February, 1998

Advisor Answers

Visual FoxPro 5.0 and 3.0 and FoxPro 2.6 for Windows

Q: I have found what I think is a big bug in VFP's SQL JOINs. In the sample program below, the JOIN condition should return 9 records, but it doesn't always do so. Some integer values seem to break the JOIN condition.

I know that in memory, VFP stores numeric, float, double and integer data as 32-bit integers if the decimals are 0. In all of these cases, the JOIN loses records. If I use the Currency type (the only numeric type stored differently in memory), the JOIN works.

If I force the JOIN to convert the numeric type to string (by testing STR(ID1) = STR(ID2)), all works.

I've found a number of specific values that fail. The constants NUM_1, NUM_2 and NUM_3 here represent one such set.

Here's my test code:

```
#define NUM_1  6415              && 6415
#define NUM_2  NUM_1 + 1         && 6416
#define NUM_3  NUM_1 + 2         && 6417

CREATE TABLE Parent FREE (ID N(4))
CREATE TABLE Child  FREE (ID N(4))

&& Insert parent with ID = 6415 and 3 children for it
INSERT INTO Parent (ID) VALUES (NUM_1)
INSERT INTO Child  (ID) VALUES (NUM_1)
INSERT INTO Child  (ID) VALUES (NUM_1)
INSERT INTO Child  (ID) VALUES (NUM_1)

&& Insert parent with ID = 6416 and 3 children for it
INSERT INTO Parent (ID) VALUES (NUM_2)
INSERT INTO Child  (ID) VALUES (NUM_2)
INSERT INTO Child  (ID) VALUES (NUM_2)
INSERT INTO Child  (ID) VALUES (NUM_2)

&& Insert parent with ID = 6417 and 3 children for it
INSERT INTO Parent (ID) VALUES (NUM_3)
INSERT INTO Child  (ID) VALUES (NUM_3)
INSERT INTO Child  (ID) VALUES (NUM_3)
INSERT INTO Child  (ID) VALUES (NUM_3)

** Now test...

TrySelect('Using N(4) as ID')

** Now try other numeric types ...
ALTER TABLE Parent ALTER COLUMN ID I(4)
ALTER TABLE Child  ALTER COLUMN ID I(4)
TrySelect('Using I(4) as ID')

ALTER TABLE Parent ALTER COLUMN ID B(8)
```

```
ALTER TABLE Child  ALTER COLUMN ID B(8)
TrySelect('Using B(8) as ID')

ALTER TABLE Parent ALTER COLUMN ID F(8)
ALTER TABLE Child  ALTER COLUMN ID F(8)
TrySelect('Using F(8) as ID')

ALTER TABLE Parent ALTER COLUMN ID Y(8)
ALTER TABLE Child  ALTER COLUMN ID Y(8)
TrySelect('Using Y(8) as ID')

* Clean up
USE IN TmpCur
USE IN Parent
USE IN Child

DELETE FILE Parent.dbf
DELETE FILE Child.dbf

**
FUNCTION TrySelect(cMsg)

   ?
   ? cMsg

   SELECT ALL                          ;
      Parent.ID AS ID_P,               ;
      Child.ID  AS ID_C                ;
   FROM                                ;
      Parent,                          ;
      Child                            ;
   WHERE                               ;
      Parent.ID = Child.ID             ;
   INTO CURSOR                         ;
      TmpCur

   ? 'Testing with "WHERE Parent.ID = Child.ID"          ', ;
     STR(_TALLY, 2)
   ?? IIF(_TALLY == RecCount('Child'), ' OK', ' !!!')

   SELECT ALL                          ;
      Parent.ID AS ID_P,               ;
      Child.ID  AS ID_C                ;
   FROM                                ;
      Parent,                          ;
      Child                            ;
   WHERE                               ;
      STR(Parent.ID) = STR(Child.ID)   ;
   INTO CURSOR                         ;
      TmpCur

   ? 'Testing with "WHERE STR(Parent.ID) = STR(Child.ID)"', ;
     STR(_TALLY,2)
   ?? IIF(_TALLY == RecCount('Child'), ' OK', ' !!!')

ENDFUNC
```

I've tried this using VFP 5's JOIN clause as well, with the same results. Can you explain what's going on here?

–Mario Cenzato (via Internet)

A: The first time I tried your sample code, every query returned 9 records, as I expected. It took me a few minutes to figure out why you were seeing different results. Then I remembered an on-line "conversation" I recently overheard and tested again, first issuing SET COLLATE to "GENERAL". Sure enough, in that case I was able to duplicate your results.

SET COLLATE determines the ordering used for sorting and comparisons. Since the ASCII character set is ordered on English-language characters, sorting in English is no problem. "A" (=CHR(65)) comes before "B" (CHR(66)) which comes before "C" (CHR(67)) and so forth. When you sort English words, it's easy for the computer to simply check the internal representation of each character and sort based on that representation.

However, many other languages include characters with diacritical marks (like accents) and other special characters (like the Spanish "CH", which sorts between "C" and "D"). While various character sets include these characters (search for "code page" in the FoxPro help for several useful topics), they're not placed in the appropriate positions in the set to make it possible to sort based on the internal representation of each character.

FoxPro 2.6 introduced the notion of a "collation sequence", a predefined ordering of the character set to make it possible to sort appropriately for one or more languages. The collation sequence based on the order of the character set is known as "Machine" and it uses the numeric value of each character. There are a number of other collation sequences available in FoxPro 2.6 and Visual FoxPro, such as "German", "Spanish", and "SweFin" (Swedish and Finnish). The special "General" collation sequence is especially appealing because it is case-insensitive. That is, "a" and "A" sort to the same position, which lets you create your indexes without using the UPPER() function.

But why should the collation sequence affect this query? After all, it's comparing fields of the same size and type containing identical data. Why wouldn't they match? After playing with the data for quite a while, I wondered whether Rushmore was playing any role in this behavior. Since the collation sequence (used primarily for indexing) was involved and FoxPro's optimization is based on indexes, this seemed like a good place to look.

First, I added an index tag on ID to both tables. Sure enough, the query gave correct results for all numeric types. Then, I tried creating an index tag for only the parent table. With COLLATE set to "General", this gave bad results. However, a tag only on the child table gave correct results. This gave me the hint I needed.

I turned on SQL ShowPlan with SYS(3054,11) to see exactly how VFP was optimizing the query and tried the various scenarios again. In the cases that were giving correct results, VFP was using the index I'd built for it. In the other cases, VFP was building its own temporary index.

Obviously, something odd is happening (read: there's a bug) when VFP builds a temporary index. Unfortunately, I don't know of any way to look inside that temp index

to see exactly what's going on. I was able to confirm the bug in both VFP 5.0a and 3.0b, but the problem does not occur in FP2.6a.

The solution for you, however, is clear. Either avoid collation sequences other than "Machine" whenever possible (which is the advice I've heard from a number of European developers) or make sure that you build all the right indexes up front.

–Tamar