Report Writer Speedup Tips: Part Two

By Tom Purnell

In the March 2004 issue of *Technical Support* magazine, we presented six speedup tips to help your report writer queries run faster. We saw how even minor changes in the way you write your query can have a big impact on how fast it runs. This month, we will look at more report writer speedup tips, which are related to efficient file processing.

LAST month I presented six speedup tips to help your report writer queries run faster. We saw how even minor changes in the way you write your query can have a big impact on how fast it runs. (For example, we saw a case where simply reversing the order of two selection tests reduced processing by 33%.)

This month, we will look at some more report writer speedup tips. This month's tips are related to efficient file processing.

Note: the code examples in this article are written in a generic "pseudo-code," not the actual syntax of any specific report writer. You should be able to adapt the tips in this article to almost any report writer.

FIELDS FROM KEYED AUXILIARY INPUT FILES

The tips in last month's article assumed that all fields used in the selection tests came from the same input file. However, a handy feature of most report writers is their ability to use data from the primary input file to create keys for reading related records from other files (or from DB2 tables).

When your selection logic uses fields from such "auxiliary" input files, you have another performance factor to consider. Since I/O is relatively slow, your goal is to reduce I/O to the minimum amount actually required to correctly produce your report. The way you write your selection tests can affect how much I/O has to be performed.

Speedup Tip 7: when your selection logic involves fields from auxiliary input files, put the tests involving those fields as near the end of the conditional expression as possible.

Let me explain why this can sometimes help. Some report writers are procedural. That is, the user writes code (maybe similar to a simplified COBOL) that tells the report writer what to do and when to do it. Other reports writers are non-procedural. That is, the user describes the results he wants, and the report writer determines the process it will use to produce those results.

Some non-procedural report writers are designed to perform I/O only at the point that data from a record is actually needed. That is, they do not just automatically read a record from each auxiliary input file every time a new primary input record is read (as might be the case with procedural code).

If your report writer is of this latter type, you can take advantage of its I/O deferral feature when you write your selection logic. Put the tests involving auxiliary input fields as late in your selection expression as you can. That increases the chance that those tests won't have to be performed at all. (See Speedup Tips 1 and 2 in Part One of this series in the March 2004 issue of *Technical Support* magazine.) If the report writer excludes a record based on the earlier tests (involving primary input fields), it will not need to examine the contents of the auxiliary fields. That means that it can skip the I/O to the auxiliary input file for that record.

If you can exclude a large percentage of records this way (based just on primary input file tests), your report writer may be able to avoid a lot of unnecessary I/O to the auxiliary file.

Let's consider an example using our large database of people. Assume that the database contains an ID number for each person. The ID number can be used as the key to another file that contains address information. Now assume that you want to include people in your report whose sex is male and who live in Chicago.

The harder test to pass is the city test. Thus, based on Speedup Tip 1, we would normally put that test first. However, the city field comes from an auxiliary input file. Therefore, in this case it is probably more efficient to write the IF statement this way:

```
IF FILE1.SEX='M' AND FILE2.CITY='CHICAGO'
```

In the above statement, 50% of the input file will be excluded based on the first test alone. That means that 50% of the time the I/O to the auxiliary address file can be avoided. This statement reduces the amount of I/O to the address file by half, compared with putting the city test first. (If the city test had been written first, the address record would have to be read 100% of the time.)

Here is another tip involving keyed auxiliary files (for those report writers that defer I/O until needed):

Speedup Tip 8: if you can perform a selection test using either a field from the primary input file or a field from an auxiliary input file, choose the primary file test.

For example, assume that you need to select all the records in the database from California and Arizona. Assume that the main database contains a two-byte state code. Since we also want to print the full state name in the detail lines of the report, the state code is used to read a record from a separate state file. So you have the following two ways you could write the selection tests:

```
1. IF FILE2.STATE-NAME='CALIFORNIA' OR 'ARIZONA'
2. IF FILE1.STATE-CODE='CA' OR 'AZ'
```

Obviously you would choose to test the state code (line 2 above) rather than the full state name. If you test the state name, the state file must be read for 100% of the input file (during the record selection phase). If you test the state codes, then the state file only needs to be read for the records that are actually selected and printed in the report (during the detail line formatting phase).

