Indexes Are Key to Application Performance and Response Times

Manage NPIs in IBM® Db2® for z/OS® with BMC AMI Data for Db2® to improve performance and increase savings

Prepared by Craig S. Mullins, President and Principal Consultant of Mullins Consulting, Inc.

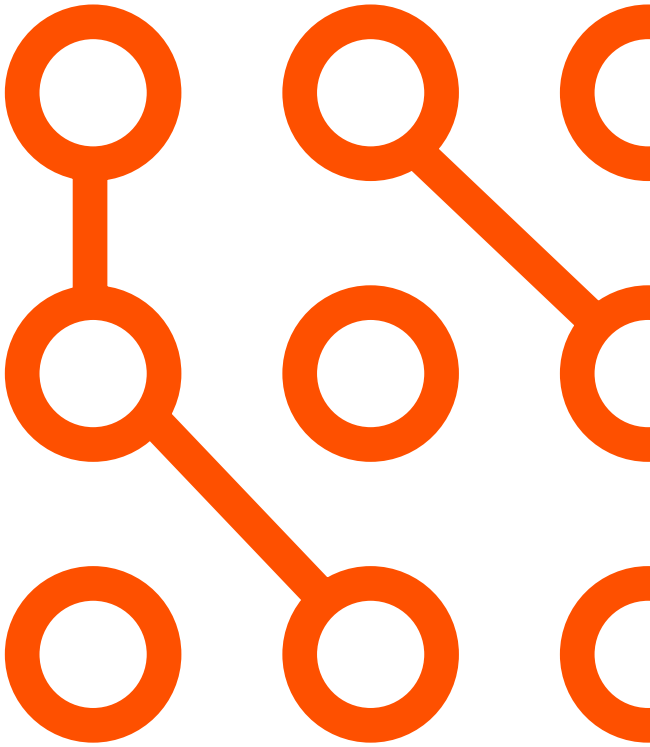
bmc

Table of Contents

Executive Summary

Today's Db2 database applications and systems must accommodate an ever-growing amount of data, while being accessed more rapidly and from more sources than ever before—and all without any prolonged downtime. This reality is stretching the ability of database administrators to be able to manage their Db2 database structures using traditional tools and utilities.

Indexes are the key to application performance and therefore customer response time. However, difficulties managing these NPIs create conflicts between Db2 administrators and customer applications. One of the more difficult aspects of modern Db2 administration is managing non-partitioned indexes (NPIs) on tables in large partitioned table spaces. This white paper examines the existing situation and provides guidance on how to overcome the difficulties in managing NPIs in your Db2 for z/OS databases and applications.



Database Growth

The amount of data that organizations acquire, store, and manage continues to grow unabated. According to a study conducted by IDC[i], the size of the digital universe doubles every two years and will continue to do so until at least 2020. That means by 2020, the amount of data that we collectively must manage will grow in size to 40,000 exabytes, or 40 trillion gigabytes; more than 5,200 gigabytes for every person on the planet.

Many factors are driving this growth. For example, the desire to make better decisions based on machine learning and analytics applied to big data sets is driving up the amount of data that is being created and stored. Furthermore, additional industry trends, such as mobile computing and social media also drive up data creation and usage. And as we move to connect any device with an on/off switch to the internet, the Internet of Things (IOT) will contribute even more to the generation of additional data.

At the same time, the cost of storing data is decreasing rapidly. The cost to store a megabyte (MB) of data was $1 million in 1956, but today it costs less than $0.05 per gigabyte (GB)[ii]. We seem to have no trouble filling it all up, and with storage being nearly free and new data techniques promising advanced ROI, there is no real desire for most organizations to stop collecting more and more data.

What about Db2?

