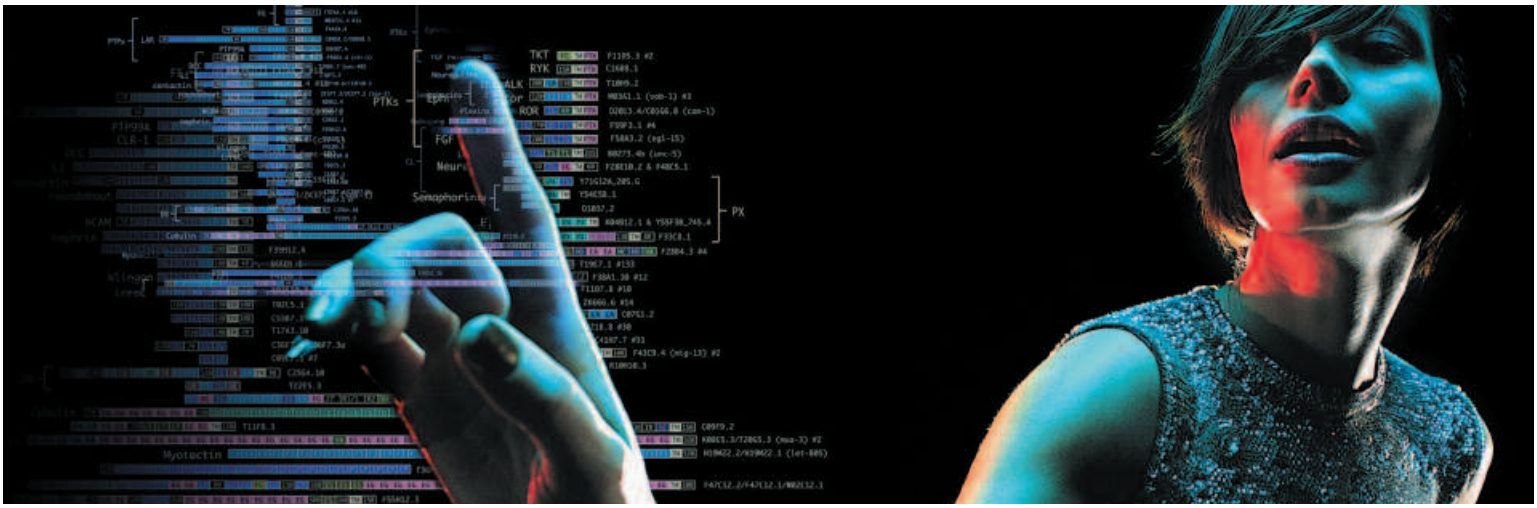


Better DB2 Performance and Availability:

How Change Control Over Rebinding Pays Dividends

Craig S. Mullins,
Corporate Technologist, *NEON Enterprise Software, Inc.*

white paper

NEON
Enterprise Software, Inc.

Table of Contents

white paper

Overview

"Change is inevitable. Change is constant." That's what nineteenth century English Prime Minister, Benjamin Disraeli said. If he was a DBA or an IT manager in charge of a large corporate data center, he'd probably have added, "Change is a constant headache."

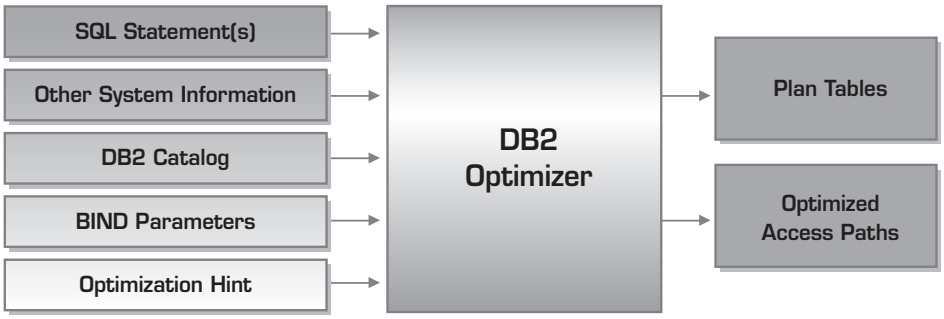
Change is unquestionably a constant in any large data center. But change also constantly causes unexpected problems that ripple through systems, databases and applications. A change in one piece of code or in one database can degrade performance elsewhere and can impact data availability.

IT professionals understand the chaotic risks that change incurs, and few shops today operate without change control mechanisms integrated into their management processes. Changes are planned and evaluated, risks assessed, and changes implemented in a controlled manner. Any changes to applications or database schemas are rigorously tested and retested before being rolled out to a production environment.

In the mainframe database environment especially, change is strictly controlled. But one exception has been DB2 access paths, which can change with every bind and rebind operation.

