Have You Considered Reorganizing Your Indexes More Frequently?

By Craig S. Mullins

Mullins Database Research
A division of Mullins Consulting, Inc.
15 Coventry Court
Sugar Land, TX 77479
tel 281-494-6153
www.craigsmullins.com

Reorganization is a fact-of-life for DB2 database administrators. But practical, appropriate reorganization best practices are not necessarily practiced by many organizations. This paper will address the topic of reorganization in general, but more specifically it will tackle the need to expand the frequency of index reorganization. The general theme is this: increasing the frequency of reorganizing indexes can improve the performance of your DB2 databases and applications.

For those of you who are not overly familiar with reorganization, it is the process of organizing data in DB2 table spaces and indexes to improve the efficiency of access to those objects. Reorganization is required periodically to ensure that the data is situated in an optimal fashion for access. The reorganization process re-clusters data, resets free space, removes pseudo-deleted data, and generally works to locate the physical data as efficiently as possible in the table space or index in questions. Reorganizing database objects can even delete and redefine the underlying VSAM data sets for STOGROUP-defined table spaces and indexes.

Proper planning and scheduling of reorganizations can be a complex subject to master. Many factors influence whether or not reorganization is recommended. But we will attempt to simplify the topic by examining the statistics and decision points for implementing efficient reorganization policies in your DB2 environment.

The first step toward implementing a best practices approach to DB2 reorganization is to adopt a metrics and thresholds-based approach to reorganization instead of an object-based approach. Unfortunately, the object-based approach is common in many DB2 shops. With this approach, database administrators create a new reorganization job for each table space that is created. The job is then scheduled to run monthly (or maybe quarterly) regardless of need. Table spaces and indexes are reorganized at the same time, too. This approach is better than nothing, but I can guarantee that it is not the best use of your time and computing resources. You will invariably be reorganizing either “too soon” or “too late” with this approach. Reorganizing “too soon” means that you are consuming precious CPU resources to run a reorganization task that is not needed... and reorganizing “too late” means that you are reorganizing long after it would have been beneficial. This too wastes resources because it causes all of your SQL requests to the disorganized tables to be inefficient.

Furthermore, waiting to reorganize indexes only until the table space is reorganized is not a wise course of action. The table space and any indexes on tables in that table space are separate DB2 database objects living in separate data sets and having different levels of disorganization. Simply sweeping them all together into a single job with no knowledge of their current structure, is ill-advised and definitely not a best practice.

Reorganizing indexes more frequently also can be beneficial because indexes tend to be smaller than table spaces and therefore, they can be reorganized more quickly. Sometimes, reorganizing an index can provide a sufficient performance boost at minimal cost, thereby delivering significant return on your investment.

A best practice approach to reorganizing your DB2 table spaces and indexes requires metrics. The metrics that are gathered by DB2 in the form of database statistics that are stored in the DB2 Catalog. You can use the RUNSTATS utility to gather statistics or, better yet, you can utilize Real Time Statistics.

RUNSTATS and Real Time Statistics

Before you can move to a metrics-based reorganization approach, metrics are needed. These can be gathered using RUNSTATS or simply queried from the Real Time Statistics tables.

The RUNSTATS utility collects statistical information for DB2 tables, table spaces, partitions, indexes, and columns. It can place this information into DB2 Catalog tables or simply produce a report of the statistical information. The statistics in these tables are used for two primary reasons: to provide organizational information for DBAs and to be used as input to the DB2 optimizer during the BIND process to determine optimal access paths for SQL queries. The statistical information can also be queried using SQL.

Real Time Statistics (RTS) can be used to improve autonomic administration of your DB2 environment. RTS provides functionality that maintains statistics about DB2 databases “on the fly,” without having to run a utility program.

Prior to the introduction of RTS, the only way to gather statistics about DB2 database structures was by running a RUNSTATS utility. RTS, on the other hand, runs in the background and automatically updates statistics in two DB2 Catalog tables as the data in DB2 databases is modified. Where RUNSTATS is a hands-on administrative process, RTS is hands-off.

