Paper 2690-2018

Using SAS® 9.4M5 and the Varchar Data Type to Manage Text Strings Exceeding 32kb

John Schmitz, Luminare Data LLC

ABSTRACT

Database systems support text fields much longer than the 32kb limit traditionally supported by SAS®. These fields can be captured as substrings using SAS and explicit pass-through logic. However, managing the resulting substrings is often challenging. This paper will show how the varchar data type, introduced in 9.4M5, can provide improved functionality. The paper solves common management and storage issues associated with the extraction of longer strings, using varchar field type and the V9 engine.

INTRODUCTION

Strings longer than 32kb (32767) characters require additional steps to fully extract into a SAS readable format. Although these strings are commonly supported in most database management systems (DBMS), they are not fully supported in SAS DATA steps or in the SAS/ACCESS® engines that facilitate the transfer of data between the DBMS and SAS. Beginning with version 9.4M5, SAS has introduced features that will aid programmers when managing these longer string variables. However, these new capabilities are not a complete solution to the challenges such data offer. This paper provides suggestions to simplify text processing in SAS when longer strings are encountered. The paper will first review the VARCHAR data type that is new in 9.4 M5. The paper will briefly touch on the traditional approach to handling long strings for systems prior to 9.4M5, and then present some new methods that can be used with the VARCHAR feature in M5.

This paper assumes the reader is familiar with general SAS coding including the SQL procedure and SQL pass-through logic. The paper provides only a brief outline of the process for extracting these longer strings into SAS, so that more attention may be focused on the management of these longer strings once they have been transferred to SAS. The code samples shown use the SAS/ACCESS for SQL Server engine, but the same process would apply for other engines including ODBC and OLEDB.

THE VARCHAR DATA TYPE

SAS has introduced a new VARCHAR data type with SAS 9.4 M5. This new data type is included to support SAS VIYA® and the new HDAT data type used within the CAS Engine. However, the core capabilities of VARCHAR are included in the DATA step as part of the BASE SAS license, whether you have VIYA and CAS in your SAS environment or not.

A VARCHAR data type is declared using the LENGTH statement, using the keyword VARCHAR:

```
LENGTH longstr VARCHAR(n);
```

where LONGSTR is the user-defined variable name and n is a numeric value that defines the maximum length that the variable string LONGSTR may assume. The maximum length of a VARCHAR is 2^{29}-1 or 536,870,911 compared to the maximum length of 2^{15}-1 or 32,767 for a fixed length string. Unlike SAS standard character variables, a VARACHAR cannot be declared with a FORMAT statement.

The SAS V9 engine does not support VARCHAR fields as a storage type. Therefore, the VARCHAR field will be available within your DATA step but will not be outputted as a VARCHAR when a V9 dataset is used as the output table. Rather, for the V9 engine, the fields will be converted to a fixed length field of no more than 32767 and stored as a CHAR variable. Any characters in the string beyond the field length will be truncated.

As an example, consider a simple case:

```
%put SAS Version:  &SYSVLONG;
options msglevel=i;
```

```
data sample1;
    length longstr varchar(160000);
    longstr = 'ABC';
run;

title 'Contents of SAMPLE1 Dataset';
proc contents data=sample1;
run;

proc print data=sample1;
run;
```

In this case, LONGSTR is a VARCHAR with a maximum length of 160,000 characters. The log for this example shows a few items worth noting, namely a report of the SAS version used:

```
%put SAS Version:  &SYSVLONG;
SAS Version:  9.04.01M5P091317
```

and a message regarding the VARCHAR datatype and the V9 engine:

```
NOTE: VARCHAR data type is not supported by the V9 engine.
Variable longstr has been converted to CHAR data type.
```

The CONTENTS procedure output confirms the field conversion indicated in the log::

```
Alphabetic List of Variables and Attributes

# Variable             Type      Len

1 longstr              Char      32767
```

The PRINT procedure shows the data remain intact in output, since the string was short enough to fit within the field length.

```
Contents of SAMPLE1 Dataset

      Obs    longstr
        1    ABC
```

PROCESSING LONGER STRINGS PRIOR TO 9.4 M5

The common approach to extracting longer strings to SAS would involve:

- Leveraging SQL pass-through to substring fields into string lengths no more than 32kb.
- Transferring the substring segments to SAS as separate strings.
- Processing data within the DATA step using each of the string subsets, often within a loop.
- Some business requirements may require pre-processing on the substrings to create more natural break points. This is done so that the desired text is not split across 2 substrings.

