November, 1997

ASK ADVISOR

FoxPro 2.x and Visual FoxPro

Q: We have a rate schedule where the user specifies rates using one or more qualifiers to look up the rate. Assume the structure of the Rate table is:

```
Rate_Id I
Qualifier1 I
Qualifier2 I
Qualifier3 I
nRate Y
```

Any of the qualifiers may have "Don't care" specified. For example, rate 1 might require a value of 1 for the first and third qualifiers, but accept any value for qualifier two. The record for this rate would contain:

```
Rate_Id            1
Qualifier1   1
Qualifier2   0
Qualifier3   1
nRate        50
```

What's the best way to find an applicable rate for a given set of qualifiers? The user enters data for all the qualifiers, not knowing which one(s) should be ignored. Currently, we're using INLIST() with LOCATE. For example, if the user specifies Qualifier1=1, Qualifier2=3 and Qualifier3=1, we search with:

```
LOCATE FOR INLIST( Qualifier1, 1, 0) ;
       AND INLIST( Qualifier2, 3, 0) ;
       AND INLIST( Qualifier3, 1, 0)
```

We have keys on all the columns. However, some of these schedules get pretty large and this lookup needs to be instantaneous.

One idea we had is to create a character key value using BINTOC() and use "???" for the don't cares. But, we're still not sure how to SEEK this record.

Any ideas?

–Mark A. Nadig (Via Internet)

A: This is a knotty problem and I had to experiment with a variety of options before finding a couple that are faster than what you're doing now.

For test data, I created a table with 40,000 records with the qualifiers (sort of) randomly filled with values from 1 to 2000. A quarter of the records have 0 for qualifier 1, a quarter had 0 for qualifier 2, a quarter had 0 for the third qualifier, while the last quarter had no 0's. I created index tags for all three qualifier columns, as well as a combined tag, using BINTOC():

```
index on ;
```

```
   bintoc(qualifier1)+bintoc(qualifier2)+bintoc(qualifier3) ;
   tag combined
```

To test the search, my test program creates a cursor (TestVals) and fills it with a specified number of records (I tested both 500 and 5000) with random values from 1 to 2000 for the qualifiers. (An alternate version generates a random number between 1 and 40000 and then copies the data from that record in the Rates table, replacing any 0's with random values.)

I tested four basic alternatives for finding qualifying records and found two of them to be substantially faster than the others. The first is the combination of LOCATE and INLIST() you're already using. The second approach takes advantage of the fact that there are only two acceptable values for each qualifier, 0 and the input value, so we explicitly test those two choices using LOCATE. I tried two variations on this theme:

```
LOCATE FOR (Qualifier1=0 OR Qualifier1=TestVals.nQual1) ;
           AND (Qualifier2=0 OR Qualifier2=TestVals.nQual2);
           AND (Qualifier3=0 OR Qualifier3=TestVals.nQual3)
```

and

```
LOCATE FOR (Qualifier1=TestVals.nQual1 OR Qualifier1=0 ) ;
           AND (Qualifier2=TestVals.nQual2 OR Qualifier2=0);
           AND (Qualifier3=TestVals.nQual3 OR Qualifier3=0)
```

I tested both forms of the conditions in case one direction or the other was faster. In fact, the original LOCATE with INLIST() was slightly faster than these two, which were almost identical. (The second form seemed slightly faster than the first, in most cases.)

The other two versions took entirely different approaches. Since it seems likely that there could be multiple matches for a given set of qualifiers, putting the results in one place seemed useful. So, the fourth version uses a SELECT into a CURSOR:

```
SELECT * FROM Rates ;
   WHERE INLIST(Qualifier1,TestVals.nQual1,0) ;
     AND INLIST(Qualifier2,TestVals.nQual2,0) ;
     AND INLIST(Qualifier3,TestVals.nQual3,0) ;
   INTO CURSOR Junk
```

This version was from one and a half to two times as fast as the LOCATE FOR INLIST() version, with the added benefit of creating a cursor containing all the matches. I didn't test a query with the OR conditions, but I'd expect it to give results similar to the INLIST() query.