DB2 always gathers real-time statistics in memory for each table space, index space, and partition in your environment. The Real Time Statistics data is stored in two DB2 Catalog tables:

- SYSIBM.TABLESPACESTATS—Contains statistics on table spaces and table space partitions
- SYSIBM.INDEXSPACESTATS—Contains statistics on index spaces and index space partitions

DB2 externalizes real-time statistics to the tables at the following times:

- When you issue -STOP DB2 MODE(QUIESCE), DB2 first externalizes all RTS values. Of course, if you stop using MODE(FORCE), no RTS values are externalized; instead, they are lost when DB2 comes down.
- As specified by the DSNZPARM STATSINT value. The default is every 30 minutes. You can adjust this parameter to collect RTS values more frequently or less frequently, as desired.
- During REORG, REBUILD INDEX, COPY, and LOAD REPLACE utility operations, DB2 externalizes the appropriate RTS values affected by running that utility.

Reorganization Guidelines

You should develop rigorous in-house standards for reorganizing your DB2 database objects because doing so is one of the most significant aids in achieving optimal DB2 performance. We have already suggested two general guidelines for adoption:

1) Implement metrics-based reorganization thresholds based on DB2 statistics.
2) Adopt reorganization thresholds for indexes as well as table spaces instead of just relying on reorganizing indexes as the same time as the table space.

The frequency of reorganization is likely to differ for every DB2 application. By driving your reorganization schedule based on thresholds you can take into account important criteria such as:

- The frequency of data modification (insertions, updates, and deletions)
- Application transaction volume
- Amount of free space allocated when the table space or index was created

The next step is to adopt reasonable thresholds for running your reorganization jobs. We will establish some guidelines for that in the next section. Keep in mind though, that these are general recommendations and will not apply for every DB2 implementation, or for every application within your shop. There are no hard and fast rules that always apply in terms of when to reorganize table spaces or indexes; that said, a firm understanding of the rules of thumb will help in understanding data disorganization.

Furthermore, the following guidelines can be used as a basis for establishing thresholds for reorganizing your DB2 indexes. Based on actual observations you can adjust the thresholds accordingly, if required, to your local requirements.

Reorganizing Index Spaces

Now that we know about the methods of obtaining statistics for reorganization, let's delve into the appropriate statistics to examine to make our reorganization decisions. Although reorganizing DB2 table spaces is important, we will devote most of our discussion here to index reorganization issues and considerations.

We will take a look at both the RUNSTATS and RTS statistics upon which you can establish thresholds for kicking off a reorganization task. Although we examine both, the strong recommendation is to move to the RTS statistics because they require no additional manual steps to collect and they should be more accurate and up-to-date since they are automatically collected as DB2 operates.

First, let's look at the RTS values that can be examined for index reorganization. If there have been a lot of INSERTs and DELETEs to an index since the last time it was reorganized you should consider reorganizing the index. As data is added to and removed from an index it can cause disorganization that impacts the performance of queries using the index. The RTS columns REORGINSERTS and REORGDELETES (in SYSIBM.INDEXSPACESTATS) can be examined to ascertain the number of index entries inserted or deleted since the index was reorganized. A good rule of thumb is to consider reorganizing an index when 25% or more entries have been inserted or deleted:

- REORGINSERTS / TOTALENTRIES >= 25%
- REORGDELETES / TOTALENTRIES >= 25%

Another index-related statistic to pay particular attention to is REORGAPPENDINSERT. It contains the number of inserts into an index since the last reorganization for which the index key was higher than any existing key value. If this column consistently grows, you have identified an object where data is inserted using an ascending key sequence. You might consider lowering the free space for such objects, because the free space is wasted space if inserts are always done in ascending key sequence. You should also consider reorganizing the index when 20% or more entries have been appended:

- REORGAPPENDINSERT / TOTALENTRIES >= 20%

The number of index leaf page splits should also be examined when considering index reorganization. The RTS column that indicates this metric is

REORGLEAFAR. Think about reorganizing the index at 10% or more:

- REORGLEAFFAR / NACTIVE >= 10%

There are actually two RTS values that expose the disorganization of physical leaf pages. The RTS column REORGLEAFNEAR shows the number of index page splits that occurred since the last index reorganization in which the higher part of the split page was near the location of the original page. Far is worse than near, and that is why we examine REORGLEAFFAR instead of REORGLEAFNEAR. But you could consider using near pages as an index reorganization metric, too, if you wish. Because the impact is less use a greater percentage as a threshold, for example 40% or more.

What is the difference between a near indirect reference and a far indirect reference? A near reference is within the prefetch quantity, whereas a far reference is outside of the prefetch quantity.

