Surviving Common DB2 Performance Problems

By Craig S. Mullins

www.craigsmullins.com

Mullins Consulting, Inc.
database management & strategy

15 Coventry Court
Sugar Land, TX 77479

tel 281-494-6153

A Mullins Consulting, Inc. white paper written for:

Quest Software, Inc.
8001 Irvine Center Drive
Irvine, CA 92618
Telephone: 949 754 8000 | Fax: 949 754 8999
www.quest.com

Published by:

Mullins Consulting, Inc.
15 Coventry Court
Sugar Land, TX 77479
Telephone: 281 494 6153 | Fax: 281 491 0637
www.CraigSMullins.com

June 2005

Surviving Common DB2 Performance Problems

In today's modern enterprise, information is power; and database systems are the predominant technology for storing, managing, and accessing the data that enterprises use to turn into information. And DB2 is one of the leading database systems on the market.

DB2 is used by 100 percent of Fortune 100 firms and 80 percent of the Fortune 500. As such, it is an important component of the applications that drive the most significant businesses in the world.

Dynamic business requirements force constant change and overwhelming complexity on the typical IT environment.

Dynamic business requirements force constant change and overwhelming complexity on the typical IT environment. Regulatory compliance issues, time-to-market pressures, and industry consolidations cause turbulence as IT struggles to keep systems in sync with business demands. But managing systems becomes even more troublesome as technology trends evolve. Examples of these trends include business intelligence, e-commerce, and rapid software versioning.

And data growth continues unabated. Indeed, Winter Corp., a research and consulting firm, publishes a semi-annual survey of the largest and most heavily used databases in the world. The most recent Winter report, published in 2003, confirms this explosion of data with the average size of an OLTP database growing from 1 TB in 2001 to 4.4 TB in 2003. Grappling with performance problems in this rapidly changing – and growing – environment is complicated, at best.

At the same time, the mainframe workforce is aging and retiring. As the number of mainframe professionals with years of experience decrease, they must be replaced with younger technicians – and these technicians require a different interface than the green-screen, ISPF interface that is still the most common mainframe interface.

These trends drive home the need for DBAs to exert constant vigilance over the performance and availability of their DB2 environment. Any performance slip can impact the business resulting in lost revenue or damaged customer relationships. Furthermore, it is important to be able to achieve this performance management capability with a modern, easy-to-use interface. Enterprises cannot reap the full range of benefits from their database systems without modern, effective performance monitoring and management capabilities.

Enterprises cannot reap the full range of benefits from their database systems without modern, effective performance monitoring and management capabilities.

The answer is to equip the DBA staff with solutions that can simplify and automate the performance management process in a cost effective manner. There are many powerful, DB2 performance monitoring tools available, but many of them are difficult to use. As the mainframe skills gap continues to widen, products with a flexible, graphical user interface make analyzing mainframe performance easier.

According to the Mainframe Report 2005 from Arcati, Ltd.[1] "when a (mainframe) person leaves, it is not simple technical knowledge they take with them but rather twenty years' intimate and detailed understanding of core business applications." Most legacy performance tuning products are character-based and difficult to master. Products deploying a GUI will be better-suited for the future workforce.

[1] The Arcati Mainframe Yearbook 2005, "The Mainframe: Forty Years On", page 7

DB2 Performance: At Fifty Thousand Feet

To help meet these challenges, DBAs must possess the necessary tools to automate the maintenance and performance management of their DB2 environment. But what do we really mean by DB2 performance?

Let's start by thinking about this question at a high level – a 50,000 foot level, if you will. Every DB2 application requires three components to operate: the system, the database and the application itself. The system refers to the DB2 subsystem installation and its interfaces to other system software (e.g. z/OS or CICS). The database refers to the database objects that house the application data. And the application is the host language code and SQL that provides each program's functionality. To effectively deliver DB2 performance, the DBA must be able to monitor and tune each of these components.

Let's take a look at some of the most common problems that DBAs encounter when managing each of these components.

DB2 Subsystem Performance

Successfully managing DB2 performance begins with the DB2 subsystem. In order to deliver consistent system performance, the DBA must have the resources to monitor, manage and optimize the resources used by DB2.

