The Oracle Magician

July, 2004 *OracleMagician.com* *Volume III, number 2*

From the Editor

Welcome to the sixth issue of the magazine, The *Oracle Magician*! This occasional newsletter focuses on various "tricks of the trade" in the Oracle world--from DBAs, architects, developers, designers, and report writers.

Thank you to the many notes and comments from readers of my book, *The Art & Science of Oracle Performance Tuning*. Your notes are most appreciated!

In this issue we present suggestions on avoiding performance problems in large, multi-terabyte databases. This article is based on my experience tuning VLDB systems.

We also provide a neat script that you can use to find the statistics for all the "slave" processes invoked during parallel processing.

As always, we accept ideas or articles from reads that have interesting performance ideas. .

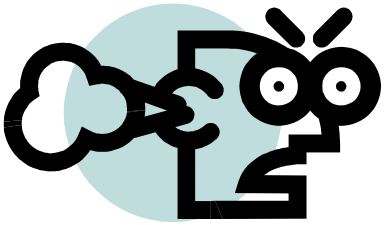
Please send all ideas to Editor@OracleMagician.com

Performance Tuning of Terabyte Databases: *Tips & Traps*

By Chris Lawson

VLDB Performance

Many performance tuning principles apply to both small and multi-terabyte databases. The stakes, however, are much higher with VLDB (Very Large Database) systems.



For instance, a slight optimizer error on a 10 Gigabyte database is a whole lot different than a similar error on a 4 TB database.

The vast majority of performance problems occur because the design did not really consider the final size of the database. That is, the program was tested for *functionality*, but not for performance. We often say that the program "won't scale."

I have found that most of the VLDB performance problems fall into these categories:

- Index problems
- Single-row processing
- Inefficient join method
- Wrong join order
- Mismatched partition sizing
- Wrong SQL hints

Continued on page 2

Table of Contents

Continued from page 1

The Basics

Many performance problems are due to forgetting very basic practices.

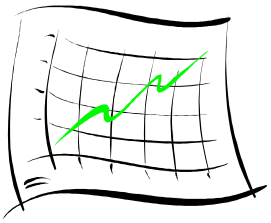
Missing indexes

Believe it or not, simple indexing errors are often the cause of big problems, even on huge databases.

Bad Statistics

Just like smaller databases, having outdated statistics will drastically hurt VLDB systems. You simply can't expect the optimizer to perform well when you feed it misinformation.

Generally speaking, a query should start a join with the table that has the most restrictive condition.



Remember that the sample size, as a percent, can usually be very small—even 1% or less for some large tables. For a table of 100 million rows, even a 1% sample means 1 million rows, which is probably far larger than necessary.

Other Problems

Single-row processing

It is quite common to see a SQL statement that is executed nicely, but is executed far too often. This usually happens because the designer has built a PL/SQL procedure that loops through all the data.

Inefficient join method

In large data warehouses, many queries return a huge result set. For example, the users might need to run a report that reads all inventory information for an entire month or quarter. This typically means retrieving *millions* of rows. With such a large result set, the *hash join* method is usually best. It is very common, however, to discover that the optimizer has selected the nested-loop method, which is more appropriate for a small result set.

Wrong join order

Generally speaking, a query should start a join with the table that has the most restrictive condition. This reduces the workload on the database engine as soon as possible—before the subsequent joins. Of course, the optimizer will endeavor to do just this, but it can make mistakes. It will sometimes be necessary to use the SQL hint ORDERED or LEADING to force the optimizer to alter the join order.

Join order is especially important in data warehouses because data warehouse queries typically involve more tables than in smaller, OLTP systems. It is not unusual to have 10 or more tables joined. I recall one monstrous query that used 28 tables and inline views! (It took me all day just to understand what the query was trying to do.)

Inappropriate Partition Sizes

Oracle partitioning has the potential to save a lot of runtime, but great care must be exercised. Probably the greatest advantage of partitioning is something called *partition pruning*. This means the elimi-

Continued on page 3

Continued from page 2

nation of partitions that are not needed to resolve a given query.

For instance, suppose you want to summarize sales data for a given month. Ideally, you only want to scan the relevant information; you don't want to wade through all the data for irrelevant months. If your table is partitioned by month, the optimizer will cleverly remove, or *prune*, the unnecessary partitions from consideration. If you have two years of information online, this means that you have eliminated 23/24 partitions. Naturally, this leads to a huge performance gain.

What happens, however, if the partition size doesn't match the report size? Suppose you run a report that needs one *quarter* of information. The optimizer can still prune some partitions, but now it must combine three of the monthly partitions. Conversely, if you only need 1 *week* of information, the best the optimizer can do is to scan one month.

As the query timeframe begins to deviate from the partition size, the performance gain fades. When there is a large discrepancy, you can even see performance that is *worse* than if you didn't use partitioning at all.

Wrong SQL hints

SQL hints are very powerful—in the right hands. A wrong SQL hint can cause either a tremendous improvement, or a devastating degradation on performance. It is common to see an inappropriate SQL hint forcing inappropriate index usage. This happens because the DBA mistakenly assumes that indexes are always better.

Sometimes, I see cases where the SQL hint doesn't really do anything at all, since the syntax is incorrect.

How can I Avoid These Problems?

Many performance problems, whether on VLDB or OLTP systems, can easily be avoided. Here are some suggestions:

Recognize that being functionally correct is not good enough.

A report may make all the right calculations, and look really pretty, but if the design won't scale, it's not very helpful. Help management to understand that a report design should not be considered complete without some type of performance testing on a large database.

Recognize that using indexes is sometimes the worst possible thing to do.

Retrieving huge amounts of data from a VLDB means that you must be wary of incorrect index usage. Data warehouses must often perform full partition scans, which is often better than using *any* index.

Recognize the inherent performance problems with single row processing

When millions of rows are involved, performing "one row at a time" processing is usually a disaster. Instead, let Oracle process an entire

group of data in one big chunk.

Remember that parallel processing can never make up for an inefficient design.

If you have a query that wrongly uses indexes, and requires 10 million logical reads, launching 5 processes in parallel doesn't solve anything—it only spreads the pain across multiple cpus. It is best to only activate parallel processing after you know that the query or transaction runs as efficiently as possible.

Become an Expert on SQL hints and Join Order

SQL hints are critically important, especially on VLDB systems. If you don't know which hint to use, or how to change the join order in a complex query, you will be severely handicapped.

Don't Settle for Poor Performance

By following these suggestions, you can preclude many performance headaches. Tuning a VLDB system can be a little more complicated, but the results are definitely worth the effort

The Oracle Magician

Editor: Chris Lawson
OracleMagician.com

Performance Statistics for Parallel Slaves

Here's a script that will list all critical statistics for all of the slaves used in parallel processing. You provide the SID of the parent SID:

```
select name, sum(value) from
V$Sesstat one, V$Statname two
where Sid in(
-- GET THE SIDS
select x.sid
from v$px_process x,v$lock l,
V$Session_Wait w
where x.sid <> l.sid(+)
and to_number
(substr(x.server_name,2)) = l.id2(+)
  and l.sid = w.sid(+)
  and nvl(l.type,'PS') = 'PS'
  and x.status not like 'AVAIL%'
  AND w.SID = &1)
  And One.Statistic# = Two.Statistic#
  and name not like '%memory%'
  and name not like '%time%'
  and name not like 'bytes%'
  group by name
  Having sum(value) > 10000 order by 2
```

Updates & Notes

If you find these tuning issues interesting, I discuss performance tuning in greater detail in my book, *The Art and Science of Oracle Performance Tuning*. It is available at most large bookstores, or online at Amazon.com.