When DB2 deletes a row index entries are pseudo-deleted. This means that the index entry is marked as deleted (pseudo-deleted), but it is not physically deleted. But it is not just a DELETE that can cause a pseudo-deleted index entry, UPDATE can, too. This is so because an index update is actually a DELETE followed by an INSERT. So you should examine the number of pseudo-deleted RIDs when determining whether to reorganize your indexes. In a non-data sharing environment, think about reorganizing indexes when 10% or more of the index is comprised of pseudo-deleted entries:

- REORGPSEUDODELETES / TOTALENTRIES >= 10% (non-Data Sharing)

In a Data Sharing environment you should be more cautious because a pseudo-deleted entry can cause S-lock/unlock when inserting to a unique index. So, consider 5% as the threshold when data sharing:

- REORGPSEUDODELETES / TOTALENTRIES >= 5% (Data Sharing)

If instead of RTS you rely upon RUNSTATS values for your reorganization decisions, you can track the page splits metrics and pseudo deleted entries. For page splits, use:

- LEAFFAR / NLEAF >= 10%

LEAFFAR is a column in SYSIBM.SYSINDEXPART; NLEAF is a column in SYSIBM.SYSINDEXES and SYSIBM.SYSINDEXPART.

For pseudo-deleted entries use:

- PSEUDO_DEL_ENTRIES / CARDF > 10% (or 5% for Data Sharing)

Both PSEUDO_DEL_ENTRIES and CARDF are columns in SYSIBM.SYSINDEXPART.

You will not have access to the number of rows inserted and deleted if you rely on RUNSTATS, but you will have access to another statistic, LEAFDIST (in SYSIBM.SYSINDEXPART). LEAFDIST helps determine the relative efficiency of each index. LEAFDIST indicates the average number of pages between successive index leaf pages. The more intervening pages, the less efficient the index will be. The definition of LEAFDIST is 100 times the average number of pages between successive leaf pages of the index.

The higher the value of LEAFDIST, the more you should consider reorganizing the index. Of course, if you already are using LEAFFAR (and possibly LEAFNEAR) then analyzing LEAFDIST will be of limited value as a threshold.

Finally, regardless of whether you are using RTS or RUNSTATS to set your reorganization thresholds, you should also monitor for index pending statuses. Whenever an index is in advisory REORG-pending status (AREO* or AREOR) or advisory-REBUILD pending status (ARBDP) as the result of an ALTER statement you should attempt to reorganize (or rebuild) the index as soon as reasonably possible.

Identifying Unused Indexes

DB2 9 for z/OS added the LASTUSED column to the SYSINDEXSPACESTATS RTS table. The LASTUSED column contains a date indicating the last time this 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.

This helps to solve the problem of determining whether or not an index is being used. Standard operating advice is to DROP or delete anything that is not going to be used. Examine the LASTUSED column over time to determine which indexes are truly not being used, and DROP the unused indexes. If the index is not being used, dropping it makes your reorganization planning easier. After all, you never have to reorganize an index that does not exist, right?

What About Clustering?

The cluster ratio of an index is an important statistic that is analyzed by the DB2 Optimizer when determining SQL access paths. Cluster ratio is stored in the CLUSTERRATIO column of the SYSIBM.SYSINDEXES table in the DB2 Catalog.

Of course, CLUSTERRATIO is of absolutely no help at all when determining whether to reorganize an index. Cluster ratio describes the data in the table space, even though it is reported in SYSINDEXES. It represents the percentage of rows that are in physical order by the index key. When data is highly clustered sequential access is improved because I/O operations can be reduced when retrieving the data in order by the clustering key.

Clustering is desirable and you should keep an eye on the CLUSTERRATIO of your indexes, especially clustering indexes. But reorganizing and index will **never** affect this statistic. To re-cluster data you must reorganize the table space containing the table upon which the index is defined.

What About New DB2 Versions and Hardware Improvements?

New versions of DB2 can impact your reorganization guidelines. For DB2 10 for z/OS, IBM has driven new improvements that can boost the performance of queries when using disorganized indexes. And new hardware features also can be used to offset the cost of disorganized indexes, at least to some degree.

Prior to DB2 10, synchronous I/O was used to scan disorganized indexes. But as of DB2 10, indexes can be scanned asynchronously uses list prefetch with the prefetch requests eligible for zIIP processing (see Sidebar). List prefetch reads a set of data pages determined by a list of RIDs taken from an index. By prefetching RIDs and accessing page by page, I/O can be reduced. With increased asynchronous processing, I/O does not need to wait for CPU processing. Some tests have shown that scans of disorganized indexes in DB2 10 can be up to 5 times faster than in DB2 9.

