undo information.
- Discrete transactions should not be used on any rows containing LONG data. Because of these restrictions, discrete transactions can be used only with a small subset of transactions. These transactions should be small and quick. If you can take advantage of discrete transactions, you should see a fairly good performance benefit.

Discrete transactions work with small, non-distributed transactions and can improve performance by deferring the writing of redo information until the transaction has been committed. Discrete transactions cannot and should not be used for all transactions. Before making modifications to your application design, determine whether discrete transactions can benefit your application. Determine whether your transactions fit the profile that can benefit from discrete transactions; also determine whether these changes will be significant enough to warrant the use of discrete transactions. If your system is under a heavy load of mostly small transactions, it may be worth investigating discrete transactions. Try implementing them and see what kind of benefit they return.

Tuning Considerations

The data warehouse is tuned to allow several large processes to run at maximum throughput. There is usually no concern for response times. We may have to tune both Oracle and the server operating system. The following sections look first at Oracle and then at the server operating system.

Server OS Tuning

We may have to tune the server OS to provide for a large number of processes (if we are using the Parallel Query option) and optimal I/O performance. Some of the things we may have to tune in the server OS are listed here; remember that some OSes may not require any tuning in these areas:

- **Memory.** Tune the system to reduce unnecessary memory usage so that Oracle can use as much of the system's memory as possible for the SGA and server processes. We may also need significant amounts of memory for sorts.
- **Memory enhancements.** Take advantage of 4M pages and ISM, if they are available. Both features can improve Oracle performance in a data warehouse environment.
- **I/O.** If necessary, tune I/O to allow for optimal performance and use of AIO.
- **Scheduler.** If possible, turn off pre-emptive scheduling and load balancing. In a data warehousing system, allowing a process to run to completion (that is, so that it is not pre-empted) is beneficial.
- **Cache affinity.** We may see some benefits from cache affinity in a data warehousing system because the processes tend to run somewhat longer. The server operating system is mainly a host on which Oracle does its job. Any work done by the operating system is essentially overhead for Oracle. By optimizing code paths and reducing OS overhead, we can enhance Oracle performance.

Hardware Enhancements

For a data warehouse, there are several hardware enhancements that can improve performance. These hardware enhancements can be beneficial in the area of CPU, I/O, and network, as described in the following sections.

CPU Enhancements

Enhancing the CPUs on our SMP or MPP system can provide instantaneous performance improvements, assuming that we are not I/O bound. The speed of CPUs is constantly being improved as are new and better cache designs. For SMP or MPP machines, the process of enhancing the CPU may be as simple as adding an additional CPU board. Before we purchase an additional processor of the same type and speed, however, consider upgrading to a faster processor. For example, upgrading from a 66 MHz processor to a 133 MHz processor may provide more benefit than purchasing an additional 66 MHz CPU with the added benefit that we now have the option of adding more 133 MHz CPUs. Because of the complexity and run time required by these queries, we can benefit from more and faster CPUs. SMP and MPP computers provide scalable CPU performance enhancements at a fraction of the cost of another computer. When upgrading our processors or adding additional processors, remember that our I/O and memory needs will probably increase along with the CPU performance. Be sure to budget for more memory and disk drives when we add processors.

I/O Enhancements

We can enhance I/O by adding disk drives or purchasing a hardware disk array. The data warehouse can benefit from the disk striping available in both hardware and software disk arrays.

Using Oracle data file striping can also help the performance of our data warehouse.

If our system performs only one query at a time and we are not taking advantage of the Oracle Parallel Query option, we may not see a benefit from a hardware or software disk array. In this specific case, we do not recommend OS or hardware striping; we should use traditional Oracle striping. Because we are executing only one query at a time without using the Parallel Query option, the I/Os to the data files are purely sequential on the table scans. This scenario is somewhat rare; any variance from "pure table scans" results in degraded performance. Hardware and software disk arrays have the added benefit of optional fault tolerance. We should first choose the correct fault tolerance for our needs and then make sure that we have sufficient I/O capabilities to achieve the required performance level. If we use fault tolerance, we will most likely have to increase the number of disk drives in our system. Another benefit of hardware disk arrays is caching. Most disk arrays on the market today offer some type of write or read/write cache on the controller. The effect of this cache is to improve the speed of writing to the disk; the cache also masks the overhead associated with fault tolerance. If our queries often perform table scans, we may see good improved performance with disk controllers that take advantage of read-ahead features. Read-ahead occurs when the controller detects a sequential access and reads an entire track (or some other large amount of data) and caches the additional data in anticipation of a request from the OS. Unlike an OLTP system in which this is just wasted overhead, in the data warehouse where we are performing DSS queries, it is likely that we will need that data soon; if we do, it will be available very quickly. Enhancements to the I/O subsystem almost always help in a data warehouse environment because large amounts of data are accessed. Be sure that we have a sufficient number of disk drives, properly configured. An I/O bottleneck is usually difficult to work around. As with all types of systems, a well-tuned application is very important.

