April, 2005

Advisor Answers

ASCAN() and queries

VFP 9/8/7

Q: Why doesn't the flag parameter for ASCAN(), allowing an exact match, work in the WHERE clause of a SQL command? It works in BROWSE, but doesn't seem to work in queries.

–Tom Ralston (Fort Myers, Florida)

A: The short answer to your question is that SET EXACT doesn't affect SQL commands. For those commands, you can get similar results using SET ANSI.

The longer answer starts with a look at the ASCAN() function, which allows you to search in an array. Two parameters added in VFP 7 give you control over several aspects of the search. Here's the syntax for ASCAN():

```
nResult = ASCAN( ArrayName, uExpression [, nStart
               [, nNumElems [, nColumn [, nFlags ] ] ] ] )
```

The first four parameters have been essentially unchanged since this function was added to the language. ArrayName is the name of the array in which you want to search. uExpression is the value for which you're searching.

nStart and nNumElems determine what portion of the array is searched. Both are measured in actual elements of the array, not rows or columns. Keep in mind that while you can address arrays with rows and columns, internally VFP sees every array, whether you declare it as one-dimensional or two-dimensional, as a single list of items. So an nStart value of 4 means to start with the fourth item in the array, whether that's the fourth item in a one-dimensional array, the second item in the second column of an array with two columns, the second item in the first column of an array with three columns, or the fourth item in the first row of an array with four or more columns. nNumElems indicates how many elements to search, regardless of the row and column structure of the array. You can specify –1 for these parameters to omit them and search the entire array; this allows you to specify the nColumn and nFlags parameters without limiting the search to particular elements.

The last two parameters were added in VFP 7. nColumn indicates which column of the array to search. You can combine this parameter with nStart and nNumElems to search only part of a single column, but as always, nStart and nNumElems refer to elements. Like nStart and nNumElems, you can specify –1 for nColumn to indicate all columns.

The nFlags parameter is the one giving you trouble. Using an additive scheme, it offers three capabilities: case-insensitive searching, local control of EXACT, and returning the row number, rather than the element number.

Case-insensitive searches and returning the row number are pretty straightforward. Add the indicated value (1 for case-insensitive, 8 for row number) to nFlags and you're set.

Controlling EXACT is more complex. By default, ASCAN() uses the current setting of SET EXACT. Two values of the flag combine to let you change that. Add 4 to indicate that you want to control EXACT locally; once you do so, add 2 if you want EXACT ON. That is, specifying nFlags as 4, 5, 12 or 13 indicates that the current SET EXACT setting should be ignored and this search should assume EXACT is OFF. Specifying 6, 7, 14 or 15 indicates that the SET EXACT setting should be ignored and this search should assume EXACT is ON.

Now that we know how to control EXACT for ASCAN(), what does that mean? EXACT determines the way VFP matches strings. Probably because it originated as an interactive tool for data manipulation, FoxPro makes it easy to do partial string matching. When EXACT is OFF, VFP matches strings only to the end of the right-hand string. For example, the following displays .T. in VFP:

```
SET EXACT OFF
?"Smith" = "S"
```

Setting EXACT ON means that strings must match exactly. This code displays .F.:

```
SET EXACT ON
?"Smith" = "S"
```

but this is .T.:

```
SET EXACT ON
?"Smith" = "Smith"
```

Now we come to the crux of your problem. EXACT, whether controlled globally by the SET EXACT command, or locally within an ASCAN() call,

affects only Xbase commands, like SEEK, REPLACE and BROWSE. It has no impact on VFP's SQL commands: SELECT, INSERT, DELETE and UPDATE. There's a similar setting, ANSI, that controls string matching in those commands.

ANSI is controlled by the SET ANSI command. Its effect is slightly different than that of EXACT. When ANSI is OFF, strings are matched to the end of the shorter string, no matter which side of the comparison it's on. So, in a query, with ANSI OFF, "Smith"="S" and "S"="Smith" are both true. When ANSI is ON, the strings must match exactly.

There's one more player in all this: the "==" operator, which you can read as "exactly equals." When you're dealing with US English and the standard ("Machine") collation sequence, "==" works the same as setting EXACT ON for Xbase commands with one exception. (I'll leave the behavior of = and == as collation sequences and codepages vary for a future column.) The single exception is trailing blanks. With SET EXACT ON, two strings that differ only in trailing blanks are considered the same by =, but for ==, the length of the strings must match as well as the content. So, this example returns .T.:

```
SET EXACT ON
?"Smith  " = "Smith"
```

but this one returns .F.:

```
?"Smith  " == "Smith"
```

I prefer to use "==" because it documents locally what I'm doing, and doesn't affect any commands other than the one using it. In SQL commands, no matter how ANSI is set, "==" ignores trailing blanks, so "Smith" == "Smith   " in a query.

With all this background, how can you get what you want? You have a list of items and you want to find all the records in a table that exactly match something on the list. There are several possibilities.

Your first choice, given that your list is in an array, is to SET ANSI ON before the query and then restore its value afterward, like this:

```
LOCAL cOldAnsi

cOldAnsi = SET("ANSI")
SET ANSI ON

SELECT * ;
```

```
  FROM YourTable ;
  WHERE ASCAN(aYourArray, SomeField, -1, -1, 1, 1) > 0

SET ANSI &cOldAnsi
```

Another choice is to use a cursor instead of an array for your list and then use the "==" operator:

```
SELECT YourTable.* ;
  FROM YourTable ;
    JOIN YourList ;
      ON YourTable.SomeField == YourList.SomeField
```

If you're not using the array for anything else, this would be my preference. Even if you are, it's easy to copy an array into a cursor using APPEND FROM ARRAY, so I might go this way anyway.

If either the original table or the list can be large, it's probably worth coding both versions and doing some speed testing.

–Tamar