Depending on the users desired outcome and the number of substrings involved, this string processing across the substring segments can become a complex process. Readers interested in a more thorough presentation on the extraction and processing of such strings should review the paper by Schmitz (2017) included in the References section.

Assume we have a SQL data table with XML strings included as a VARCHAR field. For this example, we will assume a maximum string length of 160,000 characters so 5 substrings of 32,767 characters each will be adequate to extract the entire field. SQL pass-through code can be used to divide the entry into 5 substrings and return the string as multiple substrings:

```
proc sql;
    connect to sqlsvr as src
          (dbmax_text=32767 dsn=lumin_dev authdomain=SQLAuth);

    create view XML_SampleData_vw as
    select *
    from connection to src
          (select
                tableid,
                substring(XMLResponse,      1, 32767) as XMLResponse1,
                substring(XMLResponse, 32768, 65534) as XMLResponse2,
                substring(XMLResponse, 65535, 98301) as XMLResponse3,
                substring(XMLResponse, 98302,131068) as XMLResponse4,
                substring(XMLResponse,131069,163835) as XMLResponse5
          from dbo.XML_SampleData);

    disconnect from src;
quit;
```

This will generate a DATA step view with a TableID (table key) and 5 substrings with subsets of the XMLResponse field. Note that DBMAX_TEXT option is used in the connect statement. This setting will override the default maximum length for character data used by the ODBC drivers. This setting commonly defaults to 1000 characters.

Continuing with this example, assume we are looking for an ID value that appears in the text immediate after a specific UUID value. The DATA view can serve as input to a SAS DATA step for the additional string processing requirments. In this example, the code searches for a particular UUID value within the substring text then captures the next 12 characters following the identified string. In a pre-VARCHAR process, the basic data steps would appear as:

```
data XML_Sample;

    ** LOAD VIEW AND DEFINE ARRAY OVER SUBSTRINGS **;
    set XML_SampleData_vw;
    array xmls {5} XMLResponse1-XMLResponse5;

    ** CREATE LOOP TO SEARCH ACROSS SUBSTRINGS WITH FIND **;
    pos=0;
    do i = 1 to 5 until (pos > 0);
          pos = find (XMLS(i),"DFE24CAA-04EF-4F81-86D8-9F3BABBA616B");
    end;

    ** IF FOUND, CAPTURE NEXT 12 CHARACTERS FROM STRING **;
    length IDString $12;
    if pos > 0 then
          IDString = substr (XMLS(i),pos+
                length("DFE24CAA-04EF-4F81-86D8-9F3BABBA616B"),12);
run;
```

This code loads the view created by the SQL pass-through above and searches across each substring for the position of the desired UUID code. Here, a SET statement is used and an ARRAY is declared to facilitate a loop across all the strings. A subsequent DO UNTIL loop is used to search each string for the desired code. Once found, the position is recorded relative to that particular substring and processing is passed to a subsequent SUBSTR command to capture the next 12 characters s IDString.

Of course, this approach has 1 critical limitation in that it cannot find the desired string if it is split across two substrings. It would require additional coding to concatenate strings across the break points to ensure no values are missed due to this limitation. This approach requires creating natural breaks in the string subsets or pre/post appending data from an adjoining string. Either approach would typically require that the substring extracted from the DBMS be less than the maximum 32767 characters to allow room for this additional string manipulation process. Since the focus of this paper is upon improving this process with VARCHAR, there is minimal benefit in developing that logic here. Again, readers who are interested in more details on this approach are directed to Schmitz (2017).