4. Conclusion

Tuning your SQL statements is one of the most important tasks you can do to improve performance.

In fact, you should tune your SQL statements before tuning your RDBMS server. By tuning your SQL statements to be as efficient as possible, you can use

your system to its full potential. Some of the things you can do to improve the efficiency of your SQL statements may involve as little effort as rewriting the SQL to take advantage of a property of your database

References

[1] Hitesh Kumar Sharma, Aditya Shastri, Ranjit Biswas, "A Framework for Automated Database TuningUsing Dynamic SGA Parameters and Basic Operating System Utilities", Database Systems Journal", Academy of Economic Studies-Bucharest, Romania.

[2] Hitesh Kumar Sharma, Sandeep Kumar, Sambhav Dubey, Pawan Gupta, "Auto-selection and management of dynamic SGA parameters in RDBMS", Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference.

[3] Hitesh Kumar Sharma, Aditya Shastri, Ranjit Biswas, "SGA Dynamic Parameters: The Core Components of Automated Database Tuning", Database Systems Journal", Academy of Economic Studies-Bucharest, Romania.

[4] S. Elnaffar, W. Powley, D. Benoit, and P. Martin, "Today's DBMSs: How Autonomic are They?", Proceedings of the 14th DEXA Workshop, Prague, 2003, pp. 651-654.

[5] D. Menasec, Barbara, and R. Dodge, "Preserving Qos of E-Commerce Sites through Self-Tuning: A Performance Model Approach", Proceedings of 3rd ACM-EC Conference, Florida, 2001, pp.224-234.

[6] D. G. Benoit, "Automated Diagnosis and Control of DBMS resources", EDBT Ph.D Workshop, Konstanz, 2000.

[7] B. K. Debnath "SARD: A Statistical Approach for Ranking Database Tuning Parameters" 2007.

[8] K. P. Brown, M. J. Carey, and M. Livny, "Goal-Oriented Buffer Management Revisited", Proceedings of ACM SIGMOD Conference, Montreal, 1996, pp. 353-364.

[9] P. Martin, H. Y. Li, M. Zheng, K. Romanufa, and W. Poweley, "Dynamic Reconfiguration Algorithm: Dynamically Tuning Multiple Buffer Pools", Proceedings of 11th DEXA conference, London, 2002, pp.92-101.

[10] P. Martin, W. Powely, H. Y. Li, and K. Romanufa, "Managing Database Server Performance to Meet Qos Requirements in Electronic Commerce System", International Journal of Digital Libraries, Vol. 8, No. 1, 2002, pp. 316-324.

[11] S. Duan, V. Thummala, S. Babu, "Tuning Database Configuration Parameters with iTuned", VLDB '09, August 2428, 2009, Lyon, France.

[12] H. K. Sharma, A. Shastri, R. Biswas " Architecture of Automated Database Tuning Using SGA Parameters" , Database Systems Journal vol. III, no. 1/2012.

[13] A. G. Ganek and T. A. Corbi, "The Dawning of the Autonomic Computing Era", IBM Systems Journal, Vol. 42, No. 1, 2003, pp. 5-18.

[14] H. K. Sharma, A. Shastri, R. Biswas "A Framework for Automated Database Tuning Using Dynamic SGA Parameters and Basic Operating System Utilities", Database Systems Journal vol. III, no. 4/2012.

Dr. Hitesh Kumar Sharma: Author is an Assistant Professor (Senior Scale) in University of Petroleum& Energy Studies, Dehradun. He has published 40+ research papers in International Journal and 10+ research papers in National Journals.

Christalin Nelson. S: Author is an Assistant Professor (Selection Grade) in University of Petroleum& Energy Studies, Dehradun. He has published 40+ research papers in International Journal and 12 research papers in National Journals. He is Head of Department of Analytics.