

CLEANING YOUR DATABASE GARAGE

It's spring and time to start cleaning out that garage again. Is it time to do the same for the database used to park your application data? As with sorting through items stored in a garage; deciding which database items, especially indices, to keep and which to throw away is the biggest challenge.

Indices are created when the programmer or database administrator creates a logical file, adds a referential constraint, defines a unique constraint, or issues the Create Index SQL statement. For years, system administrators have known that too many indices add "drag" to everyday operations. Unfortunately, databases tend to accumulate indexes over time since it's easy to create one but difficult to measure its cost.

So when an index is created, what is its cost?

Disk space:

The most obvious cost is the disk space that it takes up. The space can be seen by issuing a DSPFD on the logical file and looking for 'Index size'. Unless the file is very large however, the amount of space taken up by the index is normally of little concern.

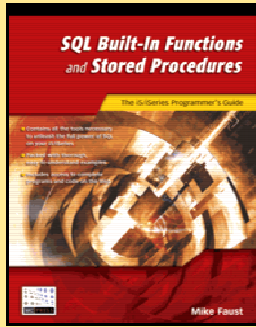
most common challenge is the length of the backup window and not the amount of tape used to save the objects.

Memory:

As indices are updated because records are changed in the physical file, memory in the job's pool is used to hold the index. This means that the memory pool must be large enough to hold it and if the index isn't used often it increases the need for memory for adequate application performance.

Application performance:

As records in physical files are inserted, deleted, or updated, the indices associated with the physical file must be updated. Both CPU and disk I/O's are required to perform this maintenance. This overhead is the biggest concern of system and database administrators.



Summer Special
Hot from MC Press
And noted author Mike Faust

Free copy with new product
License this summer
from Centerfield.
Offer expires August 31, 2005

Offer cannot be combined
with other promotions or offers.

Backup media and speed:

Because indices are generally backed up with the changed file, they also take up space on backup media and slow backups down. Unlike the disk space, the

(Continued on page 3)

LONG RUNNING

BATCH JOBS

At some point, all IT shops have jobs they wish would run faster. These could be the batch job that runs at night, backups, or reports that are run during the day. So what are the common reasons jobs run long and what can you do about them?

There are many reasons jobs run longer than they should, but fundamentally they fall into three categories:

- Not enough hardware resources exist
- The programs used by the job are not efficient
- Other work competing for resources is slowing the job down

Let's take a look at each of these root causes and see what can be done to verify when each is happening:

Hardware. Typically if you do not have enough hardware resources you already know it. If your system consistently runs poorly (i.e. you have many jobs that run too slowly and your interactive response time is poor), you likely do not have enough CPU, memory, or disk arms. To determine which part of your system is a bottleneck you can look at IBM's performance tools or interactive commands like WRKSYSSTS. These commands will show you the CPU utilization and pool faulting rates.

(Continued on page 2)

JOIN US!

Tuesday
June 7, 2005

Customer Appreciation Event
And
Art Tour

Rochester Art Center
Adjacent to the Mayo Civic Center
Rochester, Minnesota

4:00pm—8:00pm

asktheexpert@centerfieldtechnology.com

Long Running Batch Jobs (Continued from page 1)

Obviously, if your system consistently runs with CPU utilization over 90% then more processing power is needed. On the other hand, your CPU utilization may be very reasonable (e.g. consistently in the 35-55% range), yet you still have performance problems. In this case, you should look at page faulting rates in your pools as well as the utilization of your disk arms via WRKDSKSTS. If the faulting rates or arm utilization is outside of IBM's general guidelines, then it makes sense to look at either adding more memory or disk units.

Application efficiency. The causes behind poor application efficiency can roughly be put into two categories; 1) poor algorithm design or 2) inefficient database access.

Poor algorithms lead to performance problems when the application is expected to service more users or handle larger database files than it was designed for. Normally applications should be designed to follow these performance guidelines:

- If part of the algorithm can be eliminated, then do it. When unnecessary work is being done, eliminating the waste results in 100% of those resources being recovered and 100% of their contribution to response time eliminated.
- If a task can be done once and the work reused many times, then only do it once.
- If part of an algorithm has to be done many times, then try to make it efficient.