Tasks required for system tuning include the proper allocation and management of memory structures (e.g., buffer pools, EDM pool, etc.), storage management, integration of the DBMS with other system software, proper usage of database logs, and coordination of the operating system resources used by the DBMS. Additionally, the DBA must control the installation, configuration and migration of the DBMS software. If the system isn't performing properly, everything that uses the system will perform poorly. In other words, a poorly performing system impacts every database application.

> The more efficiently memory is allocated …the better DB2 will perform.

Efficient usage of memory is one of the biggest struggles for DB2 performance tuners. Relational database systems love memory and DB2 uses memory for buffer pools, the EDM pool, the RID pool and sort pools. The more efficiently memory is allocated to these structures, the better DB2 will perform.

DB2 provides 80 different buffer pools that can be configured to cache data in memory for DB2 programs to access. Forget about trying to follow a cookie-cutter approach to buffer pool management. There is no simple formula to follow that will result in an optimally buffered system that works for every implementation. Each shop must create and optimize a buffer pool strategy for its own data and application mix.

The trick is to know how to tune your buffer pools so that they match your workload. Of course, you also have to ensure that you do not exceed the memory allocation for your operating system and platform, or system paging will occur and performance will suffer.

To configure your buffer pool based on workload, you need to know how your database objects are accessed. There are basically two types of access: sequential and random. An access request is sequential if it starts by reading one record and continues reading the next records in sequence; this may or may not reflect the way the data is actually stored on disk. Random access, on the other hand, is typified by requests that access a single record based on a key.

If you have database objects that are accessed primarily sequentially, you can take advantage of this knowledge by assigning them to a buffer pool that is configured for sequential access. This is accomplished by setting that buffer pool's sequential steal threshold (VPSEQT), the parallel sequential steal threshold (VPPSEQT), and the assisting parallel sequential steal threshold (VPXPSEQT) to indicate that access is mostly sequential.

VPSEQT…specifies the percentage of the buffer pool that should favor sequential processing.

These thresholds dictate to DB2 how much of the buffer pool should be set for sequential processes. Consider the VPSEQT parameter, for example: it specifies the percentage of the buffer pool that should favor sequential processing. For example, by setting VPSEQT to 95, DB2 will favor stealing a random page over a sequential page until 95 percent of the buffer pool is filled with sequential pages. VPPSEQT and VPXPSEQT are specified as percentages of VPSEQT and are used to specify the sequential steal threshold for parallel operations.

Of course, to successfully configure your buffer pools you will need to be able to determine how your database objects are being accessed: sequentially or randomly. Furthermore, you will need to have a method of monitoring the performance of those buffer pools once configured. Each buffer pool has additional thresholds associated with it that provide usage and control information about buffer pool operations.

You will want to monitor the prefetch disabled threshold, especially for buffer pools that are tuned for sequential access. This threshold kicks in when 90 percent of the buffer pool pages are unavailable. When this threshold is reached, DB2 no longer initiates prefetch operations and will cancel prefetch operations in progress.

Additionally, you will need to watch for the data manager critical threshold. It is reached when 95 percent of the buffer pool pages are unavailable. At this point DB2 will access pages once for each row that is retrieved or updated in that page. This should be avoided at all times. Basically, when you hit this threshold and you retrieve or update more than one row on the same page, DB2 will invoke a separate page-access operation for each access. And that will cause severe performance degradation.

Finally, you will need to monitor the immediate write threshold. When 97.5 percent of the buffer pool pages are unavailable, DB2 writes updated pages as soon as the update is complete. In other words, writes are performed synchronously instead of asynchronously. This causes an increase in the number of write I/Os performed by DB2.

If any of the last three thresholds are reached there are only two methods of making changes to alleviate the problem: either increase the size of the buffer pool or reallocate the database objects to other buffer pools. But remember, larger buffer pool sizes are not always more efficient. DB2 is not the only consumer of memory in your system, so take care when allocating buffer pool memory.

