FoxTalk
PINNACLE
Solutions for Microsoft® FoxPro® and Visual FoxPro® Developers

This is an exclusive supplement for *FoxTalk* subscribers. For more information about *FoxTalk*, call us at **1-800-788-1900** or visit our Web site at **www.pinpub.com/foxtalk.**

Extended Article

So the Data is too Large...

Jim Booth

WIN 5.0 6.0 DOWNLOAD

Perhaps you've run into the 2G file size limitation in Visual FoxPro, or you're having a problem because your table needs more than the allowed number of fields. Managing huge data sets has its own unique set of problems. Even with VFP's Rushmore technology, accessing huge data sets can take a long time. Jim Booth offers some suggestions of what can be done to help in managing huge data sets.

HUGE is a relative term. There's no fixed definition for the word huge, other than very large. Within the area of database design, huge refers to extremely large data sets. The term huge has to be applied in the context of the database management system you're using. A huge data set for Microsoft Access isn't the same as a huge data set for Visual FoxPro, which in turn isn't the same as a huge data set for Microsoft SQL Server (or Oracle or any other client/server database system). In the context of this column we'll be considering Visual FoxPro databases specifically; however, the same principles apply for all other database managers as well.

In Visual FoxPro we might consider the word huge to apply when the record count reaches a few hundred million records. Tables smaller than that are generally managed quite well by VFP without special handling. Another criterion for a huge data set is the actual size of the data file on disk. VFP has a file size limitation that no file may exceed 2G in size. If a table has a relatively large record size, it might approach the file size limit before it approaches the record count mentioned above. Be warned—using these techniques for data sets that aren't suffering from the problems they address can complicate your data design with little or no benefit.

What's the problem?

There are two major problems associated with huge data sets. The first is performance. It usually takes longer to retrieve information from huge data tables. The second is limitations. Some database managers have a fixed limit on the size of the files they can manage. Visual FoxPro's file size limitation is 2G. A record in a Visual FoxPro table is limited to no more than 255 fields. A specific design requirement could exceed either of these capacities.

You might think that you'll never reach the limitations of VFP, but you can still encounter the performance problems at file sizes considerably less than the limitation. You might also encounter situations where the data that would normally be stored in a single record is relatively large—that is, many fields of data.

Okay, so where are you going with this?

Let's discuss a design approach that can address the problems associated with huge data sets. This solution is called data partitioning. Data partitioning is dividing the data between multiple tables. There are two types of partitioning that are available: horizontal and vertical. We'll discuss each of these separately. Each approach has situations where it does well, and other instances where the alternative would be better.

Horizontal partitioning

Regardless of whether your problem is files or records that are too large, horizontal partitioning offers the best improvement when the data in the record is easily divided in a logical manner into groups of fields that are often used together. **Figure 1** shows an example table design that we'll use in the discussion that follows.

For the sake of readability, the table design in Figure 1 isn't actually a huge record at all. It would be unfair to make my column longer than all the others in this issue because the figures have over 255 fields listed in them (never mind that I can't think of 255 fields that I'd want in one table). The principal points of partitioning can be described and understood even though the table design doesn't require the process at all.

The idea of partitioning a table is to divide the single table into two or more tables. The horizontal concept is

that the division is accomplished by dividing a record into two or more tables by putting some of the fields in each table. If you picture how data looks in a grid, then horizontal and vertical take on a clearer meaning. With horizontal we divide the fields into multiple tables. **Figure 2** is an example of the table in Figure 1 having been partitioned horizontally.

