

Out in Left Field

strategic tools for strategic teams Volume Two, Issue Four

FALLING OFF A CLIFF

Part Deux

In our last newsletter, we talked about why

queries involving many tables can perform very poorly. In fact, some of these queries can run for weeks (or months!) if they were allowed to complete.

This article will discuss what tools you have to help the optimizer make the correct choice – and in the most extreme case, override DB2's implementation and force the join to work the way you want.

The most important goal in tuning a join query is to achieve the correct table order. By default, the DB2 query optimizer chooses the order based on estimates it derives based on table size, statistics, and index configuration. The optimizer's general approach is to put the tables with the smallest number of selected rows on the left. In other words, the tables with the fewest join values are put first to reduce the fanout and the potential explosion of result rows.

So lets say you are generally happy with the performance of your join queries, but you have several that are very slow. What steps do you take to tune them?

Join indexes: The first step is to make sure the columns used to relate one table to another have an index built for them. For example, if you join the CUSTOMER table to the PURCHASE table by CUSTOMER_ID and

PURCHASE_ID then make sure an index exists with those keys. This is important for two reasons. The first reason is very obvious. If a nested loop join is the most efficient method to connect the data, the index will be used to search for the related rows in the secondary table. The second reason is less obvious. As part of the optimization process, DB2 tries to determine the number of rows in one table that relate to the one being joined. To do this, some tricky math is required to make an accurate fanout estimate. The estimate, if wrong, can cause a very poor join order to be used, which can have a disastrous effect on performance. While the query optimizer has multiple ways of determining the fanout, the most accurate method is to look at a permanent index that contains all of the necessary columns.

The good news is that a well-designed relational database generally has the most common join indexes pre-defined. However, users can come up with creative ways to relate data and queries are becoming more and more complex over time. These factors mean that it is likely that the original database designer could not have predicted index needs that surface as the application matures.

Selection + Join indexes:

A vast majority of join queries also contain selection criteria. If the selection criteria include equal comparisons, the columns in those predicates can also be

included in the "join index". Because the equal predicates can be combined with the join columns, a single index that has the WHERE columns followed by the join keys can be used to speed up the join processing – in essence two pieces of the SQL statement are completed at the same time.

Order by and Grouping:

If your join query has an ORDER BY or GROUP BY clause, the performance of the statement could suffer if the columns used in them come from different tables. Because the data for the ordering or grouping comes from different physical locations, the data must be "glued" together before it can be processed. This means the query optimizer has fewer choices in how to implement the query and often times it means that temporary result tables must be created to complete the query. If this type of processing must be done, one approach is to implement the temporary table yourself to remove the responsibility from the optimizer. This approach will typically take multiple passes though the data but may actually perform better than forcing all of the work to be done in a single, complex SQL statement.

(Continued on page 3)



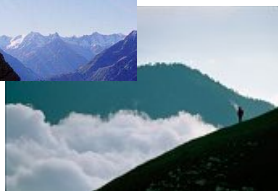
Inside this issue:

COVER STORY: Falling Off the Cliff Part Deux	1
Disk Space: Strategy & Tactics	2
Do you Need a Total Audit Solution?	3
Reader Input	3
Make Your UDFs Scream	4
This Should be a	5

teamwork

performance

success



Disk Space: Strategy & Tactics

Good chess players know that a solid strategy plays a key role in winning. A strategy provides a framework where moves are focused on achieving particular goals and help the player narrow down the number of moves to analyze. Weaker players often do not have a plan, which leads to random play and haphazard results.

Good chess players also know that tactics are important. Often a move that may not fit the overall strategy is necessary to prevent an opponent's threat or to take advantage of an opportunity that is too good to pass up.

Managing the disk space on your iSeries or IBM System i is much like a good game of chess – a clever mix of strategy and tactics can make your life much easier.

From a strategic standpoint, it is important to properly plan out what your disk requirements are over the long run so it can be installed and configured when it is needed. Conserving disk space and not wasting it unnecessarily is also key. Furthermore, it is important to keep utilization at a low enough level to guarantee that the disk subsystem performs well enough to service core business applications.

Tactically, disk space is generally only a problem if an unexpected usage spike occurs. In that situation, keeping the system up and performing well becomes a top priority. Because it is often difficult to quickly identify what is happening when a spike happens, it often turns into an emergency situation where you have to get "all hands on deck" to solve the problem.

Centerfield provides help for both strategic and tactical needs. The DASD+ product provides a vast array of planning and management capabilities to handle day-to-day cleanup and reporting. It is a powerful way to handle the many different users of disk space and ensure they are managed properly with ease. Our disk/HUNTER tool services the tactical requirements. It monitors your system at all times and reports on disk spike issues in real time so they can be addressed with a minimum amount of IT resources or risk of downtime. To accomplish this, disk/HUNTER provides reports, messages, and e-mails to identify the root cause of the problem – by user, by job, and by object.

If you have the need to checkmate your disk issues, give us a call.

