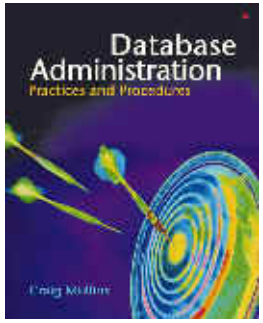
# Craig S. Mullins

Return to Home Page

December 2006


## The DBA Corner

*by Craig S. Mullins*




### The Two Biggest Database Performance "Things"

Database performance is one of those perennial topics that people just can't seem to get enough of. I guess this is so because the performance of database applications is one of the bigger issues that end users complain about. And DBAs can be heroes if they can resolve performance problems quickly. It also could be that performance problems are so ubiquitous because people keep on making the same design and coding mistakes.

With this in mind, let's take a look at my opinion on the two biggest things you need to control to keep database and SQL performance in check.

## (1) Keep Statistics Up-to-Date

Without statistics, the relational optimizer cannot accurately optimize anything. Database statistics provide information about the state and organization of the data in the database. The optimizer matches these statistics against the tables used in your SQL statements and applies query cost formulas to determine the optimal way to get at your data. The type of statistical information gathered by relational optimizers includes:

- Table details including total number of rows, compression percentages, and total number of blocks;
- Column details including the number of discrete values and the distribution range of values stored in each column;
- Tablespace information such as the number of active pages and cluster ratio;
- Index statistics including the number of leaf pages and levels, the number of discrete values for the index key, and whether the index is clustered;
- Additional information about the table space and index node groups or partitions.

Statistics are populated when you execute a utility program or command, something like RUNSTATS or UPDATE STATISTICS, depending on the DBMS you are using. Be sure to work with your DBAs to accumulate statistics at the appropriate time in the production environment.

## (2) Build Appropriate Indexes

Perhaps the most important thing you can do to assure optimal performance of your database applications is to create the correct indexes for your tables. Of course, this is easier said than done. But we can start with some basics. For example, consider this SQL statement:

```
SELECT LASTNAME, SALARY
FROM EMP
WHERE EMPNO = '000010'
AND DEPTNO = 'D01';
```

What index or indexes would make sense for this simple query? First, think about all the possible indexes that you could create. Your first short list probably looks something like this:

- Index1 on EMPNO
- Index2 on DEPTNO
- Index3 on EMPNO and DEPTNO

This is a good start, and Index3 is probably the best choice. It lets the DBMS use the index to immediately look up the row or rows that satisfy the two simple predicates in the WHERE clause. Of course, if you already have a lot of indexes on the EMP table, you might want to examine the impact of creating yet another index on the table. Factors to consider include:

***Modification impact***: The DBMS must automatically maintain every index you create. This means every row inserted and every row deleted will insert and delete not just from the table, but also from its indexes. And if you UPDATE the value of a column that is in an index, you also update the index. So, indexes speed the process of retrieval but slow down modification.

***Columns in the existing indexes***: If an index already exists on EMPNO or DEPTNO, it might not be wise to create another index on the combination. However, it might make sense to change the other index to add the missing column – but not always. The order of the columns in the index can make a big difference depending on the query. For example, consider this query:

```
SELECT LASTNAME, SALARY
FROM EMP
WHERE EMPNO = '000010'
AND DEPTNO > 'D01';
```

In this case, EMPNO should be listed first in the index. And DEPTNO should be listed second, allowing the DBMS to perform a direct index lookup on the first column (EMPNO) and then a scan on the second condition (DEPTNO).

Furthermore, if indexes already exist for both columns (one for EMPNO and one for DEPTNO), some DBMS products can use them both to satisfy this query so creating another index might not be necessary.

***Importance of each query***: The more important the query, the more you might want to tune by index creation. If you are coding a query that the CEO will run every day, you want to make sure it delivers optimal performance. So building indexes for that particular query is important. On the other hand, a query for a clerk might not necessarily be weighted as high, so that query can make do with existing indexes. Of course, the decision depends on the application's importance to the business and not just on the user's importance.

Additionally, you might consider index overloading to achieve index-only access. If all the data that a SQL query asks for is contained in the index, the DBMS might be able to satisfy the request using only the index. Consider our previous SQL statement. We asked for LASTNAME and SALARY, given information about EMPNO and DEPTNO. And we also started by creating an index on the EMPNO and DEPTNO columns. If we include LASTNAME and SALARY in the index as well, we never need to access the EMP table because all the data we need exists in the index. This technique can significantly improve performance because it cuts down on the number of I/O requests.

Keep in mind that making every query an index-only access is not prudent or even possible. You should save this technique for particularly troublesome or important SQL statements.

Proper index design involves much more than can be covered here. I've just touched on the basics.

## Summary

If you are just embarking on your journey into the wonderful world of database performance management, please, start with the two items covered in this short column. But keep in mind that we are just scratching the surface of both areas and you can benefit by additional research and education in both statistics gathering and index design. And if you are a long-time database professional, it can't hurt to bone up on these topics either. You might learn about some newer feature or function that you haven't used yet, or maybe just strengthen what you already know.

From Database Trends and Applications, December 2006.

Home.