All Db2 data resides in a table space. The table space defines the storage and other physical characteristics for the data. Each table space can contain one or more tables (depending upon the type of table space). The table defines the actual data elements and constraints for the data. As of today (Db2 12 for z/OS), there are four types of Db2 table spaces from which to choose:

1. Segmented table spaces
2. Universal partition-by-growth (PBG) table spaces
3. Universal partition-by-range (PBR) table spaces
4. Classic partitioned table space

The largest tables in your Db2 environment will be partitioned in either universal PBR table spaces or classic partitioned table spaces. Whereas segmented table spaces have a maximum size of 64 GB, classic partitioned table spaces can have up to 4,096 partitions of up to 64 GB each. Standard universal table spaces can grow up to 128 terabytes (TB) and with the relative page number (RPN) feature of Db2 12, universal PBR table spaces can accommodate up to 1 TB for each partition, resulting in a possible maximum size of 4 petabytes (PB) containing up to 256 trillion rows per table.

The bottom line is that partitioning your data is a fact of life when using Db2 to store large amounts of data.

i IDC's Digital Universe Study, sponsored by EMC. https://www.emc.com/leadership/digital-universe/index.htm
ii Which means that the cost of a GB of data in 1956, if it had even been available, would cost $1 billion!

Partitioning and Indexing

If you want to get fast access to your Db2 data, the best thing you can do is create appropriate indexes that match your data access patterns. Of course, there are other things needed to assure optimal Db2 performance, but building proper indexes should be at the top of the list when it comes to improving performance. For example, consider a customer transaction table without an index. There can be millions, or even billions of rows in such a table. Without an index, you would have to read all of the rows in the table, when you may only be seeking one or a few rows. An index limits the rows to be searched to only those that are required.

It is a relatively simple matter to create indexes on tables that are not partitioned. All of the data resides in one structure and the index is built on all of the data in that one structure. But management difficulties can arise when creating multiple indexes on partitioned table spaces. To understand why this is so, let's examine the different types of indexes that can be created on a partitioned table space.

- **Partitioned index:** A partitioned index is an index that is physically partitioned into separate data sets. Each data set corresponds to a table partition; that is, the data of partition 1 in the index will match the data of partition 1 in the table (space).

- **Partitioning index:** A variation of the partitioned index is the partitioning index, which refers to an index defined on the column(s) used to partition the data. Whenever the index key has the same high order columns and collating sequence as the columns in the partitioning key, it is considered a partitioning index.

- **Data-partitioned secondary index, or DPSI:** A DPSI is partitioned, but is not a partitioning index. This means its key is different than the limit key upon which the table space partitioning is based. But it is physically partitioned according to the partitioning scheme of the underlying data. Figure 1 shows an example of a DPSI created on a table in a table space with three partitions.

