March, 1999

Advisor Answers

UDFs in Queries

VFP 6.0/5.0/3.0 and FoxPro 2.x

Q: I have run across what appears to be a bug in VFP5.0. Consider the following statement:

```
SELECT Product_ID ;
FROM Products ;
WHERE 10 = IIF( on_order > 0, mod( 10, on_order), 0) ;
INTO CURSOR Query
```

Products comes from the VFP 5 sample data. When I run this, I get a result set with 13 records.

Now replace the built-in MOD() function with an equivalent UDF:

```
FUNCTION Mod2
LPARAMETERS nDividend, nDivisor
RETURN nDividend % nDivisor
```

and change the query to use it:

```
SELECT Product_ID ;
FROM Products ;
WHERE 10 = IIF( on_order > 0, Mod2( 10, on_order), 0) ;
INTO CURSOR Query
```

When I run this code, I get the error "Cannot divide by zero" in Mod2(). The same code runs without error in FPW2.6a.

Why doesn't the condition in the IIF() prevent the error?

-- Thomas M. Lamm (via Advisor.COM)

A: I'm assuming the example you've sent me is pared down from your real application code. There's no reason I can think of to replace a reliable built-in function like MOD() with your own home-grown version. In fact, there's a tremendous speed penalty for using your own version. (In a quick test, the built-in MOD() was about 28 times faster than MOD2().)

To sort out what's going on here, I ended up opening five different versions of FoxPro (VFP 6.0, VFP 5.0a, VFP 3.0b, FPW 2.6a and FPD 2.6a). As you noted, your code works without a problem in FPW 2.6a; it does so in FPD 2.6a, as well.

After that, it gets interesting. Testing against the Products table that comes with VFP 5.0a, both queries failed in VFP 3.0b, but only the second one failed in VFP 5.0a and VFP 6.0. A look at the Products table gave me a clue and with one change to the data, I was able to run both queries successfully in all three versions of VFP.

The change was to set the value of on_order in the first record of Products to something other than zero. When all versions ran successfully that way, it confirmed my suspicion that the problem is related to the initial processing of the query, not to its actual execution.

When you run a query in FoxPro, first the engine checks it out in various ways. Among them is a syntax check to make sure that all the required clauses are provided and that there are no data type mismatches or other syntactical errors. In order to do that, the expressions in the query need to be evaluated once to see what they return. In a one-table query, that check is performed using data from the first record. (I'm not sure what happens with multiple tables, though I'd guess the first record of each is used.)

In the case of a UDF, VFP runs it once – against the first record to make sure it's syntactically correct and to find out the type of value it returns. This is where you're getting into trouble. When the first record in Products has on_order = 0, the test run of Mod2() fails.

Actually, Visual FoxPro may execute each expression multiple times in different combinations, as it attempts to verify the syntax and to optimize your query. I've heard complaints from people using a UDF to do some kind of counting or totaling operation in a query. They find that their count or total is incorrect. This is caused by the same phenomenon – the test runs of the UDF.

In simple queries, you can probably test to determine the number of initial calls and adjust the results. For complex queries, your best bet is to avoid such UDFs altogether. Instead run a simple query that merely calls the UDF and stores the result in an intermediate cursor. Then use this cursor in the complex query.

It appears that, in VFP 3, even built-in functions were test evaluated as well. VFP 5 and VFP 6 seem to have eliminated that test, since the engine knows the return type of built-in functions.

The solution to your problem is to remember that a UDF called from a query cannot make *any* assumptions about the environment in which it runs. Just as you must pass any field values into the function to be sure that it looks at the right record, in this case, you need to test whether the values your function receives are valid. (Where you can, use IIF() rather than a UDF, since it avoids this problem altogether.)

Here's a revised version of MOD2() that prevents the error:

```
FUNCTION mod2
LPARAMETERS nDividend, nDivisor

LOCAL nReturn

IF nDivisor = 0
   nReturn = 0
ELSE
   nReturn = nDividend % nDivisor
ENDIF

RETURN nReturn
```

Writing defensive code like this is good practice anyway since a function you write for one routine may end up getting called somewhere else, too. (In fact, the routine should also check to make sure that both parameters it receives are numeric.) One of the key concepts of modular code is to make sure that each routine takes care of itself. It should make no assumptions about the environment when it starts and no permanent changes to that environment. Clearly, the second part of that statement isn't the case for routines whose purpose is to change the environment.

-- Tamar