



# Report Writer Speedup Tips—Part One

By Tom Purnell

Did you realize that how you write your report writer queries has a big impact on how fast they run? Most mainframe shops have at least one report writer utility. Report writers can be a wonderful productivity tool for programmers. They help you quickly put together one-time, ad-hoc reports and are appropriate tools for developing production reports. This article presents techniques you should use when writing your report writer queries to help them run as fast as possible.

**DID** you realize that how you write your report writer queries has a big impact on how fast they run?

Almost all mainframe shops have at least one (and often several) report writer utilities. Report writers can be a great productivity tool for programmers. They are perfect for quickly putting together those one-time ad-hoc reports. And an efficient report writer is an appropriate tool for developing production reports. You should use some techniques when writing your report writer queries to help them run as fast as possible. This two-part series discusses several of these speedup tips for the report writer.

*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.

## SELECTING RECORDS EFFICIENTLY

The way you write your record selection logic can greatly affect how efficiently your job runs. After all, those selection tests are performed on every single record in the input file. Even when you just need a dozen records out of a million record file, the selection tests must be performed a million times.

With most report writers, record selection is specified by a *conditional expression*. The conditional expression identifies the records

from the input file to include in the report. Depending on the report writer, this might be specified via a keyword named IF, WHERE, SELECT, INCLUDE, etc.

The first three speedup tips are based on a fact about report writers that you may not even know about. A well-designed report writer *stops processing* a conditional expression as soon as it determines that the entire expression is either true or false. That is, the report writer does not always perform every test in your selection logic. By writing your conditional expressions so that the report writer makes its determination as early as possible, you will eliminate unnecessary processing.

## CONDITIONAL EXPRESSIONS USING AND

Let’s look at a simple example. Assume that you are processing a large database of people. You want to select all records where the sex is male and the last name is Alexander.

There are two ways you could write the conditional expression to select these records:

1. IF LASTNAME='ALEXANDER' AND SEX='M'
2. IF SEX='M' AND LASTNAME='ALEXANDER'

Many people might assume it makes no difference which of the above IF statements is used, since the results will be the same. And in terms of the report output, that’s true. Nevertheless, one of the above statements

would use 50% more CPU cycles than the other statement. Do you know which one?

In conditional expressions with ANDed tests, *all tests* have to pass before the expression is true. If any test fails, the whole expression also fails. That means that the report writer can stop processing the expression as soon as a single test fails. It does not need to perform the remaining tests. Thus, the speedup tip for conditional expressions with ANDed tests is:

**Speedup Tip 1:** when a conditional expression has multiple tests separated with AND, put the *most difficult* test first. Put the next-most-difficult test second, and so on. The “most difficult test” means the test that the most input file records will fail. The “next-most-difficult” test means the test that will be failed most often by those records that pass the first test, and so on.

In this example, one of the tests (SEX = ‘M’) should be passed by about half of the input records. (We are assuming that the database is representative of the population at large.) The other test (LASTNAME = ‘ALEXANDER’) will be passed by only a tiny fraction of the database (around six one-hundredths of one percent, based on a quick telephone directory check).

Thus, according to the speedup tip, we would choose to put the more “difficult” last name test first (as in line 1 above):

```
IF LASTNAME='ALEXANDER' AND SEX='M'
```

Using this statement, 99.94% of the input file records will fail the first test. For all those

records, the second test will not have to be performed. The report writer will exclude the input record with just a single test 99.94% of the time. It will perform the second test (SEX = 'M') only on 0.06% of the input records.

Now consider if we had used line 2, with the sex test first:

```
IF SEX='M' AND LASTNAME='ALEXANDER'
```

In this case, the report writer would have to perform both tests on approximately 50% of the input records (instead of just 0.06%). That is because the first test (SEX = 'M') will be passed by about half of the records in the input file. For that half of the file, the second test (LASTNAME = 'ALEXANDER') then has to be performed as well. When this second test is performed, most of the records will fail it and will thus fail the entire IF statement. The result is the same, but it took a lot more CPU cycles to get there.