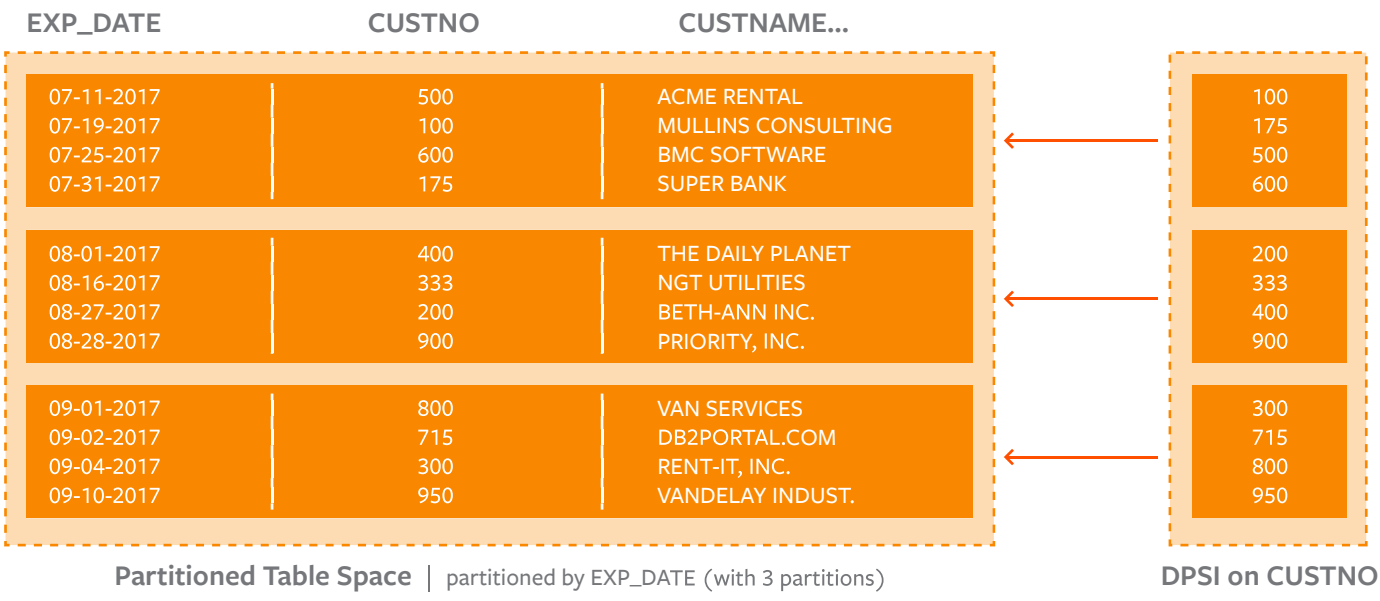


Figure 1. Data-Partitioned Secondary Index (DPSI)

DPSIs make it easier to perform maintenance tasks—such as REORG, COPY, and LOAD—at the partition level on partitioned table spaces. This is because the data in each DPSI partition aligns with the data of the same partition of the table space.

However, DPSIs can be at a performance disadvantage when compared with non-partitioned indexes (NPIs). Each partition of the DPSI contains its own index structure (a separate b-tree). Therefore, when a query uses a DPSI to access data in more than one partition, Db2 must probe each partition of the index to satisfy the predicate (and there can be up to 4,096 partitions, meaning that 4,096 index b-trees may need to be probed). This will not be as efficient as scanning a single b-tree (such as with an NPI). For this reason, DPSIs are not very widely implemented. With DPSIs, multiple probes usually are required. It all depends upon your specific usage scenarios, but the more common scenario is queries requiring data across partitions, and this is where DPSIs fail to deliver.

- **The non-partitioned index, or NPI:** An NPI is the final type of index you can create on partitioned Db2 data. With an NPI, all of the indexed data is in a single partition regardless of how many partitions are used to store the table. NPIs are the most common type of index on partitioned data and they create the most trouble in terms of maintenance and performance management. Figure 2 shows an example of an NPI created on the same table as in Figure 1 (three partitions).

