Craig S. Mullins

Return to Home Page

Vol. 12, No. 2 (October 2005)

From IDUG Solutions Journal...

The Buffer Pool

Do the Simple Things First

By Craig S. Mullins

Simplification is an imperative in this day and age of increasing complexity and ever-changing software environments. A key component of simplification, in my opinion, is to remember the basics and apply some simple rules and practices to our DB2 subsystems and applications. Indeed, many troubles surface because we don't keep track of things we already know.

This principle is backed up in the recent best-selling book ***Blink : The Power of Thinking Without Thinking***

by Malcolm Gladwell (published by Little, Brown, 2005, ISBN: 0316172324). Through the use of case studies and examples Gladwell introduces us to the power of our "adaptive unconscious" – a powerful innate ability that provides us with instant and sophisticated information. Basically, it boils down to using our experience to arrive at quick decision that are usually correct. As I read this book I pondered how its nuggets of wisdom could be adapted to how we manage DB2 systems..

So, what are the basics that we should always keep top-of-mind? Let's examine some of the primary issues and concepts that need to be addressed in order to keep a DB2 implementation humming along.

Updating Statistics

Simple negligence is a common cause of performance problems in many DB2 subsystems. DB2 needs to have accurate database object statistics in order to create efficient strategies for data access. Indeed, at IDUG in Prague last year an IBM speaker said that "Of customer problems with 'bad' access paths, around half of them were because of bad or missing statistics." That is quite

a telling indication of how a lot of performance problems occur. Put quite simply: it is our fault.

To avoid such problems make sure you run a RUNSTATS utility on a regular basis. As the volume and nature of data in your databases changes, DB2 must be made aware of the changes or performance will suffer. The RUNSTATS utility collects statistical information for DB2 tables, table spaces, partitions, indexes, and columns. When it populates this information into the DB2 Catalog the statistics are available for subsequent use by the DB2 optimizer in formulating efficient access paths. Of course, the statistics in these tables also can be used by DBAs to help determine when to reorganize. Without up-to-date statistics, both DB2 and the DBAs managing DB2 are at a disadvantage.

So just how frequently should RUNSTATS be run? Of course, the answer is “it depends.” But here is what it depends upon: how frequently the data changes, how large the object is, and data access activity. The cost of RUNSTATS usually is negligible for small- to medium-size table spaces. Of course, RUNSTATS will take longer to execute for larger objects, so plan wisely before executing RUNSTATS for very large table spaces and

indexes. You cannot avoid running RUNSTATS for larger objects because the statistics are perhaps even more important the larger the object becomes. If the data in a large object changes slowly you can probably run RUNSTATS once, and then delay running it again for a long time (until the data changes significantly). But for very volatile data, be sure to execute the RUNSTATS utility at least monthly. And you should consider sampling to reduce the runtime duration for larger objects.

You can run RUNSTATS with SHRLEVEL CHANGE to accumulate statistics without limiting concurrent activity to an object. Of course, the statistics will be more accurate if SHRLEVEL REFERENCE is used, but in today's 24/7 world such a luxury is not usually possible. At any rate, it is wise to run RUNSTATS during periods of low activity to reduce the impact of the concurrent access to both the applications and the statistics gathering process.

After running RUNSTATS the newly updated statistics are available for use. Of course, if you do not REBIND your applications your access paths for static SQL will not change. Dynamic SQL can take advantage of the

new statistics immediately. And you can examine the new statistics to determine whether your objects need to be reorganized. This is the next “simple thing” you need to keep under control.

Reorganizing When Necessary

Now that the statistics are correct we can use them to schedule reorganizations of our database objects. Reorganization is required periodically to ensure that the data is situated in an optimal fashion for subsequent access. Reorganization re-clusters data, resets free space to the amount specified in the DDL, and deletes and redefines the underlying VSAM data sets (for STOGROUP-defined objects). There are three types of reorganizations supported by the DB2 REORG utility:

- When REORG is run on an index, DB2 reorganizes the index space to improve access performance and reclaim fragmented space.

- When REORG is run on a regular (non-LOB) table space, DB2 reorganizes the data into clustering sequence by the clustering index, reclaims fragmented space, and optimizes the organization of the data in the table space.

- When REORG is run on a LOB table space, DB2 removes embedded free space and tries to make LOB pages contiguous. The primary benefit of reorganizing a LOB table space is to enhance prefetch effectiveness.

Proper planning and scheduling of a REORG utility requires an examination of the statistics in the DB2 Catalog and an understanding of how the object is being used. You can follow some general rules of thumb to help guide your reorganization planning.

DB2 provides numerous statistics that are useful in determining when to reorganize. These include PERCDROP, PAGESAVE, NEAROFFPOSF, FAROFFPOSF, NEARINDREF, FARINDREF, LEAFDIST, and CLUSTERRATIOF. Although an in-depth discussion of each of these statistics is beyond the scope of this article, the next couple of paragraphs will outline some basic tactics for determining when to REORG.

One rule of thumb for smaller indexes is to reorganize when the number of levels is greater than three. For indexes on larger tables, three (or more) levels may be completely normal. Other indicators that signify an index reorganization may be needed include when the

LEAFDIST value is large or PSEUDO_DEL_ENTRIES has grown.