MODIFYING THE APPROACH FOR M5 AND VARCHAR

Because the M5 code additions do not alter the SAS/ACCESS engines to allow longer strings, it is still necessary to execute a SQL pass-through approach to subset and extract the data. However, the VARCHAR field type does allow the user to reconstruct the full string inside the DATA step, up to the 536,870,911 character limit. In most cases, this approach will allow the user to bypass many of the substring limitations raised above. In M5, an improved DATA step using the VARCHAR type can be constructed as:

```
data XML_Sample2;

    ** LOAD VIEW AND DEFINE ARRAY OVER SUBSTRINGS **;
    set XML_SampleData_vw;
    array xmls {5} XMLResponse1-XMLResponse5;

    ** CREATE AND POPULATE VARCHAR FIELD **;
    length longstr varchar(160000);
    do i = 1 to 5;
          longstr = cat(longstr,xmls(i));
    end;

    ** FIND AND SUBSTR WILL NOW WORK OVER ENTIRE VARCHAR FIELD **;
    pos = find (longstr,"DFE24CAA-04EF-4F81-86D8-9F3BABBA616B");

    ** IF FOUND, CAPTURE NEXT 12 CHARACTERS FROM STRING **;
    length IDString $12;
    if pos > 0 then
          IDString = substr (longstr,pos+
                length("DFE24CAA-04EF-4F81-86D8-9F3BABBA616B"),12);
run;
```

The SET and ARRAY statements are retained as before. A LENGTH statement is used to declare a VARCHAR variable and a DO loop is used to concatenate each substring to the declared VARCHAR field. Once populated, the FIND and SUBSTR functions can search the entire VARCHAR field to obtain the desired outcome.

On the surface, this new approach may not appear any simpler. However, the previous approach bypasses any logic to manage cases where breaks within the substrings will result in inaccurate and/or incomplete results. In the latter case, there are no substring sections, so the logic can find the patterns even when they appear across substrings. Furthermore, there is no need to loop over substrings each time the user processes against the string data since the single VARCHAR holds the entire field.

KEY LIMITATIONS TO THE VARCHAR PROCESS

Although the VARCHAR type provides a much easier process for searching and processing on these longer strings, several limitations with the VARCHAR data type should be addressed.

One of the key limitations is saving the file since the VARCHAR is not supported in V9 engine. One option is to drop the field but retain the substrings captured from the SQL pass-through. The VARCHAR can be readily reconstructed from the substrings as desired.

```
data XML_Save;
    ** LOAD VIEW AND DEFINE ARRAY OVER SUBSTRINGS **;
    set XML_SampleData_vw;
    array xmls {5} XMLResponse1-XMLResponse5;

    ** CREATE AND POPULATE VARCHAR FIELD **;
    length longstr varchar(160000);
    do i = 1 to 5;
          longstr = cat(longstr,xmls(i));
    end;

    ** DROP VARCHAR BEFORE SAVE **;
    drop longstr;
run;

data XML_Load;
    ** LOAD VIEW AND DEFINE ARRAY OVER SUBSTRINGS **;
    set XML_Save;
    array xmls {5} XMLResponse1-XMLResponse5;

    ** RECREATE AND POPULATE VARCHAR FIELD **;
    length longstr varchar(160000);
    do i = 1 to 5;
          longstr = cat(longstr,xmls(i));
    end;
run;
```

An alternative is to write the VARCHAR as a text file. As an example:

```
filename outf1 "~/Documents/outf1.txt";
data XML_Sample3;
    file outf1 lrecl=160000;

    ** LOAD VIEW AND DEFINE ARRAY OVER SUBSTRINGS **;
    set XML_SampleData_vw;
    array xmls {5} XMLResponse1-XMLResponse5;

    ** CREATE AND POPULATE VARCHAR FIELD **;
    length longstr varchar(160000);
    do i = 1 to 5;
          longstr = cat(longstr,xmls(i));
    end;

    put longstr;
run;
```

From the log:

```
NOTE: 2 records were written to the file OUTF1.
The minimum record length was 127490.
The maximum record length was 139266.
```

The sample code provided uses a loop concatenate multiple strings together. This is important since using a single CAT function to join all fields together will fail. Consider the code:

```
    longstr = cat(xmls(1),xmls(2),xmls(3),xmls(4),xmls(5));
```

This code produces an error due to buffer limits in the string function:

```
WARNING: In a call to the CAT function, the buffer allocated for the result
was not long enough to contain the concatenation of all the arguments. The
correct result would contain 163835 characters, but the actual result might
either be truncated to 32767 character(s) or be completely blank, depending
on the calling environment. The following note indicates the left-most
argument that caused truncation.
```

Buffer errors are not uncommon using VARCHAR when string functions must buffer too many characters.