Binds and rebinds are typically done in the production environments without the benefit of oversight or prior testing. This lack of change control results in unpredictable performance impacts. In most shops, programs are moved to production and bound there. This puts DBAs at the mercy of the DB2 optimizer, which generates access paths on the fly as it binds or rebinds programs. Any issues with inefficient access paths are then dealt with in a reactive mode. Problems are addressed after the fact.

Bind and rebind are at the heart of DB2 performance. The bind process is where the DB2 optimizer decides how to process the SQL in your application programs (as shown below).



One of the biggest reasons for not implementing strict change control processes for access paths is the lack of built-in methods for ensuring access path change control discipline. Let's face it, 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. In this paper, we'll describe BIND parameters, approaches to access path management, and change control over access paths during DB2 version migration.

white paper

BIND Parameters

There are many parameters and values that must be specified when you bind a DB2 application program. The vast array of options can render the whole process 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. This is the case with parameters like ACQUIRE, RELEASE, VALIDATE, and DEGREE.

It is not the intent of this paper to delve into the myriad bind options nor 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 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).

One small exception to the advice on bind parameters 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

Bind and rebind are important components in assuring optimal application performance. It is the bind process that determines exactly how your DB2 data is accessed in your application programs. As such, it is critically important to develop an appropriate strategy for when and how to rebind programs.

There are several common 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 binding 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, whether it is days, weeks, months, or whatever period of time you deem significant. This approach can work if the period of time is wisely chosen based on the application data—but it still can pose significant administrative issues.

The final approach is from the "if it ain't broke don't fix it" school of thought. This approach 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 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 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 isolate statements that might be worse. The ideal situation would be to be able to review the access path changes before hand to determine if they are better or worse. But DB2 itself does not provide any systematic method of administering access paths that way. Of course, you can remove this impediment by using NEON Enterprise Software's Bind ImpactExpert to manage the access paths in your DB2 programs.

Let's talk more about regular rebinds as your data changes. This involves what has become known as the three R's. This means regularly reorganizing the data to ensure that it is optimally structured. That is followed by RUNSTATS to be sure that the reorganized state of the data is reflected in the DB2 Catalog. Finally, we follow that up with rebinds of the application programs that access the data structures that have been reorganized and RUNSTATEd (if you'll allow me to turn that into a verb).

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 made (via new releases or PTFs) optimizer improvements and new access techniques can be incorporated into the access paths. 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.

Of course, 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:

- RUNSTATS or RTS
- REORG
- RUNSTATS
- REBIND

Some users 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. A tool like NEON Enterprise Software's RealTime DBAExpert can be used to automate this process based on your environment and the thresholds you set.

Statistics are the fuel that makes the optimizer function properly. 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 probably 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 that never, or rarely, run RUNSTATS to gather up-to-date statistics. Make sure yours is not one of those shops!

How often should RUNSTATS be run? One answer is "As frequently as possible based on how often your data changes." This means that you will need to know a thing or two about your data growth patterns. To properly determine a schedule for statistics you need to know things about your data: 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.

So, let's ask the $10,000 question—when should rebinds be done? 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 when we get into the thousands of programs.

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. This can happen for a variety of reasons. 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 at the end of the day. And, of course, if the catalog or statistics are not accurate we can get problems, too.

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. Tackling this can be a difficult mountain to climb.

Indeed, the Four R's probably needs to become the Five R's because we need to review the access paths after rebinding to make sure that there are no rogue access paths. So, we start off with a RUNSTATS (or use RTS) 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. As we mentioned, the optimizer can make mistakes. And, of course, so can you. Users don't call you when performance is better (or the same). But if performance gets worse, you can bet on getting a call from irate users.

So we need to put in place best practices whereby we test bind results to compare the before and after impact of the optimizer's choices.

Version Migration Issues

An event that strikes dread in the heart of many DBAs is migrating to a major new DB2 version. This year, that issue revolves around migrating to DB2 Version 9. Each migration involves access path changes.

You do not need to rebind all packages and plans when you upgrade your DB2 version, but it is a good idea to do so. Why?

First of all, there are optimizer and performance improvements that you won't get without a rebind. And there may be degraded program performance with your old paths that rebind can fix. Sometimes, there will even be REBINDs that you just will not be able to avoid. DB2 will autobind in some cases, and while it will often change the access path to a more efficient access path, it could change it to a more inefficient path.

Forward-thinking organizations should adopt a liberal bind and rebind process to ensure optimal access paths based on up to date statistics. Keeping abreast of 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. If you are worried about rogue access paths, consider investing in a third party tool, such as Bind ImpactExpert?, that can assist with access path change management issues.

Failing to keep your access paths aligned with your data is a sure recipe for degraded DB2 application performance. Using a tool to analyze access path changes and intelligently apply them can keep access paths optimized and improve performance.

What About the New PLANMGMT Feature?

No discussion of DB2 access path management would be complete and up-to-date without at least mentioning the new PLANMGMT capability. The feature was originally called plan stability, but because it only works on packages, people have started calling it access path stability or package stability. Its name notwithstanding, you can use PLANMGMT to keep backups versions of your program's access paths.