Finally, I followed up on your suggestion of using SEEK. There's no way to use a single SEEK to find all the potential matches for a set of qualifiers because any of the three could be replaced by a 0 (in fact, I assumed that even two of the three could be replaced). It's necessary to use a series of SEEKs to find all the potential matches. Here's the code that sets up and performs the SEEKs.

```
nVal1 = BINTOC(TestVals.nQual1) + BINTOC(TestVals.nQual2) + ;
        BINTOC(TestVals.nQual3)
nVal2 = BINTOC(0) + BINTOC(TestVals.nQual2) + ;
        BINTOC(TestVals.nQual3)
```

```
nVal3 = BINTOC(TestVals.nQual1) + BINTOC(0) + ;
        BINTOC(TestVals.nQual3)
nVal4 = BINTOC(TestVals.nQual1) + BINTOC(TestVals.nQual2) + ;
        BINTOC(0)
nVal5 = BINTOC(0) + BINTOC(0) + BINTOC(TestVals.nQual3)
nVal6 = BINTOC(0) + BINTOC(TestVals.nQual2) + BINTOC(0)
nVal7 = BINTOC(TestVals.nQual1) + BINTOC(0) + BINTOC(0)
SEEK nVal1
SEEK nVal2
SEEK nVal3
SEEK nVal4
SEEK nVal5
SEEK nVal6
SEEK nVal7
```

This version was almost two-and-a-half times faster than the LOCATE FOR INLIST() approach. However, it raised the question of what to do with the results. It seems to me there are two choices. Either we stop as soon as we find a match or we need to place all the matching records into a cursor for further processing. I tested both approaches. Here's the code that stops as soon as a match is found (the code to set nVal1 through nVal7 is the same as above):

```
IF NOT SEEK(nVal1)
   IF NOT SEEK(nVal2)
      IF NOT SEEK(nVal3)
         IF NOT SEEK(nVal4)
            IF NOT SEEK(nVal5)
               IF NOT SEEK(nVal6)
                  SEEK nVal7
               ENDIF
            ENDIF
         ENDIF
      ENDIF
   ENDIF
ENDIF
```

and here's the version that stores all the matching records to a cursor (previously created with a single field for the rate id):

```
SEEK nVal1
SCAN WHILE Qualifier1 = TestVals.nQual1 AND ;
           Qualifier2=TestVals.nQual2 AND ;
           Qualifier3=TestVals.nQual3
   INSERT INTO Junk VALUES (Rates.Rate_Id)
ENDSCAN

SEEK nVal2
SCAN WHILE Qualifier1 = 0 AND Qualifier2=TestVals.nQual2 ;
           AND Qualifier3=TestVals.nQual3
   INSERT INTO Junk VALUES (Rates.Rate_Id)
ENDSCAN

SEEK nVal3
SCAN WHILE Qualifier1 = TestVals.nQual1 AND Qualifier2=0 ;
           AND Qualifier3=TestVals.nQual3
   INSERT INTO Junk VALUES (Rates.Rate_Id)
ENDSCAN

SEEK nVal4
```

```
SCAN WHILE Qualifier1 = TestVals.nQual1 AND ;
           Qualifier2=TestVals.nQual2 AND Qualifier3=0
   INSERT INTO Junk VALUES (Rates.Rate_Id)
ENDSCAN

SEEK nVal5
SCAN WHILE Qualifier1 = 0 AND Qualifier2=0 AND ;
           Qualifier3=TestVals.nQual3
   INSERT INTO Junk VALUES (Rates.Rate_Id)
ENDSCAN

SEEK nVal6
SCAN WHILE Qualifier1 = 0 AND Qualifier2=TestVals.nQual2 ;
           AND Qualifier3=0
   INSERT INTO Junk VALUES (Rates.Rate_Id)
ENDSCAN

SEEK nVal7
SCAN WHILE Qualifier1 = TestVals.nQual1 AND Qualifier2=0 ;
           AND Qualifier3=0
   INSERT INTO Junk VALUES (Rates.Rate_Id)
ENDSCAN
```

The version that stops when it finds a match is sometimes faster and sometimes slower than the original SEEK approach. Presumably this is partly a function of how quickly it finds a match. (When TestVals was filled with records that always had a match, the IF SEEK() version was faster than the seven SEEKs.) I suspect there are also some speed differences between the SEEK command and the SEEK() function, since the function must also test for success. This would account for the IF SEEK() version being slower when relatively few records find a match.

Finally, I expected the version that creates a cursor to be a lot slower than the seven SEEKs, but in fact, it was only a little bit slower, not enough to rule it out.

Your question didn't indicate whether you actually have exactly three qualifiers or whether there are, in fact, a lot more. You also didn't specify whether more than one qualifier can be ignored at a time. If so, as the number of qualifiers increases, the number of SEEKs needed to find all the combinations goes up geometrically, so it's possible that with enough qualifiers, the SELECT version may in fact be a better choice. In order to help you check against your actual data, both the program to generate the test data and the actual testing program are included on this month's Companion Resource Disk.

–Tamar