December, 2002

Advisor Answers

SEEK vs. SQL-SELECT

VFP 8/7/6

Q: We are currently developing a function which has to search through 5 tables of about 100,000 records each. The tables have all the same fields in them and are all indexed using the same field. The tables are going to grow and be very huge. The function must search in all the tables for a single record or a range of records.

Is it best to use SEEK in each of the tables or to use SQL-SELECT? (In our tests, the SQL join is taking forever).

–-- Frédéric Garant (via Advisor.COM)

A: The answer to your question is "it depends." In fact, your question is a good example of why it's important to test with realistic data in a realistic network environment.

First, let's define the problem. Your question doesn't say what you want to do with the records when you find them. You mention a join, which suggests you want to copy all the matching records to another table (or, perhaps, process child records where the parent record meet certain criteria), but in that case, SEEK clearly isn't sufficient to do the job by itself.

For the sake of discussion, I'll assume that the goal here is to find and "touch" each matching record.

The second issue is dealing with multiple independent source tables, but in fact, that doesn't make a difference. If the tables are pretty much the same size, the same approach should work for each of them. (If you were dealing with tables of related data, as in a parent-child relationship, the situation would be different.)

As you point out, there are two principal strategies for finding every record that meets some criteria. You can search for them one at a time, or you can grab the entire group. In FoxPro, in fact, there are quite a few ways to accomplish the task, but experience suggests three basic programming constructs to explore.

The first possibility is to SEEK the first match, then use SCAN WHILE to find the rest of the matching records. The basic structure of this version is:

```
SELECT TheRightTable
SET ORDER TO TheRightOrder
SEEK TheValue
SCAN WHILE TheField=TheValue
  * Do whatever you need to do
ENDSCAN
```

The second possibility is to use the SQL-SELECT command to put all the matching records into a cursor, then process them from there. That looks like this:

```
SELECT ListOfFields ;
  FROM TheRightTable ;
  WHERE TheField = TheValue ;
  INTO CURSOR Result NOFILTER
SCAN
  * Do whatever you need to do
ENDSCAN
```

It's important to point out that these two solutions do *not* do the same thing. The SEEK/SCAN solution gives you access to the original records. The SQL-SELECT version gives you access to copies of the records. If the process you need to perform involves manipulation of the original records, the SELECT version must be modified to collect the record numbers of the original records and then access those records, along these lines:

```
SELECT RECNO() AS nRecNum ;
  FROM TheRightTable ;
  WHERE TheField = TheValue ;
  INTO CURSOR Result NOFILTER
SCAN
  SELECT TheRightTable
  GO Result.nRecNum
  * Do whatever you need to do
ENDSCAN
```

The third alternative, LOCATE and CONTINUE, does provide access to the original records. It looks like this:

```
SELECT TheRightTable
LOCATE FOR TheField = TheValue
DO WHILE NOT EOF()
  * Do whatever you need to do
  CONTINUE
ENDDO
```

Of course, all three solutions depend on having an index tag based on the field being matched. The SEEK/SCAN version requires the tag. The other two can operate without it, but will run significantly more slowly.

I suspected that different choices would be better in different cases, so to test them, I tried four possibilities. First, I used two different tables. The smaller table has about 73,000 records and totals about 14MB. The larger has about 1,160,000 records and totals about 224MB. With each table, I tested two cases: a single matching record and multiple matching records. (Due to the nature of my test data, the number of matches averaged between 11 and 12 for the smaller table, but about 200 for the larger table. The larger table was created by concatenating 16 copies of the smaller.)

Because testing doesn't exactly mimic reality, I needed to make some adjustments in order to get valid results. First, VFP loves to cache data, and will do so whenever possible. To cut down that effect, I did two things. First, I used SYS(3050) to cut VFP's memory allocation way down, so there wasn't much room for cached data. Second, I created a small executable that constantly replaced values in the table I was testing, so that indexes would have to be reloaded in the test program. I ran the value replacement program in the background while running my tests.

The results surprised me. With the large table, LOCATE/CONTINUE was always the fastest (that is, across multiple runs of my tests). With the smaller table, SEEK/SCAN came out the best, but SEEK/SCAN and LOCATE/CONTINUE had the same order of magnitude.

More important was what was slow. On the large table, with a single result, SEEK/SCAN was about an order of magnitude slower than the other two choices. With the smaller table, the SELECT version was about an order of magnitude slower than the others, regardless of whether there were multiple matches.

There are a couple of other factors I didn't include in my tests. The first is network traffic. I tested only with local data. If you're dealing with data on a network server, you need to test that way. Second, my test tables do not have index tags on DELETED(). If you normally do have those tags, make sure they're in place before you test. (For details on why such a tag can make a big difference, see Chris Probst's article in the May, 1999 FoxPro Advisor.)

The bottom line is, as the auto manufacturers like to say, your mileage may vary. You need to test the alternatives with your data on your network with the settings you use in your application. There are plenty of variables that can influence the results of speed tests.

A few things to keep in mind when you're testing. First, shut down other applications, especially things like virus scanners (which affect the speed of opening files) and email clients (which may start running in the middle of your test).

Make sure to test enough repetitions that the numbers you see are meaningful. In my tests, I searched for 1000 different values in each table, to be sure of having timing results that were large enough to compare.

If you're working with VFP 7, beware of a bug that can seriously impact timing tests. If any breakpoints are defined (not necessarily active, just defined), code is slowed down significantly. Be sure to clear all breakpoints before doing any timing tests.

Make sure your settings match the ones you'll actually be using. Look at things like DELETED, COLLATE, EXACT, and ANSI.

To help you design your test program, I've included my whole test package on this month's Professional Resource CD. The main program is TestSpeed.PRG, but you won't be able to run it as is. At a minimum, you need to substitute your field names. If you don't already have a large test data set, this is a good time to create one. Make sure it reflects the distribution of values you expect in the real data.

–Tamar