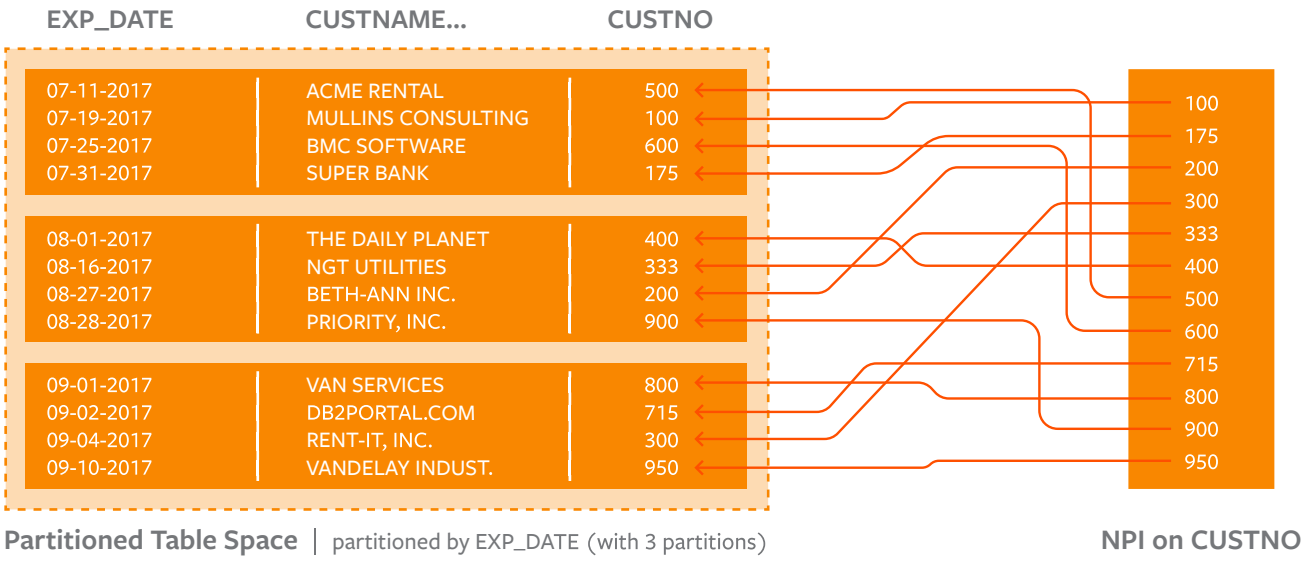


Figure 2. Non-Partitioned Index (NPI)

The Trouble with NPIs

Before discussing the problems that NPIs create, let's first acknowledge their undoubted performance benefit. NPIs are versatile and useful for most types of Db2 queries. To achieve optimal performance, the query does not have to contain predicates on the partitioning columns of the table to limit the query to a small subset of the partitions (as with DPSIs). A single b-tree is needed to store the entire NPI, so multiple scans are not required. This means that NPIs better fulfill the general requirement of indexing—to improve query performance—than do DPSIs.

Another reason why NPIs are more common than DPSIs is that NPIs have been supported longer than DPSIs. NPIs have been around for as long as partitioned table spaces, whereas DPSIs were introduced in Db2 version 8.

The basic state-of-affairs in most Db2 shops is that many NPIs have been created and need to be supported. Without NPIs, query performance suffers and that usually is not an acceptable tradeoff.

If we accept that NPIs improve performance are common, what is the problem?

NPIs can cause contention, particularly with Db2 utilities. Although it is possible to run a utility against a single table space or index partition, doing so when NPIs exist on the table causes problems because these indexes are not partitioned. Running utilities against NPIs can impact the availability of an entire table space because the NPI contains data for the entire table (space), not just for a single partition. Changing data in a single partition—whether for recovery, load, or reorganization purposes—requires modifying the entire NPI because the affected data is spread throughout the single index structure.

For example, assume that you have a large table space with 24 partitions. You may have a partitioning index or not, but you also have at least one NPI. Over time, as data is deleted, updated and inserted into the table, the table space and index(es) will become disorganized. At some point, when the data becomes so disorganized as to negatively impact performance, the REORG utility should be run to reorganize the table space and indexes to a more efficient state.

Because the table space is large, it may be difficult to reorganize the entire thing, so the DBA decides to reorganize one partition at a time. The general idea of such an approach is to reduce the amount of time and processing required to reorganize the data; a partition is much smaller than the entire table space, so less time and effort should be required to reorganize it.

The REORG utility, as part of its basic functionality, moves rows from one location to another to resolve things like clustering, indirect references, and free space. When a partition of data is reorganized, the entire NPI must be reorganized to ensure that the index entries point to the proper, new data locations. Furthermore, NPIs are always rebuilt entirely even when the partition being reorganized is empty.

Sorting large amounts of data is necessary when reorganizing an entire NPI, and can take a significant amount of CPU and time to process. Some NPIs can simply be too big to process, effectively rendering them un-reorganizable at least with standard utilities (i.e., the utility fails before it can complete the job because it runs out of resources). The larger the size of the NPI, the greater the chance that a REORG will fail, frequently during the SORT phase. Additionally, contention on NPIs can cause performance bottlenecks during parallel update, insert, and delete operations.

Data sharing issues also can arise with NPIs. Your NPIs become group buffer pool (GBP) dependent when they are being built concurrently from multiple subsystems (e.g., using concurrent LOADs). This dependency can negatively impact the performance of the LOAD jobs, possibly eliminating any performance advantages of running the utilities through multiple subsystems.