New disk hardware, notably the DS8000 R6.2, also delivers improved handling of disorganized DB2 indexes. The new hardware can meaningfully improve list prefetch I/O. By using High Performance FICON for System z channel time is significantly reduced. Additionally, a new feature known as turbo list prefetch (TLP) enhances the caching capabilities of the DS8000. Some tests have shown that throughput for list prefetch I/O may be more than tripled.

So perhaps an argument can be made that reorganizing your indexes is not as important once you move to DB2 10 for z/OS. And, indeed, there is some validity to that suggestion. However, not every shop will be using the latest IBM DS8000 disk. If you use another manufacturer's disk you will need to examine its capabilities.

Furthermore, indirect references can still cause performance problems, even for DB2 10. When you read a row with an indirect reference an additional GETPAGE is required, which requires synchronous I/O. Using list prefetch does not help with indirect references because the list of RIDs from the index contains only the original RID locations. List prefetch has no knowledge that the row has been relocated until the row is accessed.

And do not forget about pseudo-deleted index entries. Pseudo-deleted data will remain in the index, taking up space and slowing scans, until you reorganize it.

Summary

Moving to a metrics-based reorganization standard can help to improve the overall health of your DB2 databases, and thereby improve the performance of your application systems. Establishing thresholds and reorganizing based on those thresholds is the industry best practice approach. And by focusing more attention on the organization of your DB2 indexes, you can improve the ROI of your reorganization processes.

Good luck reorganizing!

SIDEBAR

The IBM System z Integrated Information Processor (zIIP)

The zIIP is a specialty processor that relieves general purpose processors of workload. Its primary goal is to accept redirected DB2 workload, although others types of workload also can be run on a zIIP. In general, distributed DB2 for z/OS workload and XML processing can be redirected to zIIP processors.

To fully comprehend what can and cannot run on a zIIP, we need to discuss TCBs and SRBs. For mainframe z/OS programs, code can execute in one of two modes: TCB mode, also known as task mode, or SRB mode. Most programs execute under the control of a task. Each thread is represented by a TCB, or Task Control Block. A program can exploit multiple processors if it is composed of multiple tasks, as most programs are.

An SRB, or Service Request Block, is a control block that represents a routine that performs a particular function or service in a specified address space. SRBs are lightweight and efficient, but are available only to supervisor state software. An SRB is similar to a TCB in that it identifies a unit of work to the system. But an SRB cannot "own" storage areas. SRB routines can obtain, reference, use, and free storage areas, but the areas must be owned by a TCB. SRB mode typically is used by operating system facilities and vendor programs to perform certain performance-critical functions.

In general, z/OS will dispatch DB2 work in TCB mode if the request is local, or in SRB mode if the request is distributed. These parallel tasks are assigned the same importance as the originating address space.

Pre-emptible enclaves are used to do the work on behalf of the originating TCB or SRB address space. Enclaves are grouped by common characteristics and service requests and since they are pre-emptible, the z/OS dispatcher – and Workload Manager – can interrupt these tasks for more important ones. There are two types of pre-emptible SRBs: client SRBs and enclave SRBs.

If the DB2 request is distributed DRDA workload, then it will be executed in enclave SRBs. If the request is coming over a local connection, then it will be dispatched between TCBs, client SRBs, and in some cases enclave SRBs (such as for parallel queries and index maintenance).

So what does all of this have to do with specialty processors? To run on a zIIP, the workload must run under an enclave SRB. The benefit of zIIPs? IBM and most ISVs do not charge for workload that runs on a zIIP. So, prudent use of zIIPs can reduce the cost of mainframe computing.

Published by:

Mullins Consulting, Inc.
15 Coventry Court
Sugar Land, TX 77479

Telephone: 281 494 6153

www.CraigSMullins.com

About Mullins Consulting, Inc.

Mullins Consulting, Inc. is a database research and consulting company specializing in database performance, database administration, and database tools. The company was founded by Craig S. Mullins, a data management strategist, researcher, and consultant. Craig has nearly three decades of experience in all facets of database systems development including developing and teaching DB2 and SQL classes, systems analysis and design, database administration and system administrator, and data analysis and modeling. He has worked with DB2 for z/OS since Version 1 and has experience working with other database technology including Microsoft SQL Server, Sybase ASE and IMS.