The point of these obvious guidelines, is that an application should be tuned by eliminating work first – followed by making parts of it more efficient. Much time and effort can be devoted to making part of an application run much faster without having a significant impact on overall performance. For example, if a programmer can make some core process run twice as fast, but that process only constitutes 4% of the total path length of the application, then the gain is only 2% which may not be significant enough to even notice.

A second issue with application efficiency relates to database access. Many applications have very large databases and probably use at least some SQL to accomplish their work. If either the SQL statement or the database has not been tuned to provide the fastest method for DB2 to use, then long-running requests can result. Which database requests are the most resource-intensive can be difficult to identify, but it is very important to work on the ones with the most leverage to avoid wasting time working on the wrong problem. Solving SQL performance issues requires an ability to isolate the root cause of the problem and intelligently experiment with different alternatives. Because the DB2 query optimizer ultimately makes the

than the background work or use a larger timeslice or both. If disk I/O appears to be the problem, then the most important work should run in the larger memory pool. Alternatively, the less important work should run in a smaller pool in order to “slow” the background work and thus free up resources for the important job(s).

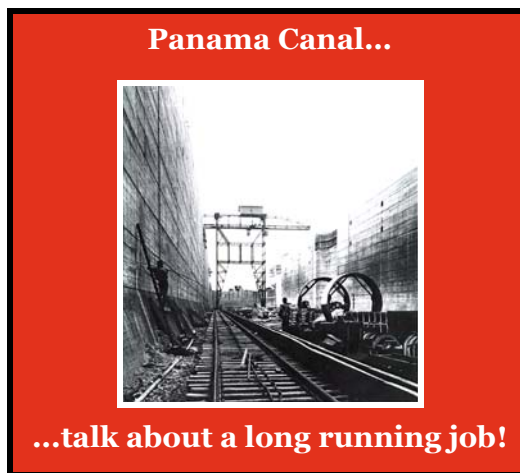
Solutions for application problems

While it is relatively obvious what approaches are necessary to solve a shortage of hardware resources or how to deal with competing work, it is less obvious what to do about poor application performance. More often than not, you have purchased the application from a third party and may not have the source code. The good news is that there are ways to speed up these application problems if they are caused by a database bottleneck.

The most straightforward way to deal with SQL performance problems is to build the correct indices. A second approach, if you are on V5R2 or higher is to make sure database statistics have been created for columns referenced in WHERE clauses that do not have an index.

If the application appears to be constrained by I/O, then the most important task is to identify which objects are causing the bottleneck. Once those have been determined, a variety of options can be used including use of the SETOBJACC command, use of the IBM journal cache PRPQ, use of the IBM SMP feature for DB2, the use of delayed maintenance indices, or the creation of an index that allows the query optimizer to choose a different access plan.

Summary. Speeding up jobs can be tricky because there are a myriad of possible causes and another myriad of possible solutions. If you approach the problem carefully and do the right kind of analysis, then the solutions will come naturally. Hopefully we've provided some insight into some of those solutions.



decision about how to access and extract information, it is most important to be able to build a request and database structure to facilitate good optimizer choices.

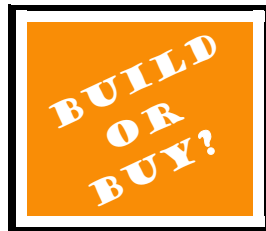
Competition. A third reason that one or more jobs may be running slowly, is that other work is contending with it. This competition can come in the form of predictable, necessary work or from ad hoc activity such as an interactive query. If competing work is predictable, the alternatives are to 1) reschedule the work so there is less overlap in the execution times or 2) to rebalance resources so that the most important jobs get more resource. The second method can be used to control CPU, disk I/O or memory depending on what the high-priority job needs the most. Obviously if CPU is the bottleneck, the important job should run at a higher priority

*Cleaning Your Database Garage
(Continued from page 1)*

Like applications that have unused programs, database indices tend to accumulate and never go away. This happens because the reason the index was created in the first place is forgotten and it can be tedious to find the programs that use the logical file and to determine if the program itself is still used. As a result, many databases have a significant number of indices just cluttering the library and adding unnecessary friction to system and application performance.

