December, 2005

Advisor Answers

Match multiple items in a query

VFP 9/8/7

Q: I have a form with a multi-select listbox. After a user chooses some items, I want to find all the records in a table that have any of the selected values in a specified field. I know how to substitute one value into a query, but how can I handle a list of values, when I don't know how many there might be in the list?

A: There are actually several ways to handle this problem. The first, building a query with a long series of ORs in the WHERE clause, would get pretty ugly, so we won't even consider that option. Instead, we'll look at two choices. The first is to use the IN operator against a list of values; the second is to put the matching values into a cursor and use that in the query. Figure 1 shows a form that lets you try both options, showing orders that include any of the selected products.

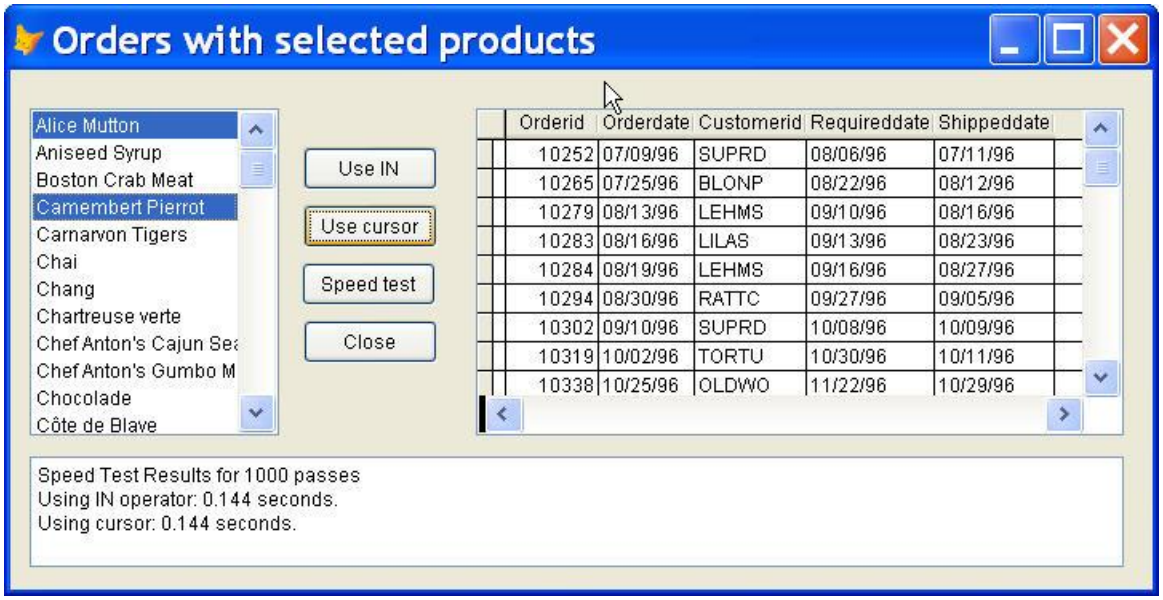

Figure 1. Looking for matches to a set of values—This form demonstrates both using the IN clause and putting the values to match into a cursor.

The WHERE clause of SQL SELECT offers the IN operator to match an expression against any of a list of values. For example, the following query finds customers in North America (all the examples here use the Northwind database):

```
SELECT CompanyName ;
   FROM Customers ;
   WHERE UPPER(Country) IN ("USA", "CANADA", "MEXICO") ;
   INTO CURSOR NACustomers
```

The trick in your case, then, is to build the list of values and substitute into the query. Here's the code in the custom UseIN method of the form (called from the Click method of the Use IN button), which does exactly that:

```
LOCAL cListValues, nIndex

* Build a list of items
cListValues = ""

WITH ThisForm.lstProducts as ListBox
   FOR nIndex = 1 TO .ListCount
      IF .Selected[ nIndex ]
         cListValues = cListValues + "," + ;
                       .List[ nIndex, 2 ]
      ENDIF
   ENDFOR
ENDWITH

cListValues = SUBSTR(cListValues, 2)

* Run the query
IF NOT EMPTY(cListValues)
   SELECT OrderID, OrderDate, CustomerID, ;
          RequiredDate, ShippedDate ;
      FROM Orders ;
      WHERE OrderID IN ;
         (SELECT OrderID ;
            FROM OrderDetails ;
              WHERE ProductID IN (&cListValues)) ;
      ORDER BY OrderID ;
      INTO CURSOR ProductsOrdered
ENDIF
```

The first section of code loops through the list, adding each selected item to a comma-separated string. The second section runs the query, using the macro operator (&) to substitute the string.

