DB2 update

Managing DB2 Access Paths

by Craig S. Mullins

June 2007

The access paths formulated by the DB2 optimizer during the BIND process are critical to your application performance. It is these access paths that determine how and how efficiently DB2 data is read and delivered to your program. Whether you are preparing a new program, implementing changes into your existing DB2 applications, upgrading to a new version of DB2, or simply trying to achieve optimum performance for existing applications, an exhaustive and thorough BIND management process should be of paramount importance.

However, many organizations are not doing everything possible to keep access paths up-to-date with the current state of their data. Oh, of course there are excuses why we do not adhere religiously to the acknowledged "best practice" of REORG, RUNSTATS, and REBIND. In this article we will examine some of these issues and outline methods for introducing a better control into your access path generation process.

The Benefits of Change Control

In mainframe environments change is traditionally controlled and managed. For every change we make to application programs, we thoroughly test it before promoting it to the production environment. Depending on the site and the project we may have unit testing, QA testing, volume testing, integration testing, and so on. Programmers test every change to minimize the risk of unintended consequences. We do the same type of due diligence with most other changes in the mainframe world. Database changes are planned and thoroughly tested. Any and all system software (z/OS, CICS, IMS, etc.) changes, including PTFs to DB2 and upgrades, are subject to strict change control procedures. We do this to maximize availability because minor problems can disrupt the production work being conducted by the business.

But one exception to the tightly controlled change management environment of the mainframe has been DB2 access paths. We run BINDs and REBINDs in the production environments without the benefit of oversight or prior testing. Oh, we may try to mimic the production environment in test, but we cannot preview production access paths in the production environment. This lack of change control results in unpredictable performance impacts. Because programs are moved to production and bound there we are at the mercy of the DB2 optimizer, which generates access paths on the fly for our programs. Any issues with inefficient access paths are then dealt with in a reactive mode. That is, problems are addressed after the fact.

One of the biggest reasons for not implementing strict change control processes for access paths is the lack of built-in methods for access path change control. And manually evaluating thousands of packages and tens of thousands of SQL statements can be quite impractical. But there are things that can be done to help alleviate this problem.

BIND and REBIND Parameters

There are many parameters and values that must be chosen from and specified when you bind a DB2 application program. With so many options the whole process can be extremely confusing – especially if you don't BIND on a daily basis. And even if you do, some of the options still might be confusing if you rarely have to change them. You know what I'm talking about, parameters like ACQUIRE, RELEASE, VALIDATE, and DEGREE.

It is not the intent of this article to explain (pardon the pun) these parameters or to give you advice on which to use when. There are many articles and books, as well as the IBM DB2 manuals that you can use to guide you along that path. Suffice it to say, that there are some standard parameters and values that should be chosen "most of the time" in certain situations. As such, a wise DBA group will set up canned routines for the programmers to use for compiling and binding their applications. Choices such as: "CICS transaction", "DB2 batch", or "analytical query" can be presented to the developer and then, based on which of the various types of programs and environments that are available, the canned script can choose the proper BIND options. Doing so can greatly diminish the problems that can be encountered when the "wrong" parameters or values are chosen at bind time.

This same process can be put in place for production binding to ensure that the appropriate parameters and values are chosen. This is especially useful when the binds are not done by a DBA, but are automated in production or done by a less-experienced change control clerk.

Of course, there should always be a method for over-riding the "standard" values for special situations, although these overrides should not be available to anyone other than a well-trained individual (DBA or otherwise).

I want to make one small exception here regarding advice on bind parameters, and that is the EXPLAIN parameter. In production, always bind your plans and packages specifying EXPLAIN YES. Failing to do so means that access paths will be generated, but you will not know what they are. This is akin to blinding yourself to what DB2 is doing and is not advisable.

Approaches to Access Path Management

At this point no one should be arguing the point that BIND and REBIND are important components in assuring efficient DB2 applications. Because the BIND/REBIND process determines exactly how your DB2 data is accessed it is important that you develop an appropriate strategy for when and how to REBIND your programs.

There are several common REBIND approaches taken by DB2 users. By far, the best approach is to REBIND your applications over time as the data changes. This approach involves some form of regular maintenance that keeps DB2 statistics up to date and formulates new access paths as data volumes and patterns change.

Other approaches include REBINDing only when a new version of DB2 is installed, or perhaps more ambitious, whenever new PTFs are applied to DB2. Another approach is to REBIND automatically after a regular period of time (days, weeks, months, etc.). This approach can work if the period of time is wisely chosen based on the application data – but it still can pose administrative issues.

The final approach can be summarized as "if it ain't broke don't fix it!" This is the worst of the several approaches discussed here. The biggest problem with this approach is that you are penalizing ***every*** program in your subsystem for fear that a program or two may have a degraded access path. 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 is real – and that is why this approach has been adopted at some sites. The problem is being able to find which statements have degraded. In an ideal world we would be to be able to review the access path changes beforehand to determine if they are better or worse. But DB2 itself does not provide any systematic method of administering access paths that way. There are third party tools that can help you achieve this though.

