Paper BB12-2014

From Wide to Tall: A Proc SQL Approach to Transposing

Brandon Crosser, University of Kansas Medical Center, Kansas City, KS

ABSTRACT

In many Extract, Transform, and Load (ETL) situations a large scale conversion is needed from a “wide” table structure to a “tall” table structure. This often occurs in transformation from a denormalized to normalized architecture. This paper presents techniques to gain efficiency in doing such loading with particular attention to reading and writing between SQL Server database tables. The techniques described within ensure that the process is table-driven and extensible with minimal code edits. A contrast of the technique with a PROC TRANSPOSE approach will be presented.

INTRODUCTION

In order the lay the foundation for this particular scenario, a description of the initial and resulting datasets will first be described. The key transforming mechanism will use an INSERT statement within PROC SQL rather than PROC TRANSPOSE. As this will be done repeatedly, it lends itself well to the usage of a macro to accomplish the task. This macro will be called within a DATA step using a CALL EXECUTE command. This results in the solution remaining table driven rather than hardcoding the call to the macro repeatedly.

SETTING THE STAGE

INITIAL DATASET

The initial dataset of this scenario is “wide”, in that it contains numerous columns. These variables consist of a few key identifying variables and numerous fact variables. These fact variables may or may not be of interest for the final resulting transformed file. This type of file often results from the merging of several independent datasets to create a single dataset. This layout does have its own set of benefits as it lends itself well to horizontal analysis and computations based between variables.

Key1	Key2	Key3	Var1	Var2	Var3	Var4	Var5	Var6	Var7	Var8	Var9	Var10
1	1	1	1.111	2.111	3.111	4.111	5.111	6.111	7.111	8.111	9.111	10.111
1	1	2	1.112	2.112	3.112	4.112	5.112	6.112	7.112	8.112	9.112	10.112
1	1	3	1.113	2.113	3.113	4.113	5.113	6.113	7.113	8.113	9.113	10.113
1	2	1					5.121	6.121	7.121	8.121	9.121	10.121
1	2	2	1.122	2.122	3.122	4.122	5.122					
1	2	3	1.123	2.123	3.123	4.123	5.123	6.123	7.123	8.123	9.123	10.123
1	3	1	1.131	2.131	3.131	4.131	5.131	6.131	7.131	8.131	9.131	10.131
1	3	2	1.132	2.132	3.132	4.132	5.132	6.132	7.132	8.132	9.132	10.132
1	3	3	1.133	2.133	3.133	4.133	5.133	6.133		8.133	9.133	10.133

Figure 1. Initial Dataset Snippet

RESULTING DATASET

The target file of this scenario is a much “taller” file than the initial dataset. The desired output consists of all of the necessary keys from the initial dataset and a single column that contains the values of all of the desired fact variables from the initial dataset. An additional key is used to distinguish the values of the initial fact variables. This type of layout lends itself well to analysis using BY groups as well as many data warehouse applications.

Key1	Key2	Key3	MetricID	MetricValue
1	1	1	2	2.111
1	1	2	2	2.112
1	1	3	2	2.113
1	2	2	2	2.122
1	2	3	2	2.123
1	3	1	2	2.131
1	3	2	2	2.132
1	3	3	2	2.133
1	1	1	3	3.111
1	1	2	3	3.112
1	1	3	3	3.113
1	2	2	3	3.122
1	2	3	3	3.123
1	3	1	3	3.131
1	3	2	3	3.132
1	3	3	3	3.133

Figure 2. Resulting Dataset Snippet

SUPPORT TABLE

A single lookup table will serve as the linchpin of the entire process. This table lists the variables in the initial dataset. This greatly reduces the maintenance of the process by ensuring that it remains table-driven rather than hard-coded. At a minimum, this table must include the new key that is used to identify and distinguish the fact variables from each other and the character text of the variable names within the initial dataset. This table may also contain various metadata and supplementary information for each variable. This information will vary depending upon need, but may include information such as text descriptions, abbreviations, analytic notes, and reporting characteristics. In this particular situation, a flag will be used to signal whether each variable is needed for the final dataset. Another benefit in having a standalone support table is that it can be maintained outside of SAS by various other applications and non-SAS users.

MetricID	VarName	Description	ReportCharacteristicFlag1	ReportCharacteristicFlag2	InUseYN
1	Var1	Variable Description 1	1	1	0
2	Var2	Variable Description 2	1	0	1
3	Var3	Variable Description 3	0	1	1
4	Var4	Variable Description 4	0	1	1
5	Var5	Variable Description 5	0	1	1
6	Var6	Variable Description 6	0	0	1
7	Var7	Variable Description 7	1	1	1
8	Var8	Variable Description 8	0	0	1
9	Var9	Variable Description 9	1	0	0
10	Var10	Variable Description 10	0	0	0

Figure 3. Support Table Snippet

APPLYING PROC SQL

Using a PROC SQL INSERT statement allows for all of or a subset of columns to be selected from the initial table and placed into the target table one at a time. For this purpose, each execution of a PROC SQL statement will INSERT the key identifying variables, the new key variable distinguishing each fact variable, and a single variable

from the original dataset. This process is then repeated using the same key identifying variables and iterating through all of the desired fact variables. In this example, the first execution will insert select Key1, Key2, Key3, MetricID, and Var2 from the initial dataset (note that Var1 has InUseYN=0). The second execution will insert Key1, Key2, Key3, MetricID, and Var3.

