What I Learned This Month: When VARCHAR Makes Sense

Scott Chapman
American Electric Power

Mainframe professionals tend to focus a great deal on efficiency. Overall, this is a very good thing, but sometimes we keep efficiency rules around longer than we need to in the face of advancing technology.

A case in point is defining character columns as either CHAR (fixed-length) or VARCHAR (variable length) in DB2. Most mainframe DBAs will promote CHAR over VARCHAR for most relatively small fields. This is because it takes an extra two bytes to store the length of the value as well as the value itself. For an 8-character field (a common length), this would typically work out to an additional 25% storage if all the values were 8 bytes long. There'd be no storage savings unless the average length of the field value was less than 6 bytes.

But perhaps more important than the storage is the little extra work that has to be done to store and manipulate the length of each VARCHAR. It isn't much, but depending on how many times the field in question is being accessed, it can (at least in theory) add up. The processing overhead is magnified when a VARCHAR is updated and its length changes: the position of where fields are located in the physical record shifts around. The recommendation in the past was to put all VARCHAR fields at the end of the table definition.

In addition, DB2 on z/OS has enjoyed access to hardware instructions to do compression. The hardware compression engines don't eliminate the compression overhead, but it does make it much more palatable. Compression can save a lot more storage than just eliminating trailing blanks.

So those of us who have long experience with mainframe DB2 just consider it a natural "law" that VARCHAR should be generally avoided. But when applications that were designed on other platforms come up to the mainframe, the use of VARCHAR is always a source of discussion. Sometimes we come to agree to change the shortest to CHAR, but often we just leave them as VARCHAR.

I've always wondered why VARCHAR seems to be so much more prevalent in database designs that originate off the mainframe. I have finally realized why. While you can write mainframe applications in many languages other than COBOL (including C/C++, Java, and any of the many languages that run in a Java JVM), the predominant language in many mainframe shops is still COBOL. In C and similar languages, strings tend to be variable length. COBOL deals much more naturally with fixed-length strings and the application programmer has to go to some bit of extra work to deal with variable length strings.

So for the COBOL programmer, it is much more natural to deal with storing and retrieving a value of "Michael " or "Bob " (note the trailing blanks), whereas a Java programmer would likely expect to store and retrieve "Michael" or "Bob" without any trailing blanks. VARCHAR is simply a more natural way of working to the Java programmer.

I came to this realization recently while debugging a Java program and discovering that the issue traced back to the fact that “Bob” and “Bob “ are not the same. That’s when the light went on for me and I understood why we see so many VARCHAR definitions in database designs. I’m surprised and more than a little chagrined that I didn’t come to this understanding years ago.

Once I realized this, my first thought was that we had changed the table definition to be CHAR instead of VARCHAR, but when I checked, I discovered that we had left the columns all as VARCHAR. Further investigation revealed that at one point in the program’s flow it was getting back the value padded with spaces and then later when it read the data from DB2 it was coming back without any padding. This was the same row of data. Further investigation revealed that the unexpected (and unwanted) padding appeared only for a few of the columns, and only in queries that contained just those problem columns or certain other columns. I was very confused.

Fortunately my friendly z/OS DB2 DBA came to the rescue again. The queries that were incorrectly padding the columns with blanks were retrieving the values from the index. When you create an index with VARCHAR columns, there is an option to either pad the values with spaces or not. There is also a DB2 subsystem option that specifies the default. Our default is to pad the values in the index because that was the way DB2 worked in v7. Redefining the index with the “NO PADDING” option fixed the problem.

This option apparently came about in DB2 v8. I couldn’t find much about the performance implications of the option. I found one reference that suggested that removing the padding (making the columns variable length) would add some amount of CPU overhead. But I couldn’t find any numbers for how much overhead might be involved. I didn’t try to calculate the impact for the application in question because it simply needs the “NO PADDING” option to function properly.

Other references suggested that there shouldn’t be a functional difference and that “NO PADDING” could improve performance because DB2 wouldn’t need to go to the data pages to determine the actual length of the values. This implies to me that there shouldn’t be a difference visible to the application, but that was not the case in my testing. We verified that with the “PADDING” option, DB2 returned the extra spaces when reading the columns from just the index in both v9 and v10. So while that feels like incorrect behavior to me, it’s at least been the way things have worked for some time in our shop.

So this month’s lesson is that VARCHAR does make sense from a Java programmer’s perspective, even for short columns. And if you have VARCHAR columns in indexes, you should consider defining the index with “NO PADDING”.

As always, if you have questions or comments, you can reach me via email at ~~sachapman@aep.com~~ scott.chapman@epstrategies.com