December, 1998

Advisor Answers

Counting Records Efficiently

Visual FoxPro 6.0/5.0/3.0 and FoxPro 2.x

Q: I want to find out how to count the number of records that meet certain conditions in a browse window that uses a one-to-many relationship. I don't want to use the COUNT command because, in a large table, it's slow.

For example, the SET SKIP TO topic of the FoxPro 2.x for Windows Help includes the following code:

```
CLEAR
CLOSE DATABASES
USE offices INDEX offices ORDER ono IN 0
USE customer INDEX customer ORDER ono IN 0
SELECT offices
SET RELATION TO ono INTO customer
SET SKIP TO customer
WAIT WINDOW "Scroll to see companies for each office" NOWAIT

BROWSE FIELDS offices.ono  :H='Office Number', ;
        offices.city  :H='Office.City', customer.comppany
```

How can I count the companies that appear in the Browse window for each office without using the Count command?

–Dr. Qasem A. Alsaleh, Kuwait

A: Let me start by summarizing the question I think you're asking (and, therefore, the one I'm going to answer). I believe that what you really want to know is how can you quickly count the records in a table that match a specified condition. Although, in your case, the condition is that they match a particular parent in a one-to-many relationship that happens to be displayed in a Browse, neither the one-to-many relationship nor the Browse changes the basic solution.

There are two fairly simple ways to count a group of records in FoxPro. (This is a good place to note that all of this information applies to every version of FoxPro from FoxPro 2.0 to Visual FoxPro 6.0, unless noted.) One is to use the COUNT command, while the other uses SQL-SELECT.

You're correct that COUNT can be slow and, in fact, before FoxPro 2.0, it almost always was quite slow. However, the addition of the Rushmore engine in FoxPro 2.0 meant that COUNT (and, in fact, any commands that take a FOR clause or react to SET FILTER) can be incredibly fast, if you set things up correctly.

To give you some idea how fast COUNT can be, I tested using a table with almost 300,000 records and a total table size of roughly 55MB. I issued a COUNT command where about 20% of the records matched the condition. The first time I issued the command, it took less than .4 seconds. After that, each occurrence took about .02

seconds. (Interestingly, even if I closed FoxPro and reopened it, the data remained cached at the operating system level and I didn't see the initial slow time again.) By contrast, an unoptimized COUNT on the same table churned for a long time and eventually brought up a warning that virtual memory was running low. While you may get different results depending on your hardware set-up, this should make it clear that COUNT doesn't have to be slow.

What does it mean to "set things up correctly?" Rushmore uses index tags to quickly find records that meet the specified conditions, whether those conditions are included in a FOR command or come from the current filter. It can use any open indexes, even the old-fashioned .IDX files, but note the word "open." Having an appropriate index doesn't help unless you open it. For this and many other reasons, I keep all my indexes in the table's structural .CDX file, which is always opened automatically when the table opens.

Even if they're open, though, Rushmore is very picky about which indexes it uses. It doesn't use filtered indexes, that is, those with a FOR clause in the INDEX command. Rushmore also doesn't use any indexes that include the NOT keyword as part of the index expression (although it happily optimizes conditions that begin with NOT). Don't include the table's alias in the key either. (That one's a good idea for other reasons as well – what happens if you open the table with a different alias?) Most importantly, for Rushmore to use an index, the index key must exactly match the expression in the command.

Let's look at some examples. Using the same FoxPro 2.x sample data, suppose we want to count all the customers in California. The Customer table has a tag with a key of State. To get Rushmore to help, we write the command like this:

```
COUNT FOR State = "CA"
```

Suppose the key for this index were UPPER(State) instead, to ensure that the tag was case-insensitive. Then, we'd need to write:

```
COUNT FOR UPPER(State)="CA"
```

When the condition has multiple parts, things get a little more complex. Again, the secret is to make sure that each individual expression matches an index tag.

The classic example for this doesn't occur in the sample data, but does in many applications. Suppose you have separate fields for first name and last name, say, cFirstName and cLastName. Suppose, further, that you have a single index tag called Name with a key of UPPER(cLastName + cFirstName). This makes sense because it's unlikely you'd want to order the table by first name in most situations, but ordering by last and then by first is quite common.

Now, what if you want to count all the people with a particular last name? You might write:

```
COUNT FOR UPPER(cLastName) = "SMITH"
```

However, Rushmore won't optimize that command since UPPER(cLastName) doesn't exactly match a tag. Instead, you either need to create a tag for UPPER(cLastName) or write your command to take advantage of the existing tag:

```
COUNT FOR UPPER(cLastName+cFirstName) = "SMITH"
```

As long as SET EXACT is OFF, you get the same results, but much faster.

Once you make sure to have all the appropriate tags and your expressions exactly match your index keys, there are still some things you can do to help Rushmore out.

First, it's important to put the matching key on the left-hand side of the expression. In most cases, Rushmore won't see it if it's to the right of the equal sign (or whatever comparison operator you use).

If you operate with DELETED set ON, it's like having a filter on the DELETED() function. You can help Rushmore out by adding an index tag on that function, like this:

```
INDEX ON DELETED() TAG IsDeleted
```

You'll probably never `SET ORDER TO IsDeleted`, but just having the tag means many operations can be speeded up. This is the single most common cause of slowdowns in applications I've seen. Even if no records are deleted, you *must* have a tag on DELETED() in order to fully enjoy the benefits of Rushmore. Without that tag, FoxPro doesn't know that there are no deleted records (even if you know better) and reads the entire table.

Finally, SET COLLATE affects optimization as well. For Rushmore to use an index, the current COLLATE setting must be the same as the one in effect when the index was created. In addition, indexes using the MACHINE collate sequence make things faster than any other sequence.

Back to your original question. In the example you show, the command you need is:

```
* Assume Customer is the current work area
COUNT FOR Ono = Offices.Ono
```

That gives you the number of customers for the current office, the one with the highlight in the Browse window. As long as you have an index tag for Ono in the Customer table, this should be blazingly fast.

If you really want to know how many customers each office has or you simply don't want to use the COUNT command, use SQL SELECT. This command puts the number of customers for each office, along with the office number, into a cursor:

```
SELECT Customer.Ono, COUNT(*) ;
  FROM Offices, Customer ;
  WHERE Offices.Ono = Customer.Ono ;
  GROUP BY 1 ;
  INTO CURSOR CustCount
```

If you only want to know the customer count for the current office, you can write it this way:

```
cOno = Offices.Ono
SELECT Ono, COUNT(*) ;
  FROM Customer ;
  WHERE Ono = cOno;
  GROUP BY Ono ;
  INTO CURSOR CustCount
```

This version creates a cursor with a single record, showing the office number and the customer count for that office.

I don't have the space here to explain each facet of these queries, but you can read about SELECT in the FoxPro Help or in any good FoxPro reference. (In fact, I wrote an article about it that appeared way back in the April '93 FoxPro Advisor.)

I think once you check your indexes and make sure you write commands to match them, you'll find that COUNT isn't too slow for your needs.

–Tamar