Obviously, when processing large files, a small CPU saving in each record can add up. Assume that the database contains one million people. Using the first IF statement shown earlier, the report writer would have to perform about 1,000,600 tests to select the correct records from the file. (That is 1,000,000 last name tests, plus 600 sex tests.) Using the second IF statement, the report writer would have to perform 1,500,000 tests. (That is 1,000,000 sex tests, plus 500,000 last name tests.) You can see that the second IF statement has to perform almost 50% more selection tests than the first statement.

## CONDITIONAL EXPRESSIONS USING OR

A similar principle applies to IF statements that use "OR logic," but with one difference.

In conditional expressions with ORed tests, the whole expression is true if *any* of the individual tests are true. That means that the report writer can stop processing the expression as soon as a single test passes. It does not need to perform any of the remaining tests. Thus, the speedup tip for conditional expressions with ORed tests is:

**Speedup Tip 2:** when a conditional expression has multiple tests separated with OR, put the *easiest* test first. Put the next-easiest test second, and so on. The "easiest test" means the test that the most input file records will pass. The "next-easiest test" means the test that will be passed most often by those records which did not pass the first test, and so on.

For example, assume now that you want a query that includes all the people in the database where *either* the sex is male or the last name is Alexander.

The best way to write the IF statement is:

```
IF SEX='M' OR LASTNAME='ALEXANDER'
```

By using the above statement, the report writer will be able to select about 50% of the file after evaluating only the first test. It will only have to perform the second test on the other half of the file.

What if we had written the statement this way?

```
IF LASTNAME='ALEXANDER' OR SEX='M'
```

In this case, the first test would be false over 99% of the time. That means that the report writer has to go on to perform the second test on 99% of the input file (instead of just 50%). While both statements would select the same records for your report, the above statement would require 33% more CPU cycles to process than the earlier statement. (That is, just under 2,000,000 tests versus 1,500,000 tests.)

One common application of this tip comes when you are including records where a certain field is equal to any one of a number of values. For example:

```
IF TRANS-TYPE = 20 OR 30 OR 40
```

You will improve performance in such a case if you put the most common value first. For example, if the majority of transactions are type 40, you should put 40 first:

```
IF TRANS-TYPE = 40 OR 20 OR 30
```

## CONDITIONAL EXPRESSIONS USING BOTH AND AND OR

If your conditional expression uses both AND logic and OR logic, you can still apply these rules. Just break the expression down into smaller, logical clauses. Each clause should use only AND logic, or only OR logic.

**Speedup Tip 3:** when a conditional expression has mixed AND and OR logic, first optimize the innermost AND-only and OR-only clauses. Then, treating each of those clauses as single tests, optimize the next outer level of such clauses, and so on.

This is easier to understand with an example. Assume that we found the following con-

ditional expression in an existing report writer query:

```
IF (LASTNAME='ALEXANDER' AND SEX='M') OR  
(SEX='F' AND LASTNAME='JONES')
```

We want to optimize the selection logic for speed. This statement uses both AND and OR logic. Thus, the first step is to optimize the innermost clauses that contain only one type of logic. There are two such clauses in this example. (The two clauses within parentheses.) Each of these clauses just contains two tests separated by AND. Applying Speedup Tip 1, we know that the harder test should come first. We can see that the tests in the first clause are already in the correct order (with the more difficult last name test first):

```
(LASTNAME='ALEXANDER' AND SEX='M')
```

However, in the second clause, we need to reverse the order of the two tests. Fewer records will have LASTNAME = 'JONES' than will have SEX = 'F'. Therefore we should move the harder, last name test first:

```
(LASTNAME='JONES' AND SEX='F')
```

The two innermost clauses are now written to run as fast as possible. The next step is to optimize the next outer level of tests. We will treat the optimized inner clauses as if each clause were just a single test. Now we just have two logical tests, separated by OR. Using Speedup Tip 2 above, we want to put the easier test (clause) first. Since Jones is a more common last name than Alexander, we assume that we will find more females named Jones than we will find males named Alexander. Therefore, we decide to reverse the order of the clauses. Now the easier of the ORed clauses comes first.