The SUBSTR function works as expected when extracting from a VARCHAR to a CHAR for results up to the maximum CHAR field length of 32767 characters. Consider the code:

```
data XML_Sample5b;

    ** LOAD VIEW AND DEFINE ARRAY OVER SUBSTRINGS **;
    set XML_SampleData_vw;
    array xmls {5} XMLResponse1-XMLResponse5;

    ** CREATE AND POPULATE VARCHAR FIELD **;
    length longstr varchar(160000);
    do i = 1 to 5;
          longstr = cat(longstr,xmls(i));
    end;

    ** SUBSTR WILL WORK OVER ENTIRE VARCHAR FIELD **;
    length FindString varchar(80000);
    FindString = substr (longstr,500,80000);
    FindLen = length(findString);
    output;


    FindString = substr (longstr,32768,80000);
    FindLen = length(findString);
    output;

    FindString = substr (longstr,125000,80000);
    FindLen = length(findString);
    output;

run;
```

```
title Summary Output from XML_Sample5b;
proc print data=XML_Sample5b;
   var FindString FindLen;
   format FindString $10.;
run;
```

The PRINT procedure shows that strings were returned as expected:

```
                    Summary Output from XML_Sample5

                       Obs   FindString   FindLen
                       1     //data.lac   32767
                       2     9</total_p   32767
                       3     /zip_code>   14267
                       4     g steps to   32767
                       5     mnId="2824   32767
                       6     responds     2491
```

A similar example can be constructed using a VARCHAR rather than the CHAR field to capture the result of the substring.

```
data XML_Sample5b;

   ** LOAD VIEW AND DEFINE ARRAY OVER SUBSTRINGS **;
   set XML_SampleData_vw;
   array xmls {5} XMLResponse1-XMLResponse5;

   ** CREATE AND POPULATE VARCHAR FIELD **;
   length longstr varchar(160000);
   do i = 1 to 5;
         longstr = cat(longstr,xmls(i));
   end;

   ** SUBSTR WILL WORK OVER ENTIRE VARCHAR FIELD **;
   length FindString varchar(80000);
   FindString = substr (longstr,500,80000);
   FindLen = length(findString);
   output;

   FindString = substr (longstr,32768,80000);
   FindLen = length(findString);
   output;

   FindString = substr (longstr,75000,80000);
   FindLen = length(findString);
   output;

run;

title Summary Output from XML_Sample5b;
proc print data=XML_Sample5b;
   var FindString FindLen;
   format FindString $10.;
run;
```

Again, the PRINT procedure confirms the desired substrings were returned.

Summary Output from XML_Sample5b

```
Obs    FindString   FindLen
1      //data.lac   80000
2      9</total_p   80000
3      tion="172"   64267
4      g steps to   80000
5      mnId="2824   80000
6      Config><vi   52491
```

However, the user should note that in either case, the final string will be truncated if the SUBSTRING range extends beyond the MAX length defined for the VARCHAR source string. In the above example, use of a start position of 130,000 and string length of 32767:

```
FindString = substr (longstr,130000,32767);
```

results in a NOTE in the log and a blank value for the returned substring since the command (130,000+32767) exceeds the maximum defined VARCHAR length of 160,000.

```
NOTE: Argument 3 to function SUBSTR('<?xml versio'[12 of 160000 characters
shown],130000,32767) at line 50 column 15 is invalid.
```

CONCLUSION

The new VARCHAR data type in SAS 9.4M5 can simplify string extraction and manipulation within the DATA step for data sources that have strings beyond the traditional 32kb limit for as CHAR variables. The VARCHAR will allow the user to scan, search and substring across the entire long string, up to a maximum length exceeding 500 Million characters.

The process is not without limitations. The VARCHAR does not alter the SAS/ACCESS engine limit on a 32kb maximum string returned from a DBMS. Furthermore, the SAS V9 engine will not save the VARCHAR data type; although alternative methods can be used to save the data for later use. Also, users must exercise caution when using these longer strings since limitations on string functions can cause unexpected results.

Users who need to process longer string data within the SAS DATA step can benefit from the new VARCHAR data type, even when VIYA and CAS are not available as part of the SAS installation.

REFERENCES

Schmitz, John, 2017. “Extraction and Use of Text Strings With SAS® When Source Exceeds the 32K String Length Limit.” *Midwest SAS Users Group Conference Proceedings,* St Louis MO.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

John Schmitz
Luminare Data LLC
John.Schmitz@LuminareData.com
www.LuminareData.com