January, 2000

Advisor Answers

Maximum Table Size

VFP 6.0, 5.0, 3.0 and FoxPro 2.x

Q: How large can a single table be in VFP 6.0? I'm having a problem creating one larger than 2GB.

–Lance J. Dorsey (via Advisor.COM)

A: In fact, you've found the limit. The documentation clearly states (see the Visual FoxPro System Capacities category in Help) that the maximum size for a table is two gigabytes. The same entry says that a table can have no more than a billion records. That makes sense since every record has one byte reserved for the deleted flag. In fact, there's additional disk space required for the table header and field-level information that makes a billion records seem more theoretical than realistic, even if you can imagine wanting records that have one byte apiece and even finessing the difference between a billion and a gigabyte.

The .DBF table header includes the number of records in the table. Four bytes (bytes 4 through 7, which in the 0-based counting system are actually the fifth through eighth bytes) are used for this purpose. In fact, with four bytes, the count could go to something over four billion, but three bytes would have limited the table to about 16 million records. While that's still a lot of records, it's nice to think that the .DBF's designers recognized that it just wasn't going to be enough somewhere down the road. The formula for turning this information into the current record count is:

```
Byte4 + (Byte5 * 256) + (Byte6 * 256^2) + (Byte7 * 256^3)
```

You can check this for yourself using the low-level file functions:

```
LOCAL nFileHand, cHeaderInfo
LOCAL nByte4, nByte5, nByte6, nByte7, nRecCount

nFileHand = FOPEN(GETFILE()) && Choose a .DBF file
cHeaderInfo = FREAD( nFileHand, 8)
nByte4 = ASC(SUBSTR(cHeaderInfo,5))
nByte5 = ASC(SUBSTR(cHeaderInfo,6))
nByte6 = ASC(SUBSTR(cHeaderInfo,7))
nByte7 = ASC(SUBSTR(cHeaderInfo,8))
nRecCount = nByte7 * 256^3 + nByte6 * 256^2 + ;
            nByte5 * 256 + nByte4
```

Of course, you don't have to do all this yourself, because RECCOUNT() returns this value for you.

So why doesn't Microsoft increase the limit, now that we have operating systems that support partition sizes larger than 2 GB? Good question. An obvious answer is backward compatibility. A change of this magnitude would affect a lot of existing applications, but Microsoft has changed the .DBF format before. There may be other, technical reasons

involved. To support larger tables now might involve using two distinct locking schemes (and perhaps two separate indexing schemes as well) – it's possible that the two types of .DBFs might be incompatible. From a more pragmatic point of view, supporting .DBFs larger than 2GB doesn't fit with Microsoft's view of the world. They see SQL Server as the appropriate repository for a database of that size. VFP is for smaller data sets and for building middle-tier components that address larger data sets.

To solve your immediate problem, which is what to do about your large table, there are several possible approaches. First, make sure that your data is properly normalized. That is, be sure that you don't have repeated data in your table, that every item in the table is needed at that level rather than belonging in a child table. (If you're not sure how to do this, take a look at Miriam Liskin's June '95 column.)

Once you're sure the data is normalized, the solutions all come down to partitioning your data, either horizontally or vertically. That is, you can divide your fields up into several different tables, keeping a record for each primary key in each table, or you can put some of the records into each table based on date or location or some other characteristic that keeps together the records most likely to be needed together. For more information on partitioning, see Val Matison's article in the March '97 FoxPro Advisor. Val was involved in the EuroTunnel project and came up with some interesting strategies for managing large data sets.

–Tamar