July, 2005

Advisor Answers

Ranking data

VFP 9/8/7

Q: I have a table with three fields: a name, a numeric value, and a position. I want to fill in the position field based on the numeric value. That is, the record with the lowest value should have 1 for the position field and so forth.

A: Before I talk about how to do this, it's important to point out that such a ranking is good only until you start changing data in the table. Essentially, you're taking a snapshot of the data and ranking it. While there are a couple of ways to keep such a field up-to-date, for a table of any size, the performance cost would be much too high. Instead, plan to recompute the rank each time you need it.

On to solving the problem. As I thought about your problem, I came up with four basic ideas to solve it: one pure Xbase, two nearly-pure SQL and one hybrid. I decided to try them all; code for each is included on this month's Professional Resource CD.

I created a table, Ranking, with three fields: cName, nValue and nRank, with index tags for all three. I populated the cName and nValue fields with random values, adding a total of nearly 80,000 records.

I'll start with the Xbase answer, RankXbaseOnly.PRG on the PRD. This approach works in any version of FoxPro back to FoxPro 2.0; with minor changes, it goes back to FoxBase. The basic idea is to set order to the nValue field and loop through the table, assigning the rank to each record:

```
USE Ranking
SET ORDER TO nValue
nPos = 0
SCAN
   nPos = nPos + 1
   REPLACE nRank WITH m.nPos
ENDSCAN
```

The second approach (RankHybrid.PRG on the PRD) uses a couple of queries to determine the rankings and then switches to Xbase with a relation and REPLACE to update the original table:

```
SELECT cName, nValue ;
   FROM Ranking ;
   ORDER BY nValue ;
   INTO CURSOR Ordered NOFILTER

SELECT cName, nValue, RECNO() as nRank ;
   FROM Ordered ;
   INTO CURSOR Ranked

INDEX on cName TAG cName

SELECT Ranking
SET RELATION TO cName INTO Ranked
REPLACE ALL nRank WITH Ranked.nRank IN Ranking
SET RELATION TO
USE IN Ranked
USE IN Ordered
```

You can't find the rank of a record with a single query. You need to put the records in the right order first and then use their record numbers to determine their rank. Ordinarily, RECNO() isn't a good choice in a query, but as long as only one table is listed, and you don't specify the alias in the call to RECNO(), you get accurate results.

With a tag on nValue, you need the NOFILTER keyword in the first query. This prevents VFP from taking a shortcut and simply looking at the original table. With a tag on nValue and without NOFILTER, the results are wrong; each record is simply assigned its own record number as a rank.

The other two solutions, which are SQL-based, work only in VFP 9. They take advantage of the ability to list additional tables in the SQL UPDATE command. The first of them (RankQueryAndUpdate.PRG on the PRD) performs the same pair of queries as in the hybrid approach, but then joins that table with the Ranking table in UPDATE:

```
SELECT cName, nValue ;
   FROM Ranking ;
   ORDER BY nValue ;
   INTO CURSOR Ordered NOFILTER

SELECT cName, nValue, RECNO() as nRank ;
   FROM Ordered ;
   INTO CURSOR Ranked

UPDATE Ranking ;
   SET nRank = Ranked.nRank ;
   FROM Ranked ;
   WHERE Ranking.cName = Ranked.cName
```

```
USE IN Ranked
USE IN Ordered
```

In VFP 9, the SQL UPDATE command (not to be confused with the obsolete Xbase UPDATE command) gained a FROM clause that lets you specify additional tables and join them to the table being updated. The replacement data can come from those extra tables, as in the example. (You might be able to do the same thing in earlier versions by using SEEK() or LOOKUP() in the UPDATE command, but such code is likely to be difficult to maintain.)

There are two ways to join the additional table(s) to the table being updated. Like this example, you can simply list the additional table(s) and use the WHERE clause to specify a join. If you prefer, you can list the update table again in the FROM clause and use an ON clause to specify the join. In my tests, there were no significant performance differences between the two.

The final approach uses another SQL capability new to VFP 9: derived tables. Rather than running the two queries before using the result in UPDATE, this version (RankUpdateSubquery.PRG on the PRD) combines the whole thing into a single UPDATE command:

```
UPDATE Ranking ;
   SET nRank = Ranked.nRank ;
   FROM (SELECT *, RECNO() AS nRank ;
         FROM (SELECT cName, nValue ;
            FROM Ranking ;
            ORDER BY nValue) Ordered) Ranked ;
   WHERE Ranking.cName = Ranked.cName
```

A derived table is a query in the FROM clause. This command has two of them. Ranked is created in the FROM clause of the UPDATE command. Before that happens, Ordered is created in the FROM clause of the query that creates Ranked.

Once I had all four versions working, I tested their performance. To my initial surprise, the Xbase only solution was about twice as fast as all the rest. I also tried a few variations of the other solutions; you'll find them on the PRD as RankHybrid2.PRG, RankQueryAndUpdate2.PRG, RankQueryAndUpdate3.PRG, and RankUpdateSubquery2.PRG.

My test program (RankSpeedTestLoop.PRG on the PRD) runs each version multiple times, while tracking how long it takes. The version shown here tests only the original four solutions; the version on the PRD tests all eight.

```
#DEFINE PASSES 10

LOCAL nStart, nEnd, nPass
LOCAL aTests[4,2]

aTests[1,1] = "Xbase only"
aTests[1,2] = "RankXbaseOnly"
aTests[2,1] = "Hybrid"
aTests[2,2] = "RankHybrid"
aTests[3,1] = "Query and Update"
aTests[3,2] = "RankQueryAndUpdate"
aTests[4,1] = "Update with Subquery"
aTests[4,2] = "RankUpdateSubquery"

SET TALK OFF

FOR nTest = 1 TO ALEN(aTests,1)
   nStart = SECONDS()
   FOR nPass = 1 TO PASSES
      * Clean up first
      CLOSE TABLES ALL
      DO (aTests[nTest, 2])
   ENDFOR
   nEnd = SECONDS()
   ?"Using", aTests[nTest,1],", total for", PASSES, ;
    " passes = ", nEnd-nStart
ENDFOR
```

There are two lessons from this exercise. The first is the familiar one that with FoxPro, there's generally more than one way to get something done. The second is that sometimes, the old ways are best.

–Tamar