ALWCPYDTA:

If your SQL statement is running within a compiled program, another option to try is to change the ALWCPYDTA setting. On recent OS/400 releases the default is *OPTIMIZE. While this is the recommended setting, sometimes SQL statements actually run faster if *NO or *YES is used. Be cautious using this setting however, because all statements in the program will also use this option which may degrade their performance. If you are using ILE you can overcome this issue. To do that, simply isolate the problematic SQL statement into its own module, use the new compile option and then bind it into the original program.

OPTIMIZE FOR n ROWS:

Another alternative that may influence how the query optimizer joins tables, is the OPTIMIZE FOR n ROWS clause. Simply put this on the end of the original join query and plug in a number for 'n' (e.g. OPTIMIZE FOR 10 ROWS). Normally you want to put in the actual number of rows expected, but at times putting a very low or very high number actually works better.

QAQQINI options:

What do you do if you've done everything suggested above and the join order is still wrong? The answer is you can use the QAQQINI query options file to pick a particular join order. The FORCE_JOIN_ORDER parameter has two options that give you greater control - *YES and *PRIMARY. The *YES option says to choose an order that matches the one in the FROM clause. In other words, the tables are joined in the same order the SQL has them listed. The *PRIMARY option is a bit more specific. It says that you want to put a particular table first in the join order. By allowing you to choose the primary table, a large number of ordering problems can be solved without having to guess at the correct order for the rest of the tables. More information on use of QAQQINI can be found at the following links:

<http://www-03.ibm.com/servers/eserver/iseries/db2/books.html>

<http://www-03.ibm.com/servers/enable/site/bi/tuner/index.html>

Do you need a Total Audit Solution?

Audit everything, or just what you want, when you want. Now you can monitor and capture system AND database activity together for a total top-to-bottom centralized audit trail for iSeries.

Easy as 1,2,3. Pick your files, pick your fields, pick your conditions to monitor then set it and forget it!

Contact us, if you need to:

- Capture system and data base activity regardless of the method
- Track only fields you need, ignore the rest
- Capture audit data only if specific configured conditions are met

Additional functions you can configure!

- Require cascading electronic signatures for approvals
- Send notifications via email, SMS, pager or AS400 message
- Attach files like PDF's, Excel's or Word documents to specific audit records

Reader Input

Came across your newsletter while looking for a solution to a specific problem. It's fantastic. Thanks!

AK, Canada

Thank you for you newsletter as always i found it useful. In the last newsletter there was a very good explanation on VARCHAR columns on iSeries.

SM, Israel

I'm a novice at SQL and always enjoy your newsletter. There always seems to be at least one article that I can learn from.

This month I enjoyed both features (varlen and date/time sql functions-and how to use in RPGLE) and the shorter tip on RTV*. But the varlen article in particular was interesting. I'd just picked up a project that had multiple 60 and 75 byte character fields that looked like excellent candidates for varlen.

Since I didn't know much about them (this file or varlen), I decided a couple of tests were in order. I ran the SQL statements that told me the usage of the fields and two things surprised me. First, how quickly I'd get to 99% in some cases (they really did use the space!). Secondly, how many records weren't used at all. The file has about 22,000 records and the lowest number of empty records was 14,000 with some fields having over 20,000 empty

records. Doing some quick math (14000*60)-(22000*2) = about 796000 saved bytes in just one field. That would cover the two extra bytes for every varlen field in the file and I still had 16 more fields with even more empty records.

This started me drooling about how much disk space I was going to save and in some ways, how much better the performance would be! So, I created a duplicate source changing 17 fields to varlen with no allocated space. I then copied the data (CPYF) and checked the size of the files using dspfd. I was surprised to see the varlen file taking up 2 meg more total space. Then I did an unscientific study of performance by using query to display the whole file to the screen. The varlen file took about 20% longer to display the last record.

After letting that stew for a couple of hours, it dawned on me that simply copying the file probably didn't properly convert those fields. So, I drew up a little SQL to RTRIM those fields as they were being inserted. Voila! A 20 meg reduction in the file size (well over 50% in this case) and about a 20% speed improvement in my query test (still unscientific, but I could count the seconds out and see the difference).

Thanks for the good articles.
MK, Packerland USA



Make Your UDFs Scream

Here's the SQL 1999 standard definition of a User Defined Function (UDF).

An SQL-invoked function is an SQL-invoked routine whose invocation returns a value. Every parameter of an SQL-invoked function is an input parameter, one of which may be designated as the result SQL parameter. An SQL-invoked function can be a type-preserving function; a type-preserving function is an SQL-invoked function that has a result SQL parameter. The result data type of a type-preserving function is some subtype of the data type of its result SQL parameter.

And a similar IBM InfoCenter definition:

A function is an operation denoted by a function name followed by one or more operands that are enclosed in parentheses. It represents a relationship between a set of input values and a set of result values. The input values to a function are called arguments. For example, a function can be passed two input arguments that have date and time data types and return a value with a timestamp data type as the result.