REPLACE FILE I/O WITH AN INTERNAL LOOKUP

Since random I/O to auxiliary files is slow, consider whether you can use an internal "lookup" instead of reading a record from a file.

Speedup Tip 9: try replacing a file lookup with internal lookup code instead.

For example, assume that your primary input file contains two-byte state codes. You want to print the entire state name in your report. One approach might be to read a record from a state file, using the state code as the key:

```
INPUT CUSTOMER-FILE
AUX   STATE-FILE READKEY(STATE-CODE)
PRINT LASTNAME ADDR CITY STATE-FILE.STATE-NAME ZIP
```

However, it is often faster to use a case-type construct to expand the state name, instead of reading it from a VSAM file:

```
INPUT CUSTOMER-FILE
COMPUTE NAME-OF-STATE =
  WHEN(STATE-CODE='CA') ASSIGN('CALIFORNIA')
  WHEN(STATE-CODE='TX') ASSIGN('TEXAS')
  ...
  ELSE                  ASSIGN('UNKNOWN')
PRINT LASTNAME ADDR CITY NAME-OF-STATE ZIP
```

The COMPUTE statement in the example above functions as a state lookup routine and eliminates the need for I/O to the state file. Notice that we ordered the state tests by population (not alphabetically), as discussed in the first article in this series.

In some cases, there will be too many possible values for such an internal lookup to be practical. Or the entries may change frequently, making maintenance of the code difficult. In that case, you might consider a combination approach. Use an internal lookup to efficiently satisfy the most common cases. Then resort to reading from the file only for those cases not handled by the lookup code:

```
INPUT CUSTOMER-FILE
AUX   STATE-FILE READKEY(STATE-CODE)
COMPUTE NAME-OF-STATE =
  WHEN(STATE-CODE='CA') ASSIGN('CALIFORNIA')
  WHEN(STATE-CODE='TX') ASSIGN('TEXAS')
  ...
  WHEN(STATE-CODE='WY') ASSIGN('WYOMING')
  ELSE                  ASSIGN(STATE-FILE.STATE-NAME)
PRINT LASTNAME ADDR CITY NAME-OF-STATE ZIP
```

In the above example, whenever the state code is one that is covered by a WHEN condition, no read will be performed to the state file. That is because even though there is a statement that defines the relationship to the state file, no data from that file would actually be needed (since the ELSE clause referring to the state file would not be executed). An efficient non-procedural report writer should not read a record from the state file in that case.

On the other hand, when a state code is encountered which is *not* covered by any of the WHEN parameters, the ELSE clause would assign the STATE-NAME field from the STATE-FILE. In that case (and only in that case) the report writer would need to read a record from the state file.

LIMIT THE KEYRANGE

Many report writers allow you to specify a "key range" parameter for a keyed input file. This limits the number of records read from the input file to just those records in the range that your report needs. If you do not use this parameter, the report writer has to read the entire VSAM file.

Speedup Tip 10: if your input is a keyed VSAM file, read only the records in the desired key range.

For example, assume that the input file for a run is a large KSDS (keyed) VSAM customer file. The key for this file is a two-byte state code followed by a 10-byte customer number. Assume that you want a report that lists all of the male customers in New York. Normally, you might write something like this:

```
INPUT CUSTOMER-FILE
IF STATE-CODE='NY' AND SEX='M'
```

In the above example, the report writer must read through the entire customer file, testing the state code and the sex field to determine which records to include in the report. However, since the key to this file happens to begin with the state code, we could take advantage of the report writer's key range option:

```
INPUT CUSTOMER-FILE KEYRANGE('NY')
IF SEX='M'
```

The above statements result in the same report but will run much faster. Instead of having to read every record in the customer file, the report writer can now jump in right at the first record whose key begins with NY. It then reads records sequentially from that point. And, after reading the last record whose key begins with NY, it stops reading the file altogether. This run is much faster because the report writer does not have to read the customer records for all of the other states and perform the IF tests on them.