A good buffer pool monitoring technique is to monitor the buffer pool hit ratio for each of your buffer pools. The hit ratio can be calculated as follows:

```
((GETPAGES - SUM OF ALL PAGES READ) / GETPAGES ) * 100
```

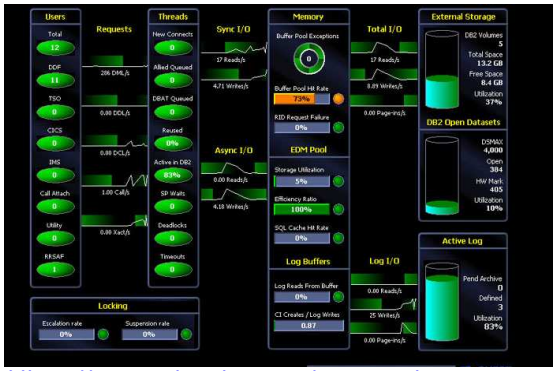
The "sum of all pages read" must account for synchronous and asynchronous I/O. This ratio basically tells you the percentage of times that a requested page was available in memory. When a page is available in memory DB2 does not have to read the page from disk and performance improves. The larger the hit ratio number, the better.

Up to this point we have been more worried about reading data, than writing it. But there are thresholds that need to be tuned that control how modified data is written from the buffer pools back to disk. These are the deferred write thresholds (DWQT and VDWQT).

DWQT is set as a percentage of the buffer pool that can be occupied by updated pages and in-use pages. The default value is 50 percent, but this is usually too high for most sites. When the percentage of unavailable pages exceeds the threshold, asynchronous writes are scheduled for the updated pages. VDWQT is similar to DWQT, but for a single data set; its default is 10 percent (and that is likely to be too high as well).

By lowering your deferred write thresholds your DB2 subsystem will have regular asynchronous writes of updated pages to disk. Pages that are frequently referenced will remain in the buffer pool even if updates are written out to disk.

But never set VDWQT higher than DWQT; doing so would have no effect because the pages for each data set also apply to the overall percentage.



Visually monitoring and managing thresholds and hit ratios aids in DB2 performance tuning.

Another consumer of memory in your DB2 subsystem is the EDM Pool, which is used for caching internal structures used by DB2 programs. This includes DBDs, SKCTs, CTs, SKPTs, and PTs. It also includes the authorization cache for plans and packages, as well as the cache for dynamic SQL mini-plans. DB2 V8 splits the EDM pool into separate pools for each of these structures. As a general rule, you should shoot for an 80 percent hit rate for EDM pool objects. Another way of stating this is that only one out of every five times a structure is required should it need to be loaded into memory.

It is important to monitor EDM Pool usage because an EDM Pool failure can bring DB2 to a screeching halt. If there is no place to cache the structures needed for a new iteration of a program to run, then that program will wait. And so will the user trying to run the program.

Of course, memory allocation is not the only component of DB2 subsystem tuning. You will also need to be able to view information about locking to resolve lock timeout and deadlock situations, as well as to be able to configure the locking system parameters (DSNZPARMs) appropriately. Actually, having a method to quickly and easily view the DSNZPARM settings for your DB2 subsystem is a necessity – especially for those times when you are in the middle of resolving a difficult problem.

Determining the cause of locking problems can be particularly troubling. When wait time is excessive, performance will degrade. A vigilant DBA must be capable of tracing the lock list to determine if it is properly sized, checking the average time that applications are waiting (and why), and figuring out how many locks are allocated for a database.

Determining the cause of locking problems can be particularly troubling.

Keep in mind, too, that this is a high-level paper on DB2 performance management and that there are many more components required to achieve proper DB2 system performance. Other system elements requiring attention include allied agent (CICS, TSO, etc.) tuning, monitoring disk I/O, tuning logging, and Parallel Sysplex configuration and management for DB2 data-sharing shops.

Keeping an eye on so many performance monitoring and tuning details can be an overwhelming task without a tool that can gather and visually represent the diagnostic information in an organized fashion.

DB2 Database Objects and Performance

Although many factors contribute to poor database performance, it is particular important to pay attention to your database objects. The term database objects refers to the table spaces, tables, and indexes that comprise your DB2 databases.

Of course, the first factor for ensuring efficient database objects is practicing proper database design. This means starting from a fully normalized logical data model. The data model should be used as a template for building your physical DB2 database, deviating from the model only for performance reasons.

After implementing an effective physical DB2 database, the next step is to ensure that you are capturing statistics appropriately for the database objects.

After implementing an effective physical DB2 database, the next step is to ensure that you are capturing statistics appropriately for the database objects.

