Winter 2010

The Buffer Pool

To Rebind or Not to Rebind: Why Is That Even a Question?

by Craig S. Mullins

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. Shops that conform to this approach never REBIND unless they absolutely have to. The biggest problem this approach creates is that it penalizes every program in your environment because of the fear that one, or maybe a couple of SQL statements will experience 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 changes.

Of course, the possibility of degraded performance after a REBIND is real – and that is why some sites have adopted this approach. However, a best practices approach to rebinding DB2 programs would be to r regularly REBIND and develop practical methods to deal with the troublemakers.

The Five Rs

The best practice methodology for DB2 programs is to perform regular REBINDs as your data changes. This has been summarized in the past as following The Three Rs, but as we will learn it is a few Rs short of being optimal.

So what are The Three Rs?

1. Regularly Reorganizing to ensure optimal structure
2. followed by RUNSTATS to ensure that the reorganized state of the data is reflected in the DB2 Catalog;
3. and finally, REBINDing all of programs that access the reorganized structures

This technique is professed to be able to improve application performance because access paths will be better designed based on an accurate view of your data. But anyone attempting to implement this approach will notice a few shortcomings. Trying to adopt 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 DB2 Catalog or, even better the Real-Time Statistics (RTS). RTS is better because the statistics will be up-to-date without having to invoke a utility that will consume CPU resources. So, the first step transforms The Three R's into The Four 4 R's – RTS, 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 real-time statistics (or RUNSTATS statistics if you are still stubbornly avoiding RTS). In this way you will be reorganizing at the right time, instead of based upon some pre-conceived (and probably inaccurate) notion of when a REORG might be beneficial.

Statistics (those generated by RUNSTATS) 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.

Additionally, there are multiple types of statistics that can be generated by RUNSTATS. The RUNSTATS utility can gather statistics about table, indexes, and columns. The parameters you choose dictate what type of statistics are generated and at what level of detail. You can also collect distribution statistics, correlation statistics and histogram statistics. These can help when data is skewed or correlated.

So what type of statistics should you collect? Well, an in-depth discussion of that topic warrants an article all to itself. But at a high level, you can follow these basic rules of thumb:

- Always collect the base statistics (table and index)

- Then collect column statistics for important WHERE and ORDER BY clauses
- Collect Non-Uniform Distribution Statistics when data is skewed (e.g. beer drinkers skew male)
- Collect correlation statistics when two or more columns are highly correlated (e.g. CITY, STATE, ZIP – the zip code 02120 is in the city Boston is in the state Massachusetts).
- Collect histogram statistics when data skews by range (e.g. Lunch rush or Christmas shopping season).

Getting back to our discussion on rebind strategy, we must keep in mind the reason we are running all of these RUNSTATS and REORGs. And that is to improve performance, right? But only with regular REBINDs will your programs take advantage of the new statistics to build more efficient access paths (except, of course, for dynamic SQL which can use the new statistics the next time a dynamic access path is created).

So it makes sense to REBIND regularly as we run RUNSTATS and REORGs. But again, what about the potential for degraded access paths? The optimizer is not perfect (though it is very good) and we are not perfect (perhaps not collecting the correct level or amount of statistics). What is needed is a way to examine the results of a REBIND in terms of its impact on SQL performance.

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.

Regularly rebinding means that you will need to review the resultant access paths and correct any "potential" problems. Indeed, The Four Rs become The Five Rs because we need to review the access paths after rebinding to make sure that there are no problems. So, we should begin an inspection of the realtime stats to determine when to REORG. After reorganizing we should run RUNSTATS, 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 third party tools that can help you achieve this though. Such a tool will pre-run a REBIND to generate new access path information without creating a new package. This information is compared to the existing access paths and run through a set of rules to determine if the new access path is improved, worse, or the same. Action can then be taken to avoid rebinding only those packages where the performance of any SQL will degrade. And the DBAs can dig in to find out the cause of the degradation before rebinding the problem packages.

The New REBIND Options Help, But…

Over the course of the past few releases IBM has added improvements to the REBIND capability. An important one is PLANMGMT, or package stability, which is a DB 9 for z/OS feature. It delivers the ability to keep backups versions of your program’s access paths. Why is this useful?

Well, as we discussed earlier, after rebinding your program, sometimes access paths degrade. When this occurs, PLANMGMT can be used to fall back to previous access paths (provided, of course, that we specified the appropriate PLANMGMT option).

There are three options that can be chosen:

- **PLANMGMT(OFF)** - No change to existing behavior. A package continues to have one d no backup access paths
- **PLANMGMT(BASIC)** - A package has one active copy. One additional prior copy (PREV erved.
- **PLANMGMT(EXTENDED)** - A package has one active copy, and two additional prior co US and ORIGINAL) are preserved.

Therefore, if you REBIND using either BASIC or EXTENDED you have a set (or sets) of insurance access paths that can be called upon if you encounter degraded performance. To switch back to a previous access path you can REBIND using the SWITCH parameter, as follows:

- **SWITCH (PREVIOUS)** - changes the current and previous packages. The existing current package takes the place of the previous package; The existing previous package takes the place of the current package. This option works with both the BASIC and EXTENDED PLANMGMT options
- **SWITCH (ORIGINAL)** - clones the original copy to take the place of the current copy; The existing current copy replaces the previous copy; The existing previous copy is discarded. This works only with PLANMGMT(EXTENDED).

As of DB2 10 for z/OS, which just recently became generally available, there will be a new REBIND parameter named APRETAINDUP. This parameter is used in conjunction with EXTENDED or BASIC PLANMGMT. If set to NO old copies are not saved if the new access paths are identical to the old. This is a useful option because the only reason you would want to save a backup copy of access paths is if they were different, right?

IBM is also planning a few post-GA additions to DB2 10 for z/OS for access path management. The first is a nice new capability called access path reuse. It will be implemented using the APREUSE parameter (for both BIND and REBIND). If APREUSE(YES) is specified, DB2 will attempt the ability to reuse existing access paths. For a BIND, of course, this will only be attempted for queries that have not changed.

Another planned parameter is the APCOMPARE parameter for access path comparison. This will allow DB2 to raise a message when access paths change during a BIND or REBIND.

All of these newer options are useful, but they do not change The Five R’s best practice approach outlined above. The PLANMGMT option is a reactive one and can be useful for problem packages. Having a way to go back to an old access path can also be useful if it is missed during the fifth R (the review stage). And the new APCOMPARE and APREUSE DB2 10 features will help us in terms of comparing and reusing existing access paths, but they will not tell us whether changed access paths are good or bad.

The Bottom Line

DB2 shops should implement best practices by implementing an approach that conforms to The Five R's. The fifth R is the only step that requires additional tooling and/or manpower to accomplish. This involves testing access paths to compare the before and after impact of the optimizer's choices. Only by adopting such an approach will you be optimizing your DB2 application environment in a way that doesn't cause anxiety for the DBAs.

From IDUG Solutions Journal, Winter 2010.

DB2PORTAL.com