Here is the rewritten selection logic, now arranged to run as fast as possible:

```
IF (LASTNAME='JONES' AND SEX='F') OR  
(LASTNAME='ALEXANDER' AND SEX='M')
```

## SERIES OF CONDITIONAL EXPRESSIONS

Often a report writer query will involve a series of mutually exclusive conditional expressions. The report writer may have its own special construct (perhaps called CASE, SELECT or WHEN). Alternatively, you might simply have a block of nested IF statements.

Consider this series of IF statements, used to expand state codes into state names.

```
IF STATE-CODE = 'AL' THEN STATE='ALABAMA'  
ELSEIF STATE-CODE = 'AK' THEN STATE='ALASKA'  
ELSEIF STATE-CODE = 'AZ' THEN STATE='ARIZONA'  
ELSEIF STATE-CODE = 'AR' THEN STATE='ARKANSAS'  
...  
ELSEIF STATE=CODE = 'WY' THEN STATE='WYOMING'  
ELSE STATE='UNKNOWN'  
ENDIF
```

This example shows the states being checked in alphabetical order, which certainly seems like a logical way to write the code. However, if run-time performance is what you want, you are better off arranging the IF statements so that the most common cases occur early in the list. That is because, obviously, once the correct state has been found, the remainder of the nested IF statements will not be executed.

**Speedup Tip 4:** in a series of exclusive conditional expressions, put the conditional expression that is most likely to be true first. Then put the conditional expression that is next most likely to be true, and so on.

The most common state code in the file will be CA, since California is the most populous state in the United States (and since we are assuming a representative nationwide database.) Thus the IF statement for California should be first in the list. That way, for 12% of all input records, only the very first IF statement has to be processed. Compare that to the alphabetical list where the first IF statement (for Alabama) would catch only 1.5% of the input records.

If we order the nested IF statements according to population, the list looks like this:

```
IF STATE-CODE = 'CA' THEN STATE='CALIFORNIA'  
ELSEIF STATE-CODE = 'TX' THEN STATE='TEXAS'  
ELSEIF STATE-CODE = 'NY' THEN STATE='NEW YORK'  
ELSEIF STATE-CODE = 'FL' THEN STATE='FLORIDA'  
...  
ELSEIF STATE=CODE = 'WY' THEN STATE='WYOMING'  
ELSE STATE='UNKNOWN'  
ENDIF
```

Now, for about one-third of the input file, the correct state will be found within the first four IF statements. (In the alphabetical list, the correct state would be found in the top four statements for less than 5% of the input records.) The report writer will have to evaluate all 50 IF statements for only 0.2% of the input records (for Wyoming, which coincidentally comes last in population as well as alphabetically).

Putting the IF statements in the above order ensures that the report writer performs the fewest overall number of state code comparisons, and thus ensures the best performance.

If you have a database that is not representative of the United States as a whole, you can probably get results that are even more dramatic. (And with less work.) For example, if the vast majority of people in your database live in your company's home state, just move that state to the top of the list and leave the rest in alphabetical order. That one simple change might reduce your state name processing by 90% or more. (Especially if your company happens to be located in Wyoming!)

Of course, series of conditional expressions are used for all sorts of things other than state name lookups. Often it will be hard to determine the precise probability of each case. Nevertheless, even if you are only able to identify the one or two most common cases, putting those first can result in a significant benefit.

## DO NOT EVALUATE THE SAME CONDITIONAL EXPRESSION MORE THAN ONCE

Here is yet another way to speed up evaluation of conditional expressions in your report writer requests.