Anyway, let's go back to the best approach again, and that is to REBIND on a regular basis as your data changes. This approach has become known as the three Rs. To implement this approach you:

1. Regularly reorganize the data to ensure that it is optimally structured.
2. Follow that with RUNSTATS to be sure that the reorganized state of the data is reflected in the DB2 Catalog.
3. And follow that with a REBIND for all the application programs that access the data structures impacted by the REORG and RUNSTATS.

At any rate, your goal should be to keep your access paths up-to-date with the current state of your data. Failing to do this means that DB2 is accessing data based upon false assumptions. DB2 is unlikely to make the same access path choice as your data grows – and as patterns within the data change.

By REBINDing you can generally improve the overall performance of your applications because the access paths will be better designed based on an accurate view of the data. Additionally, as DB2 changes are introduced (PTFs, new version/release) optimizer improvements and new access techniques can be incorporated into the access paths. That is, if you never REBIND, not only are you forgoing better access paths due to data changes but you are also forgoing better access paths due to changes to DB2 itself.

Adopting the Three R's approach can pose additional questions. For example, when should you reorganize? In order to properly determine when a REORG is needed you'll have to look at statistics. This means looking at either RUNSTATS or Real-Time Statistics (RTS). So, perhaps it should be at least 4 R's – in other words:

1. RUNSTATS or RTS
2. REORG
3. RUNSTATS
4. REBIND

Now it is true that some folks don't rely on statistics to schedule a REORG. Instead, they just build the JCL to REORG their database objects when they create the object. So they create a table space then build the REORG job and schedule it to run monthly, or quarterly, or on some regular basis. This is better than no REORG at all, but it is probably not the best approach because you are most likely either reorganizing too soon (in which case you waste the CPU cycles to do the REORG) or you are reorganizing too late (in which case performance is suffering for a period of time before the REORG runs). Better to base your REORGs off of statistics and thresholds using either RUNSTATS or RTS.

Without accurate statistics there is little hope that the optimizer will formulate the best access path to retrieve your data. If the optimizer doesn't have accurate information on the size, organization, and particulars of your data then it will be creating access paths based on either default or inaccurate statistics. Incorrect statistics will cause bad choices to be made – such as choosing a merge-scan join when a nested loop join would be better, or failure to invoke sequential prefetch, or using the wrong index – or no index at all. And the problem of inaccurate statistics is pervasive. There are shops out there that never, or rarely, run RUNSTATS to gather up-to-date statistics. Make sure yours is not one of those shops!

When should you run RUNSTATS? One answer is "As frequently as possible based on how often your data changes." To do this you will need to know a thing or two about your data growth patterns: what is its make-up, how is it used, how fast does it grow, and how often does it change? These patterns will differ for every table space in your system.

Next we need to decide when to REBIND? The best answer for this is when statistics have changed significantly enough to change access paths. When we know that data has significantly changed it makes sense to REBIND after the RUNSTATS completes. But the trick is determining exactly when we have a "significant" change in our data. Without an automated method of comparing and contrasting statistics (or even better yet, access paths) coming up with an answer in a manual way can be time-consuming and error-prone – especially if we have thousands of DB2 programs to manage.

As we REBIND, we always must be on alert for rogue access paths. A rogue access path is created when the optimizer formulates a new access path that performs worse than the previous access path. This can happen for a variety of reasons. Of course, number one is that the optimizer, though good, is not perfect. So mistakes can happen. Other factors can cause degraded access paths, too. The access paths for volatile tables depend on when you run the RUNSTATS. Volatile tables are those that start out empty, get rows added to them during processing, and are emptied out at the end of the day. And, of course, if the catalog or statistics are not accurate we can get problems, too.

So adopting the Four R's approach implies that you will have to develop a methodology for reviewing your access paths and taking care of any "potential" problem access paths. Indeed, the Four R's becomes the Five R's as we add a step to review the access paths after REBINDing to make sure that there are no rogue access paths:

1. Start with a RUNSTATS (or use RTS) to determine when to REORG.
2. Reorganize the table spaces (and indexes)
3. After reorganizing, run RUNSTATS again,
4. Follow that with the REBINDs.
5. Then we need that fifth R – which is to review the access paths generated by the REBIND.

The review is of utmost importance because the optimizer can make mistakes. And, of course, so can you. But your users will not call you when performance is better (or the same). They only dial your numbers when performance gets worse. As such, proactive shops will put best practices in place to test REBIND results comparing the before and after impact of the optimizer's choices.

The Plan Tables

A lot of information is contained in the PLAN_TABLE. After the optimizer creates the access paths and populates the PLAN_TABLE with data representing those access paths, we need to examine the results to determine if everything is satisfactory.