Inaccurate statistics are a leading cause of inefficiencies in DB2 applications. Indeed, during a presentation at a recent IDUG conference (2004) a presenter from IBM's Silicon Valley Lab indicated that as many as half of all access path "problems" presented to IBM's technical support were caused by inaccurate, outdated, or missing statistics.

Statistics are accumulated by the RUNSTATS utility. You should schedule a regular execution of RUNSTATS for all dynamic database objects. If the data grows or changes on a regular basis, you need to plan and run the RUNSTATS utility accordingly. Only by having up-to-date statistics can the DB2 optimizer have the correct information to generate accurate access paths for your SQL queries.

RUNSTATS also gathers additional information about the current state of each object. This state, or organization data, is used by the DBA to determine when an object needs to be reorganized. Table spaces and indexes become disorganized as business users run application programs that insert, update, and delete data in DB2 tables – causing the once-ordered data to become fragmented. Disorganized data can be particularly detrimental to the performance of your DB2 systems. To understand how disorganization impacts performance, let's review the concept of clustering.

Each DB2 table can have a single clustering sequence defined for it. This is accomplished with a clustering index. When clustering is defined, DB2 *attempts* to maintain the sequence of the rows in the physical order in the table space.

But as data is inserted and modified and pages fill up, there may not be sufficient space available to insert the data in the proper sequence. So the data is inserted where space exists and clustering begins to break down. As the data becomes unclusted, the performance of sequentially accessing data in clustering order begins to degrade because more I/O operations are required to retrieve the data. Running a reorganization will re-cluster the data, and thereby improve performance.

Of course, clustering is not the only organization detail that can impact performance. You must also keep an eye on other details such as:

- The percentage of each table space containing data from tables that have been dropped. As this percentage increases, performance gets worse.
- The leaf distance for indexes, which indicates the average number of pages between successive index leaf pages. Performance degrades as the leaf distance increases.
- Far-off and near-off pages for indexes, which estimates how many of the rows in the table are inefficiently located. As the number of far- and near-off rows increases, performance gets worse.
- Near and far indirect references for a table space, which indicate the number of rows that have been relocated either near (2 to 15 pages) or far away (16 or more pages) from their original location. This relocation can occur as the result of updates to variable length rows. As the number of near and far indirect references increases, performance gets worse.

All of these factors impact on the efficiency of your database objects. Failing to monitor and correct organization problems will cause every application accessing these objects to suffer performance degradation. The wise course of action is to monitor these statistics. Set thresholds for each that cause a REORG to be scheduled when the threshold value is reached. For example, monitor the cluster ratio and when it falls below 90 percent schedule a REORG.

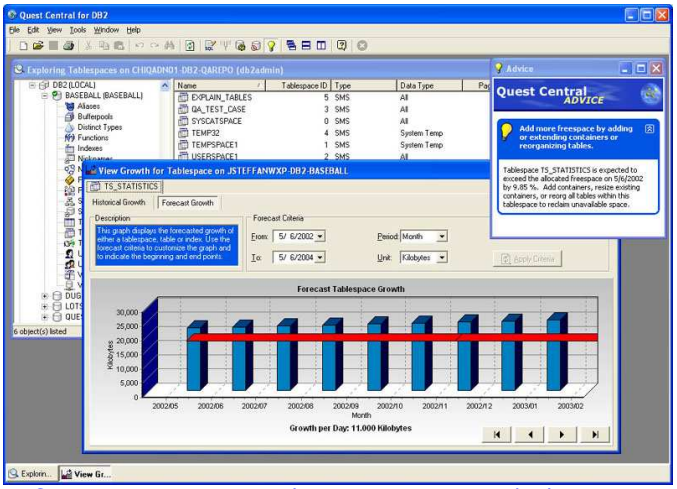
Failing to monitor and correct organization problems will cause …performance degradation.

Creating the appropriate indexes for your DB2 tables and applications is a discipline that spans database and application tuning. To create appropriate indexes, the DBA must have the SQL that is to be used against the DB2 objects in order to understand the access patterns. Furthermore, an understanding of the workload that is being run against your system is needed. This information enables you to balance index creation and buffer pool placement so that your setup matches what is being run against it.

Usually, though, DBAs tend to create indexes before having knowledge of the SQL that will be running against their tables. Some indexes are required regardless of the SQL being run, for example to support a primary key or unique constraint, or to enhance the performance of referential constraints. But the vast majority of your indexes should be to support SQL in your application code.