Nothing new here, you pass in some arguments and you get a return value. You've been using SQL UDFs for as long as you've been using SQL. Remember MIN, MAX, AVG, SUM... etc? All of these are built-in functions. They are an inherent part of Data Manipulation Language (DML) and you use them automatically. I contend that this automatic use came about because all those built-in functions perform well. If they had not, you would have found another way to perform the same function, i.e. write it in RPG. ☺

Potential poor performance is probably the main reason why *System i* SQL developers shy away from creating their own UDFs. I consider this a sad state of affairs, as self-created UDFs can perform well and indeed will also surface much of

the same functionality we have delivered over the past 20-30 years of coding in RPG, COBOL, CL, C, etc. The important difference is that self-created UDFs make the functions available to *any* SQL-capable interface, hence making them available to the rest of the world.

The good folks at Big-Blue have been busy enhancing our favorite database, and I strongly urge you to revisit your UDFs keeping in mind some performance tips I'm about to outline.

When you create a UDF via the CREATE FUNCTION command, you can specify or not specify number of parameters. Any parameters not explicitly specified will be set to a default value. In general, IBM opts for safe defaults in keeping with an unstated policy of "do no harm". One seldom realized fact is that some of the parameters are decisive in determining how a given UDF is ultimately going to perform. Chief among these are FENCED, DETERMINISTIC and EXTERNAL ACTION. Let's examine each of them in more detail.

1) Use NOT FENCED (small performance gain)

FENCED is a default value on DB2 for *System i*. It causes a UDF to execute in a secondary thread and slows down the UDF's execution slightly. One "benefit" of the FENCED attribute is that it causes the UDF to execute in its own memory space, but that type of control is really only important on other platforms as we can trust the *System i* (yeah!) to handle memory for us while executing the UDF in the main thread.

Availability: V5R2 & SQE processed query

2) Use DETERMINISTIC every chance you get (large performance gain)

(Continued on page 5)

THIS SHOULD BE A NO BRAINER

Metadata: (Greek: meta-+data "information") means data about data.

Metadata – not exactly a cocktail hour buzz word, and not something most people look forward to analyzing – but it's the heart and soul of any worthwhile system & application performance analysis. We undertake the laborious process of metadata analysis frequently as part of our **database/ASSESSMENT** service offering. We do charge a nominal sum for this service, but we sure don't make any money when we do one.

You may want to have a **database/ASSESSMENT** performed when you need to know:

- If you have room for performance improvement on your system
- If you really need a hardware upgrade
- If you can use some of Centerfield's tools to postpone or make that hardware upgrade count.

Your part is easy. We send you a small collection utility and you send back the collected metadata. Many man-hours later, you sit down with us for a teleconference and review 100 or so pages of reports and recommendations already e-mailed to you.

(Make Your UDFs Scream—Continued from page 4)

If I'm reading the CREATE FUNCTION command diagram correctly, NOT DETERMINISTIC is the default. This is a sensible default as it guarantees that the UDF will execute its code every time DB2 invokes it. However, for some functions it really doesn't make sense for the UDF to execute every time. A good example is the frequent conversion of a date stored in NUMERIC 8 format into a true DATE data type, where every time you pass in 01012001 your UDF conversion routine will return a DATE value synonymous with '01-01-2001'. What you really want is for DB2 to call your function once per unique value of numeric date, cache it and subsequently just return the cached DATE value for that same input argument value. This is exactly what DETERMINISTIC does!

That said, some functions are inherently non-deterministic. For example, in the currency conversion function, where the value of the Dollar against the Euro can oscillate daily, you wouldn't want that type of UDF specified as DETERMINISTIC. Rather, you'd want it non-deterministic and possibly driven off some external input (i.e. Web service).

Availability: V5R3 & SQE processed query

The really interesting fact is that almost every time we perform a **database/ASSESSMENT**, we find something lacking within the customer's system. One would hope that with the large number of IBM and ERP vendor consultants at customer sites we wouldn't see as many easily addressed issues as we do, but we do.

What if we don't find anything to improve upon? That has value in itself! You just got a pat on a back from acknowledged DB2 experts that you're doing a fabulous job as a DBA and system administrator. Your management will take notice and it may help that personnel performance evaluation that's coming up soon.

What if a hardware upgrade will solve your problems? We'll tell you that. This is counter to our business plan as a tools vendor, but we always do what's best for the customer, not Centerfield.

As I see it, it is a no-brainer.



Elvis Budimlic
Development Director

3) Use **NO EXTERNAL ACTION** if at all possible (tremendous performance gain)

The *System i* default is EXTERNAL ACTION. This is another safe default, but as is often the case, here we may not need to trade performance for safety. Let me be clear, you need to specify EXTERNAL ACTION if you're sending a program message, data queue entry, writing a record to a file, sending an email etc. in your UDF. These are all examples of external actions and they would not be performed had you specified NO EXTERNAL ACTION.

However, whenever there is no external action performed in the UDF, NO EXTERNAL ACTION will yield a huge performance gain, as DB2 is then free to optimize your UDF.

Availability: Available since V4R4 and interrogated with the caching support for the deterministic attribute

Now that I've tickled your creative gene, recreate your UDFs and stored procedures specifying these parameters where appropriate and let me know what you find.