Many questions can be answered by analyzing the results of EXPLAIN – questions like:

- if we are joining what type of join is used (NLJ, MS, Hybrid),
- was an index used, and if so how many columns matched,
- are we doing a scan, and if so what type of scan (full or page range)
- is prefetch being used, and if so what type (sequential, list)
- was a hint used
- was parallelism used, and if so what degree and type (I/O, CPU, Sysplex)
- was a sort required, and if so why (Join, Unique, Group By, Order By)
- what type of locking is required

And that just covers the main PLAN_TABLE. The EXPLAIN option also populates two optional tables, if they exist:
- DSN_STATEMNT_TABLE which contains DB2's estimate of the processing cost for an SQL statement
- DSN_FUNCTION_TABLE which contains information about function resolution.

Of course, for any of this information to be returned you have to have bound specifying EXPLAIN(YES). Any change to any of these items between Rebinds means a change in access path – which can be positive, or a potential problem. Over time, performance analysts can determine which changes are good and which might be problematic – but it takes experience (and perhaps some luck) to do this correctly. Using a tool that automates the process can also make the task much easier and more accurate.

So, how do you determine what access paths have changed? Sometimes the program has changed, too – which can make it challenging to find the exact SQL statements to compare. When just the access paths change it will be easier to compare them and spot the changes, but there is still a wealth of data that needs to be analyzed to do this justice.

And when you are talking about thousands of programs being rebound, do you really have the time to review every access path to make sure it is fine? This question alone causes many folks to go back to the "Let It Ride" mentality – which is too bad, because it is an inferior approach, especially when there are products that can help.

Version Migration Issues

Let's talk, for a moment, about an event that some shops are still facing, namely migrating from DB2 V7 to V8. And if you are already happily running on V8, remember that V9 is now generally available so you will soon be planning a similar (though not as dramatic) migration.

First of all, let's be clear, you do ***not*** have to REBIND all of your packages and plans when you move to V8. But it ***is*** a really good idea to do so, and most of you will probably REBIND your applications when you get to V8. Why?

First of all, there are optimizer and performance improvements that you will not get without a REBIND. And there will be degraded program performance that will occur when you get to V8 that REBIND can rectify. And for some of you, there will even be REBINDs that you just will not be able to avoid. Let's examine each of these issues briefly.

First of all, what is the "degraded performance" issue? The problem occurs when DB2 turns off fast column processing. DB2 V3 introduced a function called an SPROC. An SPROC, or SELECT procedure, enables fast column processing. Essentially, this enhancement examines SELECT statements that are executed repeatedly and builds an internal procedure that moves all the columns in one move rather than one column at a time. You have no external control over when or if DB2 uses them. And the more columns that are specified on a SELECT, the greater the performance gain could be.

How does this all tie into V8? If a plan or package is using an SPROC in V7, the SPROC is using 31 bit code. When you attempt to run that same plan or package in V8 without REBINDing it first, it needs to be in 64 bit. It isn't, so DB2 disables the procedure. The only way you can re-enable the SELECT procedure is by REBINDing the program. Until you do that REBIND, and if the plan or package uses an SPROC, your application's performance will be degraded. When you REBIND performance will improve. Along those lines, the IBM redbook titled “DB2 UDB for z/OS Version 8 Performance Topics” specifically warns of this problem, cites the potential for CPU increases of up to 10% and recommends global REBINDs.

What about those REBINDs that cannot be avoided? DB2 V8 will autobind any plans and packages that were bound prior to DB2 Version 2 Release 3. So you might experience an execution delay the first time such plans are loaded unless you REBIND them yourself. And DB2 might change the access path due to the autobind, potentially resulting in a more efficient access path – or a more inefficient access path.

And such actions are likely to become more common in future DB2 versions. In several conference presentations, folks at IBM have suggested that in the future DB2 may autobind any plan or package that was last bound on an “out of service” version of DB2. Today, only V7 and V8 are in service, so think about that when you are considering your REBIND approach.

There are additional reasons to REBIND when moving to V8. DB2 V8 in NFM uses a different format for its DBDs, packages and plans. Before it can use a DBD, plan or package from an older DB2, it must first be expanded to the new V8 format. This causes more overhead. What should you do? Here is the advice right out of the afore-mentioned redbook:

> *After you have entered new-function mode, we recommend that you plan to rebind all of your plans and packages. DB2 will then store the plans and packages in the DB2 catalog in the new format. DB2 will no longer need to expand the plans/packages each time it needs to use them.*

Summary

Forward-thinking organizations should adopt a liberal approach to REBINDing their applications to ensure optimal access paths based on up to date statistics. Keeping DB2 updated on data changes and making sure that your programs are optimized for the current state of the data is the best approach. This means regular executions of RUNSTATS, REORG, and REBIND. And if you are worried about rogue access paths, consider investing in a third party tool that can assist with access path changes management issues.

Failing to keep your access paths aligned with your data is a sure recipe for declining DB2 application performance.

From DB2 Update, June 2007.