**Speedup Tip 5:** if your request refers in multiple places to the same conditional expression (of two or more tests), save the value of that expression in a variable.

That is, assign the value of the common expression to a Boolean variable (or to a Y/N character field). Then use the variable wherever that expression is needed. This way the report writer only has to compute the value of the expression once, while using the result as many times as needed. Obviously, the longer and more complex the common expression is, the more benefit you will get from this tip.

For example, assume that your request contains a computation based on case-like conditions:

```
COMPUTE X =  
  WHEN((A=B OR C>D) AND E=1) ASSIGN(1.23)  
  WHEN((A=B OR C>D) AND E=2) ASSIGN(8.45)  
  WHEN((A=B OR C>D) AND E=3) ASSIGN(0.29)
```

You could improve performance by evaluating the common part of the conditional expressions (in the WHEN parameters) just once, saving the result in a Boolean variable, like this:

```
COMPUTE TEMP =  
  WHEN(A=B OR C>D) ASSIGN(TRUE)  
  
COMPUTE X =  
  WHEN(TEMP AND E=1) ASSIGN(1.23)  
  WHEN(TEMP AND E=2) ASSIGN(8.45)  
  WHEN(TEMP AND E=3) ASSIGN(0.29)
```

## DO NOT PERFORM THE SAME COMPUTATION MORE THAN ONCE

Use a similar technique if your request uses the same *computational* expression in more than one place.

**Speedup Tip 6:** if your request performs the same computation in multiple places, just compute its value once and save it in a variable. This reduces the amount of processing time spent on the calculations.


For example, assume that your request contains these COMPUTE statements:

```
COMPUTE X = ((B - C) * 100) / C + 0.02  
COMPUTE Y = ((B - C) * 100) / C + 0.09  
COMPUTE Z = ((B - C) * 100) / C + 1.57
```

Improve your performance by computing the common part of the expressions just once, like this:

COMPUTE TEMP = ((B - C) \* 100) / C  
 COMPUTE X = TEMP + 0.02  
 COMPUTE Y = TEMP + 0.09  
 COMPUTE Z = TEMP + 1.57

## CONCLUSION

In this article, I presented six tips for speeding up the execution of your report writer queries. Most of this month's tips involved conditional expressions and the best way to write them. As you write future queries (or modify existing slow-running queries), I hope you give some thought to these tips. A little time spent optimizing your conditional expressions for speed will pay off every time they run. Next month, I will conclude this series with seven more speedup tips for the report writer—all related to efficient file processing. 



*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](mailto:tpurnell@pacificsystemsgroup.com).*

## Test Your Report Writing Skill

Ready to try your own skill at implementing Speedup Tip 1?

Assume these facts. You will read a tape containing a very large transaction log. The log covers the whole month of February 2004. There were roughly 100,000 transactions each day. Assume that the transactions were evenly distributed around the clock. Your task is to make a report showing all the transactions that occurred on February 15 between 9:00 AM and 9:01 AM.

Question: which of the following IF statements would you expect to run faster? (The answer is below.)

- A. IF DATE=2/15/04 AND TIME>9:00 AND TIME<9:01
- B. IF DATE=2/15/04 AND TIME<9:01 AND TIME>9:00
- C. IF TIME>9:00 AND TIME<9:01 AND DATE=2/15/04

**Answer:** Choice B is the fastest way to perform the selection tests. It arranges the three tests in decreasing order of difficulty, as Speedup Tip 1 suggests. The date test comes first because it is the hardest test to pass. (Since the transactions were evenly distributed over the 29 days in February, only 1/29th of the records in the file will pass the date test.) The next-hardest test to pass is TIME < 9:01. (Since the transactions were spread evenly among the 24 hours of each day, approximately 9/24ths of the records will pass that test.) That leaves the TIME > 9:00 for last. It is the easiest test to pass. Over half of the records in the file (15/24ths) will pass that test.

*Supporting Enterprise Networks and Operating Environments*

# SUPPORT