

## **RDBMS application tuning : don't change your RDBMS parameters ... not immediately at least**

*Stéphane Faroult, Oriole Corporation*

When systems go into production and begin to reach cruising speed you can expect complaints about poor performance - something you could add to death and taxes as one of the few certainties in life.

Many a knowledgeable database administrator tends to fight performance problems with the usual DBA weapons - reorganizing the database, moving files around, increasing memory and altering arcane parameters. This can bring performance gains of 20 to 30% - not bad, but not enough if you want to shine as a guru.

If you really want to get massive performance gains, you have to strike at the heart. This can be done surprisingly easily, even with packaged software. When looking for system performance improvement, you have two priority areas to explore.

### 1. Indexing

Indexes have always been at the core of performance tuning; but today's indexing problems are a quantum leap from indexing problems in the days when RDBMS users were just a bunch of pioneers. Databases are quite often overindexed these days, and packaged software such as the ubiquitous ERP program is especially blameworthy in this respect - since it modestly attempts to fulfill the needs of businesses ranging from the shipyard to the Mom and Pop grocery store, many indexes are added 'just in case', including indexes on columns which may be highly selective in some cases, but, tough luck, not in yours. Unfortunately, consultants who tailor this type of system to your needs are often ERP experts, but their skills rarely encompass the grass-roots, unsexy database design side as well.

Overindexation means that, in the best case, the optimizer is smart enough not to use many of the indexes - but you will pay a heavy overhead cost during inserts (useless indexes have to be inserted too) and updates which affect the indexed columns. At worst, the optimizer may mistakenly choose to use an index which will slow down, rather than speed up, fetches; the additional processing required by fetching row addresses in indexes before fetching the rows themselves outweighs the benefits of direct data access faster than most people realize. In fact, when more than 10% of rows are returned, a full table scan is often as fast if not faster than an index search.

The database dictionary can answer most of your questions regarding indexes and help you spot those which are at least questionable if statistics have been previously collected :

- Indexes on small tables (less than 100 rows) are totally useless, unless of course they are here to implement unique constraints (primary or unique keys)
- Indexes for which there are less than 8 different key values should come under very close scrutiny (this of course refers to 'traditional' tree-type indexes, not bitmap ones). Here, the distribution of values is of course extremely important; if the values are roughly equally distributed, the index can be scrapped. If, however, searches are for keys which belong to a small minority, then the index can be kept. It can make sense to have an index on a personnel table on the 'SEX' column if you are frequently searching for females in a predominantly

male environment or vice versa. However, indexing in a security trading system, a column which says whether the order is a buy or a sale - something I have seen recently - is totally senseless.

- Concatenated indexes should be concatenated from the most selective to the least selective column; this is an old, well-known rule which needs to be emphasized again and again, especially as the self-styled, highly-flexible packaged software may well index a series of columns the first of which contains a single value at your company.
- Indexes on single columns which also appear as the first column in a concatenated index are redundant. Database engines can use the concatenated index when only the first columns in the index appears in the search condition.

Many useless indexes are unfortunately often automatically created on foreign keys by database design tools - something which I suspect comes from a locking peculiarity with Oracle (there is an exclusive lock on the referenced table when the referencing table is updated or inserted if the columns in the foreign key are not indexed); whereas I have indeed encountered a few cases when this was a concern, in many cases reference tables will not be updated concurrently with the referencing ones, and the only consideration when deciding whether indexes on foreign keys which refer to static tables must be kept, should be whether they actually speed up searches on the columns in the foreign key or joins.

For Oracle users, the [idxaudit.sql](#) script can help you identify those indexes which are questionable.

Changes to the indexing of the tables used by a software package should of course be made only with the technical blessing of the software provider - something which you should be easy to obtain when these changes are the price of customer satisfaction.

## 2. Unveiling the costly statements

When users complain about performance, the first thing to do is to understand why it is so slow. The right answer to this question is not, of course 'because the CPU is 100% busy'; what we want is what happens. When you can point with some certainty to database-related activity (a poorly tuned operating system is quite another matter), then it is not always easy to pinpoint a specific process. Bursts of activity and peaks are often noticed just a shade too late to catch the specific statement responsible for them; however, it is sometimes possible to write a script which relates high CPU consumption and database activity such as [oraproc.ksh](#), which helps, but while it improves your chances of really seeing what goes wrong, it's still no sure bet.

A number of (usually expensive) tools are also able to monitor the operating system and database activity simultaneously, and then relate both. Besides the price, the drawback is obviously the simple fact of monitoring, which always has its cost whatever the tool proponents try to pretend. And of course the more things you monitor, the higher the cost.

In fact, pure database indicators are so closely related to CPU consumption or I/O that trying to keep an eye on everything is usually an expendable luxury. With Oracle for instance, the database kernel always collects statistics on statements such as the number of logical reads executed to process the statements, a value which comes as

close as you can to CPU consumption. Even the tightfisted can get good results with a script such as `<HREF>peep.sql</HREF>` which will at least tell you where all the CPU cycles have gone.

Once this is done, either the programs you are monitoring belong to a software package, in which case the thing to do, once you have identified the statements which hurt, is to try your charms - or the bargaining power of your company - on your customer representative to have problems fixed quickly; or they have been developed in-house, which puts you in a better position.

Putting your findings under the nose of whomever it may concern will certainly help, but beware of believing that tuning individual statements is always the smartest thing to do - something that self-styled 'expert-systems' assume implicitly.

More and more often, even with a decent database design, performance problems stem from algorithms - several statements to select or update data which could be selected or updated in a single pass, for instance, or an inefficient ordering of statements. A classical example is the 'update/insert if not found' processing which can be written in different ways (*check for the key, if found update, else insert*; or, *insert, if a primary key is violated update*; or, *update, if no row is updated insert*); the least efficient one 'costs' twice as much as the most efficient one, and yet all statements executed are, taken individually, impeccable.

You will be surprised at the performance gains that a thorough review of indexes and the identification of the costly statements - an amazingly small number most often - will bring to you. Then you'll be able to improve the physical lay-out of your database and fine-tune the RDBMS parameters - but do it on sound foundations ...

*Stéphane Faroult became interested in Oracle tuning with Version 5, back in early 1987, and has kept being strongly interested in the subject since then ... He now works with Oriole Corporation ([www.oriolcorp.com](http://www.oriolcorp.com)) which provides server-based tools and utilities. He is reachable by e-mail at [sfaroult@oriolcorp.com](mailto:sfaroult@oriolcorp.com)*