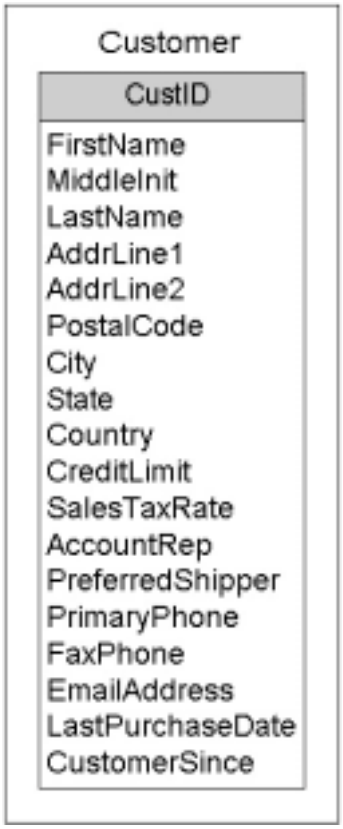
As the figure demonstrates, horizontal partitioning is accomplished by dividing the fields between, in this case, two tables that each contain a subset of the fields. Notice that the CustID primary key of the Customer1 table exists in the Customer2 table as a foreign key to allow the relationship between these two tables to exist. The two new tables will have a one-to-one relationship.

What's the impact on the code?

The biggest impact on your code has to do with how you can see all of a customer's fields at once. If the horizontal partitioning was done because the number of fields exceeded the maximum allowed for a table, then you're limited in how you can combine the fields to get them all together. The reason is that no matter what you do, you can't get all of the fields into one table or cursor. In this situation you need to relate the two tables to each other and then show fields from each table. This can be done relatively easily using the SET RELATION command in VFP.

Here's an example of using the SET RELATION to retrieve data from these two tables:

Figure 1. A table with large records.




```
SELECT 0

* Open the Customer1 table with the alias of Customer.
* Set the order for the sequencing of the records in
* alphabetical order.
USE Customer1 ORDER LastName ALIAS Customer

* Open the Customer2 table with the alias of
* CustomerExtension in a free work area.
* Set the order for the relationship.
USE Customer2 ORDER CustID ALIAS CustomerExtension IN 0

* Establish the relationship between the tables.
SET RELATION TO CustID INTO CustomerExtension

* Move the record pointer to the first record.
LOCATE

* Show some data from each table together.
WAIT WINDOW "Customer: " + ;
   Customer.LastName + ;
   " Credit Limit: " + ;
   ALLTRIM(STR(CustomerExtension.CreditLimit,10,0))
```

If the horizontal partitioning was done because the total size of the table was approaching the file size limit but the record size was well within the limits, then SQL SELECT can be used to retrieve data for display and manipulation. The following code example would get you all of the fields from both tables for those customers in the state of New York:

```
SELECT Cust1.*, Cust2.* ;
  FROM Customer1 Cust1 JOIN Customer2 Cust2 ;
    ON Cust1.CustID = Cust2.CustID ;
 WHERE Cust1.State = "NY" ;
 ORDER BY LastName ;
 INTO CURSOR Customers
```

This will provide you with a cursor named Customers that has all of the fields from the two tables in a single cursor.

If I can combine the fields together into a cursor, why did I need the partitioning in the first place? Well, it's possible that, if you combine the fields from the two tables into one table for all of the records, the table's size would exceed the database manager's limitation

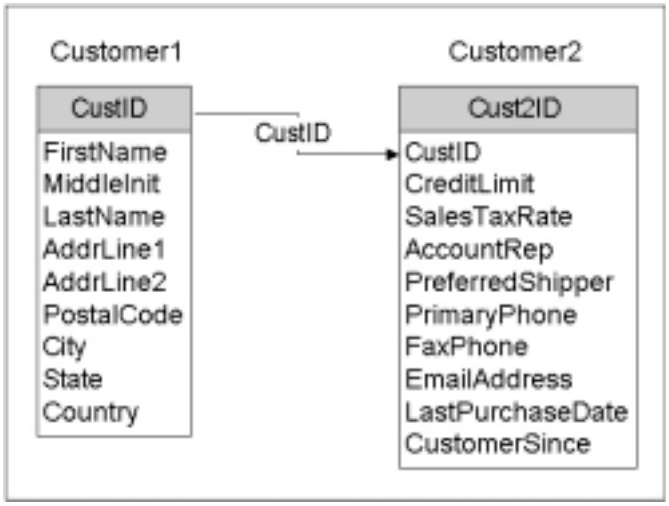


Figure 2. The example table after horizontal partitioning.

on file size.

Also, sometimes you find that your queries are taking a very long time to finish. In cases where the data fields can be logically divided into groups that work together, it's possible to create horizontal partitioning where the combination of fields from the partitioned tables seldom needs to occur or at least is limited.