So what can you do about this problem? You would be ahead of the game if there were a way to generate a list of indices that have NOT been utilized by a system for a period of time, and then delete them, or better yet, accurately assess the cost of each index in light of its benefits and then make a decision to keep or delete the index. Fortunately, delta/TRACKER and insure/ANALYSIS provide these capabilities. In fact, the existence of indices on the iSeries is one of the core issues addressed by the Centerfield Technology insure/SQL suite of iSeries performance tuning tools.

A good technique for a truly clean and useful garage, is to take a weekend, have a garage sale with all the items you've not used for over one-year and then throw away all un-sold items. But always keep and maintain your tools; you never know when you'll need them.



to control the use of parallelism and the subtle aspects that are important when building similar tools.

If you wanted to write a simple tool to control who and when parallel queries could be run, how would you go about building it? The first thing to think about would be to identify what operating system functions allow you to control parallelism. On the iSeries, the two easiest ways are through the use of the QQRDEGREE system value or through the use of the CHGQRYA command. Next, you need to think through the requirements of the tool. Some questions you might ask yourself:

- **Is there a need to have different rules for certain jobs or users?** If there is, then using the system value would not provide a viable solution since system values apply to all jobs and users.
- **Are there times during the day that the use of parallelism would be allowed and other parts of the day that it should be restricted?** If so, you need to poll the system and time of day or have a mechanism to intercept new database requests so that the parallel settings can be changed on the fly.
- **Do you need to treat ODBC/JDBC jobs differently?** If you do then you will

In our last newsletter, we talked about the performance benefits of database parallelism as well as the pitfalls. In this issue, we discuss how a tool might be written

either need to write an iSeries exit program for the database server, occasionally capture a list of server jobs and change their settings, or use work management APIs to be informed when new jobs start. If exit points are used, then special considerations are necessary for installation and knowing when your program will be called.

- **Is there a need to log (audit) changes made to jobs?** Often it is very desirable to have a "trace log" of information that shows what changes were made, when they were made, and why they were made. In some environments, all tools have this requirement to one extent or another in order for IT to diagnose functional or performance issues.
- **Would it be useful to treat batch jobs differently than interactive?** This requirement means you must define what "interactive" and "batch" mean. For example, some shops consider ODBC or JDBC server jobs to be interactive work even though they are considered batch jobs by OS/400.
- **Does configuration need to be simple?** In very dynamic shops, user profiles change often, applications are always being modified, and system upgrades occur on a predictable schedule. If these changes mean that the rules regarding the use of parallel queries will change over time, then it makes sense that the tool should be simple to configure.

This article explains some of the techniques required to build tools to control parallelism (or other system and job settings) and provide a set of questions to think about before the tool is written. We hope it has provided useful insight into tool construction and the hidden challenges to building robust and flexible utilities.

Short on time and staff? Then [click here](#) to see what years of development can save your iSeries shop.

Fun Reading!

Elvis Budimlic
Development Director
ebudimlic@centerfieldtechnology.com

SYSTEM STATE ~ TO BE OR NOT TO BE?

Recent articles in the electronic press have talked quite a bit about IBM's changing policy for system state programs. Within the next year or two, the iSeries will prevent system state programs from being installed under i5/OS. The purpose of this policy is to increase the security of the iSeries by preventing certain program from taking liberties with your system or objects. The new restrictions are likely to impact many vendors that sell software to the iSeries market.

Centerfield Technology, however, has always had the policy that our products and tools would run at the highest security levels on the iSeries and conform to IBM standards relative to the use of publicly available interfaces. As a result, you can be assured that software written by Centerfield will continue to operate unimpeded by IBM's new rules.



Solution400 International (800) 231-5280 (760) 631-5280 (760) 631-5285-fax 44(0) 192-576-4248-UK



GSI
Your Global Business Solution

Kevin R. Herrig
President

Office: 770-479-0777
Mobile: 678-524-0787
kherrig@gsi-solutions.com

3760 Sixes Road
Suite 126, PMB 240
Canton, GA 30114

For newsletter sponsorship opportunities, please contact:
Jennifer Halverson at 507.287.8119 extension 101
or jhalverson@centerfieldtechnology.com