Notice that in the second run we also dropped the state code test from the IF statement. Since the key range parameter guarantees that only records with NY in the state code are read, there is no need to test for that in the IF statement. Dropping this test provides an additional improvement in performance.

SKIP CLEARING THE I/O AREA

When processing files with variable length records, you (or your report writer) may be clearing the I/O area before each read. And I/O areas are often quite large—a thousand bytes or more—by default. You may have explicitly coded the clearing logic yourself. Or your report writer may just do it for you by default. This is to ensure that when a short record is read, it is not followed by leftover data from a previous longer record. Such leftover data could cause incorrect results in some cases.

On the other hand, you may know that your particular report only uses data from the fixed, early portion of each record. That leads to the next speedup tip.

Speedup Tip 11: if you are sure that any leftover data in the input area won't affect your report, save CPU time by not clearing the I/O area before each read from the input file.

Check to see if your report writer has an option to suppress automatic I/O area clearing.

INCREASE VSAM BUFFERS

Direct (random) reads to VSAM files are inherently slow. A single random read may involve multiple EXCPs (to read different levels of index blocks and then data blocks).

Speedup Tip 12: report writer performance can often be improved by increasing the number of buffers available for your VSAM files.

Having more buffers increases the chances that VSAM will find a needed record already in one of its buffers, eliminating the need for a disk access. Even for VSAM files that are read sequentially (the primary input file for your report), increasing the number of buffers may improve performance.

Your report writer may allow you to specify the number of buffers directly in your request. If not, you can specify VSAM buffers in the execution JCL. Just use the BUFNI and BUFND parameters in your DD or DLBL statement.

The BUFND parameter specifies the number of "data buffers" that the VSAM access method should use when processing the file. The BUFNI parameter specifies the number of "index buffers" to use.

The best value to use for these parameters depends on how the file is used in your query. The following suggestions are based on recommendations found in IBM's VSAM manual.

For the primary input file (read sequentially), increasing the number of data buffers to 4 or 5 should improve performance. At some point after that, excessive paging may cancel any benefit. (Increasing the number of *index* buffers from the default does not normally improve performance for sequential reads.)

For an auxiliary input file (read randomly), increasing the number of data buffers to 3 or 4 should improve performance. After that, more benefit is obtained by increasing the number of index buffers. Increasing the number of index buffers should improve performance for random reads up to a certain point. After that, excessive paging may cancel any benefit. Optimal performance is sometimes achieved by having one index buffer for each level of the file's index.

If you have long-running, VSAM-intensive jobs, you may wish to experiment to find the optimal number of buffers to use.

SPEED UP THE DEVELOPMENT CYCLE

The process of developing new reports often entails making minor changes and re-running a request over and over. If the input file you are using contains a million records, this can obviously take some time. Your report writer probably has some options intended to help speed up your development runs. (Later you just remove these options to obtain your full production results.)

Speedup Tip 13: be sure to use limiting options during development to speed up your trial runs.

See if your report writer has options that allow you to:

- limit the number of input file records read for the run
- stop reading the input file after a certain number of records have been selected
- print only a limited number of pages (or lines) of the report
- print only a limited number of detail lines per control break. This allows you to test and debug your control break processing, without having to print what could be voluminous detail data.

CONCLUSION

In this series of articles, I have presented 13 tips to help speed up the execution of your report writer queries. Is it worth the effort to implement these tips? Well, if you are reporting on a file that has 100 records in it, feel free to write the query any way you like! It won't make any measurable difference. But if your report comes from a very large file, and especially if it will run on a regular basis, it probably is worth the effort to implement some of these speedup tips. It may require just a little more thought. But the results will be worth it.

NaSPA member Tom Purnell is Product Manager for Spectrum Writer with Pacific Systems Group. This northern California-based software company specializes in 4GL report writers for OS/390 and VSE. Tom has over 25 years experience in mainframe software development. He has also worked as a volunteer teacher in a Russian orphanage, teaching basic computer skills to orphans. He can be contacted via e-mail at tpurnell@pacificsystemsgroup.com.