For example, the partitioning in the customer example in Figure 2 has the demographic fields in Customer1 and the financial fields in Customer2. It's possible that, when working with the financial data, we'd only need a small number of fields from the other table.

Vertical partitioning

As opposed to horizontal partitioning, where we divided the fields between multiple tables, with vertical partitioning we divide the records between two or more tables. Vertical partitioning works well in situations where the record size (number of fields per record) isn't approaching the limits of the database manager, but the record size combined with the number of records is approaching the file size limit.

When we approach the file size limit, we must make multiple smaller files in order to continue to record information. **Figure 3** is an example of using vertical partitioning on the customer data from Figure 1.

It has partitioned the data based on the spelling of the customer's LastName. Although this might be a questionable way to vertically partition the data because

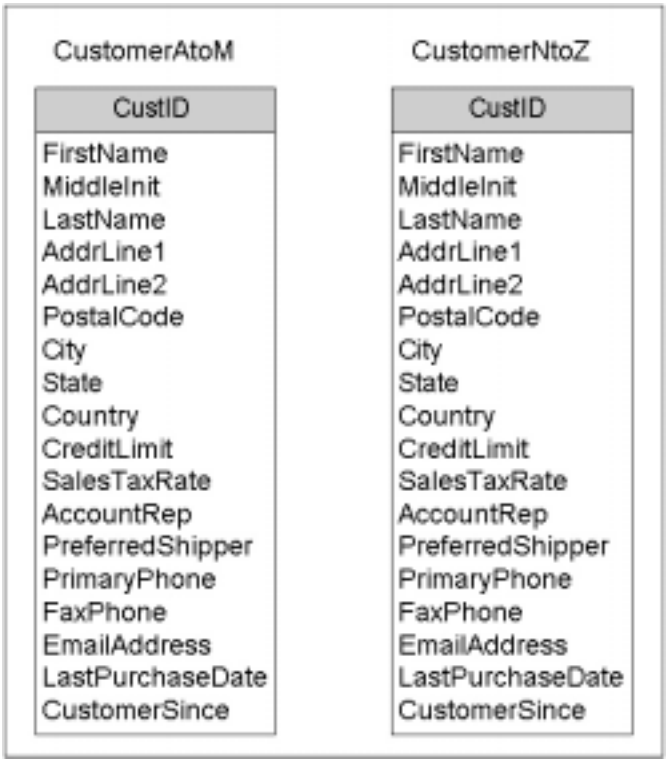


Figure 3. Vertical partitioning of the table in Figure 1.

of the retrieval requirements, it does demonstrate how to accomplish vertical partitioning. Some partitioning boundaries that might make more sense would be: on State, on AccountRep, on City, and so on. Essentially, using some logical dividing factor that relates to the retrieval of data from the Customers would make more sense as the partition boundary. The reason that the LastName was used here is simply that drawing the diagram of vertical partitioning on State would require that there be 50 tables, one for each state, and wouldn't take into account the possibility of records that are outside the 50 states.

You might also choose simply to divide records into two or more tables based on a completely random factor, and continue to split records off into new tables when the existing tables become "full." This is perhaps the safest way to vertically partition, but it can potentially add another layer of logic, as all operations have to consider all tables as possibly containing members of a desired record subset.

How do you work with vertically partitioned data?

Vertically partitioned data is more difficult to work with than horizontally partitioned data simply because there are usually more tables involved. But even with the same number of tables, combining data from the tables is not a simple matter of using a JOIN or relationship. You have to use a UNION in SQL, and there's no direct way of combining the data from the two tables using simple xBase syntax.

The following code sample would combine the records from the two customer tables and retrieve all of the customers in New York:

```
SELECT CustomerAtoM.* ;
  FROM CustomerAtoM ;
 WHERE State = "NY" ;
UNION
 SELECT CustomerNtoZ.* ;
   FROM CustomerNtoZ ;
  WHERE State = "NY" ;
  ORDER BY LastName ;
INTO CURSOR NYCustomers
```

Putting it together

Data partitioning can overcome some very real problems in data design. Horizontal partitioning is easier to manage than using vertical partitioning, but no matter which partitioning you use, the overhead in your application code will be greatly reduced if you do the partitioning using boundaries that won't need to be crossed often. This requires the use of only one of the partition tables at a time.