With plan stability you can fall back to a previous version of your package's access paths. Of course, you will need to have bound using the PLANMGMT feature in order to do this. There are three options than can be chosen for PLANMGMT:

- PLANMGMT(OFF) - No change to existing behavior. A package continues to have one active copy.
- PLANMGMT(BASIC) - A package has one active copy. One additional prior copy (PREVIOUS) is preserved.
- PLANMGMT(EXTENDED) - A package has one active copy, and two additional prior copies (PREVIOUS and ORIGINAL) are preserved.

If you bound using either BASIC or EXTENDED, the next time you REBIND you can fall back to an earlier "version" of that program's access paths if performance suffers due to rogue access paths.

This might seem to be a solution to the problems discussed in this white paper, but it is not really. First of all, if you choose to bind all of your packages using PLANMGMT, you will double or triple the amount of storage required in the DB2 Directory for your programs. And probably more importantly, this is a reactive solution. That is, you can fall back, but only after you have experienced a problem. In order to fall back, you must do another rebind using the SWITCH option, which has 30% CPU overhead.

Isn't there a better solution? *Well, yes, there is!*

Bind ImpactExpert as a Change Control to Intelligently Optimize Access Paths

Bind ImpactExpert provides control of DB2 bind and rebind processes. Intelligent and functionally rich, Bind ImpactExpert automates analysis and management of DB2 binds and rebinds, pinpointing binds and suppressing rebinds that would degrade application performance across the production environment.

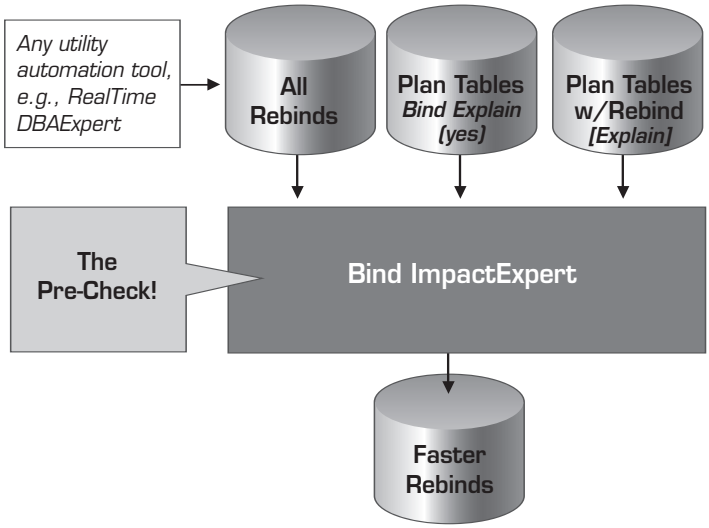
By identifying performance-degrading binds and rebinds, Bind ImpactExpert ensures optimal application performance, but more importantly from a change control perspective, it prevents the surprise performance degradations that sometimes happen with routine rebinds.

Using sophisticated analysis of bind and rebind impacts, Bind ImpactExpert will automatically:

- Prevent rebinds that would result in access paths that degrade or have no improvement on performance.
- Identify and generate the statements for rebinds that result in access paths that improve performance.
- Integrate into the change control environment for binds to automatically pinpoint changed SQL statements and determine access path degradation that will result.
- Help optimize applications by identifying access path changes that degrade performance, so the SQL can be tuned. It will identify the actual SQL.
- Precheck the impact of a new DB2 version before migration without rebinding.
- Prevents binds where there is no change in the SQL for the changed application and a bind is not necessary.

In most DB2 environments, rebinds are an all or nothing proposition. They are either always done or never done. Both approaches can result in degraded performance.

A better approach is to evaluate the impact of each rebind to determine if the access path is better or worse. Will performance be enhanced or degraded? Most rebinds do improve performance, but some can dramatically degrade it. Bind ImpactExpert not only prevents rebinds that would degrade application performance, it also prevents unnecessary rebinds, reducing CPU and system resource usage and avoiding possible contention and lock issues. Bind ImpactExpert guarantees rebinds that improve performance.



Using an automated, intelligent approach to DB2 bind and rebind processes, you can bring the rigor of change control processes to binds and rebinds and ensure that only positive impacts to your access paths are implemented. Learn more about how Bind ImpactExpert does this by visiting the product web page: www.neonesoft.com/BAI.shtm

About NEON Enterprise Software

NEON Enterprise Software is a technology leader in enterprise data management software and services. As the rules of business change, our solutions let you efficiently control, protect, retain and manage data to comply with today's business and legal requirements. Founded in 1995, NEON Enterprise Software serves customers worldwide with its team of dedicated industry experts.

For more information about NEON Enterprise Software, visit www.neonesoft.com or call 281.491.6366 or 888.338.6366.

white paper