Furthermore, the NPIs may not be built efficiently and therefore may not provide the best performance when they are used after the completion of the LOAD jobs.

Addressing NPI Challenges

The issues noted previously have caused some organizations to take drastic measures to accommodate NPIs. Example of techniques to minimize NPI issues include denormalization to enable multiple access paths, writing multiple programs with different access paths, maintaining a table that indicates when a non-NPI access path should be used, and enforcing application slowdowns during maintenance windows.

Some organizations drop NPIs before loading across members and rebuild them later. Others have minimized the creation of NPIs or even eliminated them altogether, forcing access via a partitioning or partitioned index. This is probably the most troubling aspect of the NPI manageability issue: not building the indexes needed to optimize query and application performance, but compromising that performance to accommodate shortcomings in the tools being used. When you take this approach, your application programs and queries will not be running as efficiently as possible, impacting the experience of your end users and reducing the ROI of your business applications.

In reality, none of these solutions are ideal. The best approach would be to create all of the NPIs needed to optimize your Db2 applications and be comfortable that you can manage them effectively as needed.

BMC AMI Data for Db2® Tools and Utilities

Most of the tools and utilities for managing Db2 for z/OS originally were built decades ago. Things have changed dramatically in IT and database systems since the 1980s. This means that decades-old technology has been modified and tweaked over the years to keep up with new needs and capabilities. This approach can work for a period of time, and maybe longer for some types of products. But the demands of the digital and mobile age, including rampant data growth and big data requirements, have made it impossible for traditional Db2 utilities to work on the large table spaces and NPIs that are becoming more and more common across the industry.

Fortunately, BMC AMI Utilities for Db2® provides true 24x7 utilities with zero outages, designed for the largest data sets and highest transaction volumes of today's digital enterprise. Made to work in concert across your Db2 environment, BMC AMI Utilities for Db2® handles a wide range of data management tasks more efficiently without taxing Db2 resources. In addition, the utilities offer full availability, even while performing maintenance reorganization. And with a centralized architecture that provides intelligent policy-driven automation, combined with innovative processing methods, DBAs can easily manage increasing amounts of data with zero downtime and high integrity to meet new digital demands.

A hallmark of the BMC AMI Utilities for Db2® is the no-sort REORG, which completely eliminates the need to sort data. Using customized algorithms based on the data being processed, BMC AMI Utilities for Db2® dramatically reduce both CPU and DASD usage, enabling larger objects to be processed using only a fraction of the resources previously required. BMC AMI Utilities for Db2® eliminate the long outages you have become accustomed to when running Db2 utilities, helping you to deliver around-the-clock availability for your Db2 applications.

This means that you can build NPIs that support the processing needs of your Db2 applications without worrying about how or whether you can manage them later. Furthermore, these tools allow you to proactively manage Db2 for optimized costs and peak performing applications.

Conclusion

Don't remain stuck in the past with your Db2 environment. Embrace the next generation with BMC AMI Utilities for Db2®. Stop worrying about how you're going to manage your Db2 structures and begin concentrating on delivering quality and service to your end users. Plus, ensure that you can manage your Db2 environment, no matter how big it grows!

About The Author

Craig S. Mullins is a data management strategist, researcher, and consultant. He is president and principal consultant of Mullins Consulting, Inc. and has been named by IBM as a Gold Consultant and an IBM Champion for Analytics. He was also named one of the Top 200 Thought Leaders in Big Data & Analytics by AnalyticsWeek magazine. Craig has over three decades of experience in all facets of database systems development and has worked with Db2 since V1.

For more information

To learn more about BMC AMI Utilities for Db2®
visit bmc.com/it-solutions/bmc-ami-utilities-db2.html

About BMC
From core to cloud to edge, BMC delivers the software and services that enable over 10,000 global customers, including 84% of the Forbes Global 100, to thrive in their ongoing evolution to an Autonomous Digital Enterprise.

BMC—Run and Reinvent **www.bmc.com**