When you face a situation where the size of the data will create a need for partitioning, you should first *consider using a different database manager* that doesn't have the limitation you're encountering. Only if using a different database manager isn't possible should you

consider using partitioning.

A real-world example of one place where I used vertical partitioning was in a software application I developed for a local court system. The probate courts in my home state handle a number of different types of cases, such as decedent estates, guardianships, and conservators. Every case in all of these types has a person known as the fiduciary who is responsible for the case. We initially used a single fiduciary table to store information about these people, but the performance was terrible (this was before Rushmore was available). The real problem was that the performance was poor for all case types, even though the decedent estate cases outnumbered the other types by an order of 100-200 to 1.

I hit upon the idea that it would greatly improve the performance of the other two case types if the decedent estate fiduciaries were in a separate table from the other case type fiduciaries. So we created DFid, Gfid, and CFid tables. They all have an identical structure, but they record fiduciaries for only one type of case. The improvement was so noticeable that we did the same thing for all of the case files in the system: fiduciaries, interested parties, documents submitted, documents produced, and so forth.

We even did some Horizontal/Vertical partitioning to the main case data by using a single Master Case table horizontally partitioned into multiple Secondary Case tables. **Figure 4** shows the idea of this dual partitioning.

In the diagram, there's horizontal partitioning in the case information being divided into fields in the Master Case table and additional fields in one of the secondary case tables. The secondary case tables are vertically partitioned along the boundary of the case type.

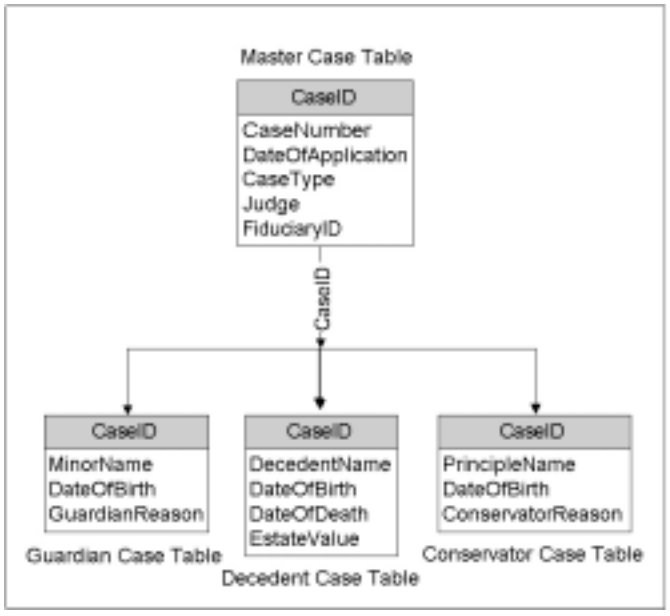


Figure 4. A real-world example of a combination of both horizontal and vertical partitioning used together.

Summary

Data partitioning is a technique that can allow you to manage data sets that might, at first, seem outside the capabilities of your selected tool. Partitioning can also, though rarely, be helpful in organizing the data in a system even when system limitations aren't being stressed.

It's important to consider the ramifications of partitioning on the data retrieval code in your application before deciding to use it. Finally, and most importantly, *always consider using a different database manager* before you resort to data partitioning to solve problems involving limitations of a data management tool. ▲

DOWNLOAD **02BOOTSC.ZIP at www.pinpub.com/foxtalk**

Jim Booth is a Visual FoxPro developer and trainer. He has spoken at FoxPro conferences in North America and Europe. Jim has been a recipient of the Microsoft Most Valuable Professional Award every year since it was first presented in 1993. He is co-author of *Effective Techniques for Application Development* and *Visual FoxPro 3 Unleashed* and is contributing author for *Special Edition Using Visual FoxPro 6.0*. Jim is also the technical editor for *Database Design for Mere Mortals*. 203-758-6942, jbooth@jamesbooth.com, www.jamesbooth.com.