In order to use this approach, there must first be an existing destination table to receive the inserted data. In some cases, this table may already contain data. If this is the case, then the process will append the values from the initial dataset to the existing observations within the destination table. If, however, a new table containing only the data from the initial dataset is desired, then a "shell" for the destination table must first be created with all of the appropriate data formats and contain no observations. To create a shell destination table based upon an existing populated table, apply the LIKE statement of PROC SQL.

```
Proc SQL;
Create Table Work.Tall
Like Example.Tall
;
Quit;
```

ADDING AND EXECUTING A MACRO

The PROC SQL will need to execute for each desired fact variable of the initial dataset. Obviously, creating separate pieces of PROC SQL code for each desired measure would result in lengthy code and lead to difficulties in maintenance. Simply adding a macro that contains the PROC SQL statement, would result in a more streamlined approach. However, this solution would result in a call to the macro for each fact variable. This would require edits to the program as new fact variables were introduced or retired. This is where the support table comes in. By maintaining all of this information in a standalone table, the process can be maintained simply by making edits to the data within that table. Essentially, the goal is to call the LoadTall macro for each observation in the LU_Var dataset. This is accomplished by using a CALL EXECUTE statement within a DATA step pointed at the LU_Var dataset. The result is that the values for MetricID and VarName from the LU_Var dataset are passed to the macro for each desired variable

```
%Macro LoadTall(_VarName_, _MetricID_);
Proc SQL NoPrint;
Insert Into Tall Select
Key1,
Key2,
Key3,
&_MetricID_ as MetricID,
&_VarName_ as MetricValue
From Wide
Where &_VarName_ ^= .
;
Quit;
%Mend;


Data _NULL_;
Set LU_Var;
Call Execute('%LoadTall('||VarName||','||MetricID||')');
Where InUseYN = 1
;
Run;
```

At this point, the values and scope of the variables is very important to follow. The macro is called for each observation of the LU_Var table. This results in the values from the LU_Var table being passed to the PROC SQL statement. The ID value of MetricID is assigned for the TALL table. More importantly, the text value of VarName from the LU_Var table is passed to the PROC SQL statement within the LoadTall macro. When the statement receives the text value of the variable name, it selects the corresponding column from the WIDE dataset. The NOPRINT option is

added simply to reduce the repetition of the syntax within the log. The WHERE statement within the DATA step controls which observations of LU_Var call the macro.

EFFICIENCIES OVER PROC TRANSPOSE

A major benefit of this approach over using PROC TRANSPOSE, is that the data are transposed one column at a time rather than in a single large operation. This incremental approach effectively partitions a very large dataset into multiple smaller datasets for loading. This can greatly increase the efficiency of loading data. For instance, if the target of the load is a SQL Server database, fewer small data inserts is much quicker than single large one[1]. There is also more informative logging, as the SAS log will identify individual variables that fail to load and continue on to the next.

ADDITIONAL EFFICIENCIES

Since the PROC SQL statement needs to read the entire initial dataset, the most important addition to the code is the SASFile statement. This will read the initial file into memory prior to the load, thus drastically increasing the overall efficiency of the process. Considerations should also be made regarding the location of all datasets. With the increasing popularity of solid state computing, local processing has made tremendous strides in efficiency and speed. Many times a file may be located on a networked location or shared drive. Repeated reading and writing over such a network is not ideal. Options within LIBNAME statements, such as BULKLOAD=YES, can also greatly improve performance.

CONCLUSION

The techniques demonstrated here fit a very particular scenario. However, many of the concepts can be applied to a wide array of other situations. In particular, the use of a CALL EXECUTE statement within a DATA step can have many other applications. This allows for values within a dataset to be passed to any macro for any given number of observations of that dataset. Also, as demonstrated by this process, the values passed to that macro can even be variable names of a separate dataset. If a particular independent operation is desired for several variables of dataset, the usage of PROC DATASETS can create a list of variable names to pass to a macro for the execution of the desired operation.

REFERENCES

[1] http://technet.microsoft.com/en-us/library/dd425070%28v=sql.100%29.aspx

CONTACT INFORMATION

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

Name: Brandon Crosser
Enterprise: University of Kansas Medical Center
Address: 3901 Rainbow Rd. M/S 3060
City, State ZIP: Kansas City, KS 66160
Work Phone: 913-945-6653
E-mail: bcrosser@kumc.edu