THE BUFFER POOL

Things You've Probably Neglected

By Craig S. Mullins

We're all busy. Frequently it can seem like you just got in to the office and already it is past quitting time! There is so much to do and so little time to do it all. And we all work more than 40 hours a week… these are some of the common complaints of the busy DBA.

OK, so maybe you all have a point. But I'm going to pile on to your problems because there are some very critical issues that many, if not most, DB2 database administration staffs have yet to conquer. And if you take some time to get the things I'm going to talk about implemented, some of the tasks that you seemingly perform on a daily basis can be reduced.

So please take a little bit of time to read through the next couple of pages and give some thought to how you address each of these areas.

REBIND Strategy

Rebinding is as good a place to start as any. One of the most important contributors to the on-going efficiency and health of your DB2 environment is proper management of DB2 access path changes. A thorough REBIND management process is a requirement for healthy DB2 applications.

But many shops do not do everything possible to keep access paths up-to-date with the current state of their data. Approaches vary, such as rebinding only when a new version of DB2 is installed, whenever PTFs are applied to DB2, or to rebind automatically after a regular period of time. Although these methods are workable, they are less than optimal.

The worst approach though is the "if it ain't broke don't fix it" mentality. In other words, many DBA groups adopt "never REBIND unless you absolutely have to" as a firm policy. The biggest problem this creates is that it penalizes every program in your subsystem for fear of a few degraded access paths. This results in potentially many programs having sub-optimal performance because the optimizer never gets a chance to create better access paths as the data and environment change. Of course, the possibility

of degraded performance after a REBIND is real – and that is why many sites avoid regularly rebinding their programs.

Even so, the best approach ***is*** to perform regular REBINDs as your data changes. To do so, you should follow the Three R's. Regularly **r**eorganizing to ensure optimal structure; followed by **R**UNSTATS to ensure that the reorganized state of the data is reflected in the DB2 Catalog; and finally, **r**ebinding all of programs that access the reorganized structures. This technique can improve application performance because access paths will be better designed based on an accurate view of your data.

Of course, adopting the Three R's approach raises questions, such as "When should you reorganize?" To properly determine when to reorganize you'll have to examine statistics. This means looking at either RUNSTATS in the catalog or Real Time Statistics (RTS). So, the Three R's become the Four 4 R's – RUNSTATS (or better yet, Real Time Statistics, which we'll talk about in the next section), REORG, RUNSTATS, then REBIND.

Some organizations do not rely on statistics to schedule REORGs. Instead, they build reorganization JCL as they create each object – that is, create a table space, build and schedule a REORG job, and run it monthly or quarterly. This is better than no REORG at all, but it is not ideal because you are likely to be reorganizing too soon (wasting CPU cycles) or too late (causing performance degradation until REORG).

It is better to base your REORGs off of thresholds on catalog or real-time statistics. Statistics are the fuel that makes the optimizer function properly. Without accurate statistics the optimizer cannot formulate the best access path to retrieve your data because it does not know how your data is currently structured. So when should you run RUNSTATS? One answer is "as frequently as possible based on how often your data changes." To succeed you need an understanding of data growth patterns – and these patterns will differ for every table space and index.

The looming question is this: why are we running all of these RUNSTATS and REORGs? To improve performance, right? But only with regular REBINDs will your programs take advantage of the new statistics to build more efficient access paths, at least for static SQL applications.

Without an automated method of comparing and contrasting access paths, DB2 program change management can be time-consuming and error-prone – especially when we deal with thousands of programs. And we always have to be alert for a rogue access path – that is, when the optimizer formulates a new access path that performs worse than the previous access path.

Regular rebinding means that you must regularly review access paths and correct any "potential" problems. Indeed, the Four R's become the Five R's because we need to ***review*** the access paths after rebinding to make sure that there are no problems. So, we should begin with RTS (or RUNSTATS) to determine when to REORG. After

reorganizing we should run RUNSTATS again, followed by a REBIND. Then we need that fifth R – which is to review the access paths generated by the REBIND.

The review process involves finding which statements might perform worse than before. Ideally, the DBAs would review all access path changes to determine if they are better or worse. But DB2 does not provide any systematic means of doing that. There are tools that can help you achieve this though.

The bottom line is that DB2 shops should implement best practices whereby access paths are tested to compare the before and after impact of the optimizer's choices. By adopting best practices to periodically REBIND your DB2 programs you can achieve better overall application performance because programs will be using access paths generated from statistics that more accurately represent the data. And by implementing a quality review step there should be less need to reactively tune application performance because there will be fewer access path and SQL-related performance problems.

Real Time Statistics

Another sadly neglected area within many DB2 shops is the adoption of Real Time Statistics (RTS). RTS have been available for several years now, but they are not widely implemented. A proper understanding and implementation of RTS can reduce the need to schedule and run some RUNSTATS jobs, at least those RUNSTATS needed to determine when to reorganize. And with RTS you can be sure that you are making informed decisions because the statistics will be up-to-date, without ever having to run a utility to get them.

With RTS, DB2 gathers performance and maintenance statistics about database objects "on the fly," without having to run a utility program. DBAs are accustomed to scheduling the RUNSTATS utility to gather database statistics. You can think of RTS as similar to RUNSTATS, but without having to worry about when and how to run it.

Real time statistics cannot completely replace RUNSTATS because RTS are never used by the optimizer to determine access paths. But the RTS statistics can be used by DBAs to determine when to administer and maintain their DB2 databases.