This solution has a couple of limitations. The most important relates to the IN operator. In VFP 8 and earlier, the list of items with IN is limited to 24. In VFP 9, the limit has been raised, but is restricted by the value of SYS(3055). In addition, macro-substituted strings are limited to 8,192 characters. In VFP 8 and earlier, the limit on IN is likely to hit you first, but in VFP 9, depending on the data you're matching, the macro limit could be a significant problem. However, when the overall list of items is small, using IN is a viable solution.

The second option isn't affected these limits. Instead of building a string, put the selected values into a cursor and use that cursor in the query. Here's the code in the form's custom UseCursor method (called from the Use cursor button's Click method):

```
LOCAL cListValues, nIndex

* Build a list of items
CREATE CURSOR ProductsChosen (ProductID I)

WITH ThisForm.lstProducts as ListBox
   FOR nIndex = 1 TO .ListCount
      IF .Selected[ nIndex ]
         INSERT INTO ProductsChosen ;
            VALUES ( VAL(.List[ nIndex, 2] ))
      ENDIF
   ENDFOR
ENDWITH


* Run the query
IF RECCOUNT("ProductsChosen") > 0
   SELECT OrderID, OrderDate, CustomerID, ;
          RequiredDate, ShippedDate ;
      FROM Orders ;
      WHERE OrderID IN ;
         (SELECT OrderID ;
            FROM OrderDetails ;
              JOIN ProductsChosen ;
                ON OrderDetails.ProductID = ;
                   ProductsChosen.ProductID ) ;
      ORDER BY OrderID ;
      INTO CURSOR ProductsOrdered
ENDIF
```

In this case, the first section of the code loops through the list, adding the primary key of each selected item to a cursor. Then, the second section runs the query, joining the newly created cursor to existing tables to restrict results to those orders containing the selected items.

You may be surprised by the use of a subquery in both cases; its purpose is to get a single list of the relevant orders, which are then pulled out from the Orders table. If the whole process is performed in the main query (whether it's with IN or a join), the same order may appear more than once in the results.

In my testing, there's no performance difference between the two methods. Use the Speed test button to check performance on your machine. The code is set up to run each technique 1000 times, and provide feedback every 100 passes, but you can change the values of

the variables nPassCount and nIncrement in the SpeedTest method (called from the Speed test button's Click method) to try other combinations:

```
WAIT WINDOW "Testing speed of two approaches" NOWAIT

LOCAL nPass, nStart, nEnd, nINTime, nCursorTime
LOCAL cMessage, nPassCount, nIncrement

nPassCount = 1000
nIncrement = 100

WAIT WINDOW "Testing IN operator" NOWAIT
cMessage = "IN operator: Pass "

nStart = SECONDS()
FOR nPass = 1 TO nPassCount
   IF MOD(nPass, nIncrement) = 0
      This.edtSpeedTest.Value = cMessage + ;
         TRANSFORM(nPass)
      This.edtSpeedTest.Refresh()
   ENDIF
   This.UseIN()
ENDFOR
nEnd = SECONDS()
nINTime = nEnd - nStart

WAIT WINDOW "Testing cursor" NOWAIT
cMessage = "Cursor: Pass "

nStart = SECONDS()
FOR nPass = 1 TO nPassCount
   IF MOD(nPass, nIncrement) = 0
      This.edtSpeedTest.Value = cMessage + ;
         TRANSFORM(nPass)
      This.edtSpeedTest.Refresh()
   ENDIF
   This.UseCursor()
ENDFOR
nEnd = SECONDS()
nCursorTime = nEnd - nStart

This.edtSpeedTest.Value = "Speed Test Results for " + TRANSFORM(nPassCount) +
" passes" + CHR(13) + CHR(10) + ;
   "Using IN operator: " + ;
   TRANSFORM(nINTime) + " seconds." + ;
   CHR(13) + CHR(10) + ;
   "Using cursor: " + TRANSFORM(nINTime) + " seconds."
This.edtSpeedTest.Refresh()
WAIT CLEAR
```

The example form in Figure 1 is included on this month's Professional Resource CD.

–Tamar