Of course, there are more severe database performance problems that you can encounter. For example, consider the impact to your applications of running out of space. When a table space runs out of space and has no more room to expand or extend, any attempt to add more data will fail. And a failure is the worst kind of performance problem – an availability problem.

The best approach to managing the performance of your DB2 database objects is to use a tool that can help you to automate the setup and monitoring of space and organization thresholds. If you fail to take this approach, you are probably either wasting CPU cycles by reorganizing before it is required; or you are incurring performance problems because you are not reorganizing until well after it is required.



Space management issues can severely impact DB2 performance.

A dynamic, automated toolset can help you to reorganize at just the right time and thereby optimize your resource usage and your DB2 systems… and to predict outages and problems so you can resolve the issues before the problems happen.

DB2 Application Performance

The third component of DB2 performance is the application itself. As much as 80 percent of all relational database performance problems can be tracked back to inefficient application code. The application code consists of two parts: the SQL code and the host language code in which the SQL is embedded.

Although SQL is simple to learn and relatively easy to learn at its most basic level, it is actually quite difficult to master. There are, of course, some general rules of thumb you can apply to help you to build efficient SQL statements. Keep in mind though, that a rule of thumb is helpful as a general guideline, but it is not to be applied in every situation.

The first major rule of thumb for SQL programming is to build the SQL to do the work instead of the application program. By this I mean that you should code the proper WHERE clauses in your SQL instead of waiting to filter data in your host program. Additionally, do not treat DB2 tables like master files. Code SQL joins instead of multiple cursors where a "master" cursor is read that then drives the other cursors. SQL is a set processing language and it should be coded that way.

Another common problem is retrieving more data than is required. Your SQL statements should retrieve only the columns required, never more. Each column that has to be accessed and returned to you program adds overhead that can be avoided if the column is unnecessary. Sometimes programmers try to use the SELECT * shortcut – and though

that may be fine for quick and dirty testing, it should never be allowed in production programs.

It is also important to understand the different types of predicates and their impact on performance. Stage 1 predicates are better than Stage 2; and indexable predicates are better than non-indexable.

A predicate that can be satisfied by the Data Manager portion of DB2 is referred to as Stage 1. A Stage 2 predicate has to be passed from the Data Manager to the Relational Data System to be satisfied. The Data Manager component of DB2 is at a level closer to the data than the Relational Data System. The earlier in the process that DB2 can evaluate the predicate, the more efficient processing will be. So, Stage 1 predicates are more efficient than Stage 2.

Additionally, a query that can use an index has more access path options, so it can be more efficient than a query that cannot use an index. The DB2 optimizer can use an index or indexes in a variety of ways to speed the retrieval of data from DB2 tables. For this reason, try to use indexable predicates rather than those that are not.

At this point you are probably asking something like "which predicates are Stage 1 and which are indexable?" Where predicates are processed can change from version to version of DB2. This information is documented in the IBM DB2 Database Administration Guide manual, which can be downloaded from the IBM web site free-of-charge.

Another technique for improving performance is to create indexes to support your ORDER BY and GROUP BY clauses. If an index is available on the columns you specify to these clauses, DB2 can use the index to avoid invoking a sort. And that is likely to improve the performance of your SQL statement.

After the SQL is coded, to ensure proper performance you can review and interpret access path information collected using the EXPLAIN command. The EXPLAIN command can be executed on a single SQL statement or multiple statements in an entire program. EXPLAIN captures information about the access paths chosen by the DB2 optimizer and populates it into a special table called a PLAN_TABLE. The PLAN_TABLE contents are encoded and a DBA or analyst must interpret that data in order to determine exactly what DB2 is doing.

To effectively tune SQL though, you will require additional performance metrics than just what is in the PLAN_TABLE. Based on execution circumstances DB2 may have to re-evaluate an access path at run time: for example, to avoid using RIDs in the event of a RID pool failure or if there is a lack of storage for a list prefetch operation.

SQL performance tuning requires an ability to interpret the contents of the PLAN_TABLE in conjunction with the SQL code, host language code and information from the DB2 catalog...