If you think about this for a moment, RTS can help you to minimize RUNSTATS CPU consumption. Traditionally, RUNSTATS has been used for two reasons:

1. To gather statistics used by the optimizer
2. To gather statistics to be used by the DBA.

After enabling RTS, you can eliminate any RUNSTATS jobs that are being run simply to gather statistics that help you determine when to reorganize your databases. Thereafter, RUNSTATS needs to be run only for SQL optimization purposes. Indeed, by specifying UPDATE ACCESSPATH to direct RUNSTATS to collect only optimization statistics

you need never collect any other statistics using RUNSTATS once RTS has been implemented.

Some shops may be able to dramatically reduce the number of RUNSTATS jobs required. If you rarely, or never, REBIND your plans and packages, and you do not use dynamic SQL in production, then RUNSTATS will rarely need to be run. As you know, though, I am not advocating the avoidance of rebinding your applications – if you read the first section of this article, you know that I recommend doing just the opposite! However, I do want to point out an opportunity to save CPU by minimizing RUNSTATS executions if you utilize this admittedly suboptimal approach to rebinding.

An additional benefit of RTS over RUNSTATS is that they are more up-to-date. Heck, the phrase "real time" is right in their name. That means the queries you run against the RTS tables will be more accurate than those you run against the standard DB2 catalog tables, which will be only as recent as your last RUNSTATS. So your decision criteria will improve when they are based on RTS instead of RUNSTATS statistics.

Optimally, you would use one of the products on the market that automate maintenance tasks based on RTS, but with some work you could "roll your own" by writing some code that feeds a scheduler. Of course, the cost of a product can be offset by the savings and accuracy it offers; most DBAs have enough to do without worrying about maintaining additional code in a "roll your own" solution.

Keep in mind, too, that DB2 9 for z/OS brings improvements to RTS. First of all, the RTS tables are moved to the DB2 catalog. Previously, users were responsible for creating and managing these tables outside the scope of the DB2 catalog. Additionally, we get some new statistics in DB2 9, the most important of which offers usage information for indexes. A new column, LASTUSED in the SYSINDEXSPACESTATS table, contains a date indicating the last time the index was used. Any time the index is used to satisfy a SELECT, FETCH, searched UPDATE, searched DELETE, or to enforce a referential constraint, the date is updated.

A recurring historical problem in DB2 has been determining whether or not an index is being used. You can always query your PLAN_TABLEs for static SQL, but what about dynamic? That is more difficult. Now, you can simply query the LASTUSED column to see when the index was last used. Of course, you will have to give it some time because you might have an index supporting a rarely used query. Most shops have queries and programs that run quarterly, or even annually, but nevertheless are very important. So don't just start using RTS to drop indexes a month after you've migrated to DB2 9!

In DB2 9, RTS also takes advantage of the new data types, so RTS will better document table space and index storage. The TOTALENTRIES column (for indexes) and SPACE, TOTALROWS, DATASIZE, and UNCOMPRESSED_DATASIZE columns (for table spaces) all use the BIGINT data type, which can handle values as large as nine quintillion.

However, the biggest advantage of all proffered by RTS is the same as it was on the day it was introduced – the ability to use accurate, up-to-date statistics about DB2 objects to make administration decisions. The newer features have only improved our ability to do so. Continuing to ignore RTS will only make your administration efforts more difficult… and less accurate!

Archiving Dormant Data

Another problematic area that has been given little attention by many shops is handling dormant data. Most production databases are growing in size by leaps and bounds, but a lot of that data is not active. By active I mean the data participates in business transactions or is being actively queried and reported upon. But dormant data cannot be just deleted because much of it has to be retained for business or regulatory purposes. So it just sits there in the operational database, taking up space and causing problems.

As data volumes expand, it impacts operational databases in two ways:

1. additional data stresses daily processing, potentially impairing performance especially for scans, but even potentially for indexed access;
2. database administration tasks are negatively impacted.

In terms of performance, the more data in the operational database, the less efficient transactions running against that database tend to be. Table scans must reference more pages of data to return a result. Indexes grow in size to support larger data volumes, causing access by the index to degrade because there are more levels to traverse to return an answer.

The other impact is in increased time to perform database administration tasks. Image copies, REORGs, and even unloads take longer to run on larger databases. Recoveries (local and disaster recoveries) will take longer, too, because more data must be recovered. In many cases the lengthened outages have become unacceptable, causing companies to again seek ways to lighten up the operational databases.

These impacts on database and application performance and on-going database maintenance are causing many companies to seek solutions that archive data out of the operational system to an archive data store. Archiving removes the data from the production environment, but preserves the data so it can be accessed as needed for business, compliance, and e-discovery requirements.

Performance and administration issues are ancillary to the regulatory issues, but combined all of these issues drive the need to archive data out of the operational environment and into an archive data store. However, generally it is the legal requirements that have the biggest impact in terms of data volume expansion.

A proactive database archiving practice is the best approach for avoiding these growing data expansion problems, as well as addressing the long-term data retention requirements of industry and governmental regulations such as Sarbanes-Oxley, PCI DSS, and HIPAA.

Synopsis

Of course, these are not the only things you might have neglected, but they are probably the best places to start. Some organizations may be “behind the curve” and may not even

have appropriate backups for all of their table spaces. Other, more advanced organizations, may have moved on to try to tackle the production of appropriate audit trails. The bottom line, though, is that most organizations have areas that have been neglected; consider taking some time to reflect on the things yours has neglected, and come up with a plan for addressing them!