The cost of reorganizing an index is small compared to the cost of reorganizing a table space. Sometimes, simply executing REORG INDEX on a table space's indexes can enhance system performance. Reorganizing an index will not impact clustering, but it can do the following:

- Possibly impact the number of index levels.
- Reorganize and optimize the index page layout, removing inefficiencies introduced due to page splits.
- Reset the LEAFDIST value to 0 (or close to 0).
- Reset PSEUDO_DEL_ENTRIES to 0.
- Reduce or eliminate data set extents.
- Apply any new PRIQTY, SECQTY, or STOGROUP assignments.
- Reset free space.

Additionally, reorganizing indexes using SHRLEVEL CHANGE is simpler than reorganizing table spaces online because REORG INDEX SHRLEVEL CHANGE does not use a mapping table. This makes reorganizing indexes with concurrent data access easier to administer and maintain.

Consider reorganizing table spaces when CLUSTER RATIO drops below 95 percent for its clustering index or when FARINDREF is large. Reorganizing a large table space as soon as the CLUSTER RATIO is not 100 percent could produce significant performance gains.

Also, if you have enabled Real Time Stats (RTS) be sure to examine the columns of SYSIBM.TABLESPACESTATS and SYSIBM.INDEXSPACESTATS that provide information on REORG, REBUILD, and RUNSTATS. The RTS tables can help you to determine when to run a REORG by examining the last time a utility was run and what has happened since. Of course, RTS does not obviate the need to run RUNSTATS.

When you run a table space REORG consider running an inline RUNSTATS. Doing so will cause DB2 to gather new statistics during the reorganization process. It is more

efficient than running a separate RUNSTATS after the REORG – and you'll want updated statistics after reorganizing. To generate inline RUNSTATS, use the STATISTICS keyword.

Remember, when scheduling your REORG jobs to take into account the level of concurrent activity to be allowed during the reorganization. There are three SHRLEVEL options for REORG: NONE, REFERENCE, and CHANGE. Allowing concurrent access during the reorganization requires a shadow copy of the table space and requires additional administrative effort – but it will not disrupt production work. Of course, consider running online REORGs only within reason – you do not want to run an online REORG during the busiest portion of the workday.

Coding Appropriately

Another pervasive problem permeating DB2 systems is the "flat file" development mentality. What I mean by this is when a programmer tries to access DB2 data the same way that he would access data from a flat file. DB2 is 'relational' in nature and, as such, operates on data a set-at-a-time, instead of the record-at-a-time processing

used against flat files. In order to do justice to DB2, you need to change the way you think about accessing data.

To accomplish this, all users of DB2 need at least an overview education of relational database theory and a moderate to extensive amount of training in SQL. Without such a commitment your programmers are sure to develop ugly and inefficient database access code – and who can blame them? Programmers are used to working with files so they are just doing what comes naturally to them.

SQL is designed so that programmers specify ***what*** data is needed but they cannot specify ***how*** to retrieve it. SQL is coded without embedded data-navigational instructions. The DBMS analyzes SQL and formulates data-navigational instructions "behind the scenes." This is foreign to the programmer who has never accessed data using SQL.

Every SQL manipulation statement operates on a table and results in another table. All operations native to SQL, therefore, are performed at a set level. One retrieval statement can return multiple rows; one modification statement can modify multiple rows. This feature of relational databases is called relational closure.

When accessing data, a programmer needs to think about what the end result should be and then code everything possible into the SQL. This means using the native features of SQL – joins and subselects and functions, etc. – instead of coding procedural COBOL or Java and processing tables like files.

Educating programmers how to use SQL properly is probably the single most important thing you can do to optimize performance of your DB2 applications.

Building the Correct Indexes

The final "simple thing" we will discuss in this article is building proper indexes. This is a job for the DBA and the proper way to do it is by workload, not by database object. What do I mean by that?

Most of the time, when we build new databases we create groups of objects. We'll create a database, then groups of table spaces and tables. And every time we create a new table we usually create the indexes on that table. This approach is not the best.

Instead, we should build indexes based on workload. Indexes should support the predicates in the SQL that is written to access your tables. Building indexes to support predicates of the most frequently executed

queries and most important queries should be your first indexing step after building the unique indexes required to support primary keys and unique constraints.

Of course, this requires knowledge of how your tables will be accessed. And when you are first creating tables, you will not have any SQL. Sometimes you may have vague pseudo code descriptions of potential queries, but you won't have an accurate picture of access. Therefore, indexing has to be an incremental task, performed on an ongoing basis as code is written against your databases.

As you continually monitor and build new indexes, be sure to review the old ones that were created. Remember, although indexes can improve SELECT access, they will degrade the performance of INSERTs and DELETEs (as well as any UPDATEs of indexed columns). So, be sure to drop those unused indexes.

You should also take care to learn your applications and the type of indexes that can best serve them. Of course, this coverage of indexing techniques has been necessarily high-level. Skillful, performance-oriented

indexing is a skill that can take time to master. So, maybe I was a bit rash in calling it a “simple” thing.

Summary

Be aware that this article has offered a simplified list of things to focus on – but that is the point. Yes, DB2 management is a complex, arduous task. But by paying attention to the basics and making sure you do not take shortcuts around necessary processes, the complex things can be addressed more easily – because you can be sure that the simple things have been handled appropriately.

Yes, I know, there always seems to be “time to do it over, but never time to do it right.” By paying attention to the details and making sure that the small things don’t become big problems, the time needed to “do it over” should diminish. And then you’ll be able to spend more time on the “big” issues when they inevitably arise.

From *IDUG Solutions Journal*, October 2005.

Home.