SQL performance tuning requires an ability to interpret the contents of the PLAN_TABLE in conjunction with the SQL code, host language code and information from the DB2 catalog to judge the efficiency and effectiveness of each SQL statement. Analysis of the access path information and additional performance metrics can require a tuner to make SQL modifications. Perhaps you will need to modify predicates to influence indexed access, or perhaps revise a statement to use Stage 1 instead of Stage 2 predicates.

An in-depth analysis of your SQL statement can lead to the determination that index changes or additional indexes will be required. So, you see, application tuning can lead back to database object tuning. And index tuning can be tricky; changing an index to improve the performance of one query can degrade the performance of other queries.

Build indexes on the predicates of your most heavily-executed and most important queries. Do not arbitrarily limit the number of indexes per table. Create as many as are necessary to improve query performance. But remember to balance that against your data modification needs. An index can improve query performance, but it will degrade the performance of data modification because the index has to be modified when its underlying data is inserted, deleted, or updated.

You might also consider overloading indexes by adding columns to encourage index only access. For example, if you have a query that is frequently run throughout the day and it accesses five columns, four of which are in an index, adding the fifth column to that index allows DB2 to get all of the data from the index alone. Avoiding I/O to the table space can help to improve performance.

Finally, remember that efficient SQL coding and proper indexing are not the only reasons for poor application performance. An inefficiently designed host language program can cause poor performance, too. Host language code is the code written in a programming language such as Java, COBOL, C, or your programming language of choice. SQL statements are embedded into the host language code and it is possible to have a well-tuned SQL statement embedded into an inefficient application program.

Once again, to effectively manage the performance of your DB2 applications requires flexible and easy-to-use tools. It is important that you are able quickly to find the programs that are consuming the most resources. You do not want to have to wade through pages and pages of performance reports to find the offending programs; neither do you want to have to navigate through screen after screen of options and confusing menus.

The best option is to use a tool that works as a tuning lab for your SQL. You will need the ability to find the offensive SQL and then be able to play "what if" scenarios by modifying the SQL and gauging the performance impact of each change. It is even better if the tool can provide expert tuning advice to guide you as you change your SQL statements.

The Complexity of DB2 Performance

We have discussed a number of the most common DB2 performance-related issues and problems in this paper. After digesting this information, it should be apparent to you that DB2 performance management is quite complex. Mastering all of the nuances covered in this paper is difficult – and remember, these are just the most common problems. Much more needs to be mastered to ensure efficient DB2 performance.

As the dynamics of the workforce change and new releases of DB2 add features and complexity, the problem is exacerbated. The bottom line is that software tools that automate the detection and correction of DB2 performance problems helps to simplify DB2 performance management.

> …software tools that automate the detection and correction of DB2 performance problems help to simplify DB2 performance management.

Furthermore, database performance tools need to be designed to be easy to use, yet highly functional. As skilled mainframe professionals retire, they will be replaced with younger technicians who have not had the same level of exposure to mainframe systems. These younger technicians will expect to use tools with graphical interfaces, pull-down menus, and point-and-click functionality. Of what possible use is a complex tool with broad functionality that no one can figure out how to use?

Summary

To succeed in your DB2 performance management objectives you must be able to monitor and tune all three major components of your DB2 environment: the DB2 subsystem, the database objects, and your DB2 applications.

A wise organization will be prepared to adapt to the changing workforce and the related complexities being driven into today's IT environment. One means of preparation is to deploy integrated performance tools with a modern interface to ensure the efficient performance of the DB2 environment. Doing anything less will cause performance problems… and all of the related business problems that go along with them.

About Quest Central® for DB2

Quest Central for DB2 brings together all the functionality you need throughout the day, whether you are running DB2 on z/OS, OS/390, UNIX, Windows or Linux. With an easy-to-use graphical interface and integrated console, Quest Central provides powerful performance diagnostics, database administration, SQL tuning and space management all from a common interface. To view additional information about Quest Central for DB2 or to download a free trial, go to: www.Quest.com/DB2

About Quest Software

Quest Software, Inc. (NASDAQ: QSFT) is the leading provider of application management solutions. Our software gives businesses confidence that their vital applications will be available and performing well – while simultaneously driving down the costs of managing them. By focusing on the people and technology that make applications run, Quest Software enables IT professionals to achieve more with fewer resources and get the most out of existing application investments. Founded in 1987 and based in Irvine, California, Quest Software has offices around the globe and more than 1,650 employees.