Paper SAS 1866-2015

Now That You Have Your Data in Hadoop, How Are You Staging Your Analytical Base Tables?

Steven Sober, SAS Institute Inc.

ABSTRACT

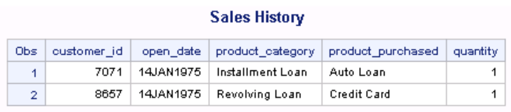
Well, Hadoop community, now that you have your data in Hadoop, how are you staging your analytical base tables? In my discussions with clients about this, we all agree on one thing: Data sizes stored in Hadoop prevent us from moving that data to a different platform to generate the analytical base tables. To address this problem, I want to introduce to you the SAS® In-Database Code Accelerator for Hadoop.

INTRODUCTION

It's not widely understood that analytics consumes a fundamentally different set of data than does query and reporting. Most business professionals are surprised when they learn that not just any data can be used for predictive modeling. Actually, the modeling data is required to be in a special format, usually one-record-per-subject, before it can be leveraged. So how does this data transformation process work?

Operational data can consist of any entity or subject (for example, customer, company, region, product, date, and so on) that has sequential or multiple records. This type of operational data, however, needs to be both summarized and converted into a single-record or row-per-subject data set before any predictive modeling work can begin. We call this data manipulation process a "transposition" of the data. For the purposes of this paper, the resulting output is called an analytic base table (ABT).

Suppose that we wanted to know, in a business setting for a large banking institution, which product categories over the lifetime of data collected are correlated and provide the bank with the best cross-sale opportunities. In order to answer this type of question, our raw transactional data needs some additional reorganization and summarization. Here is a simple example of a transactional data set that needs to be transposed before data mining can occur:

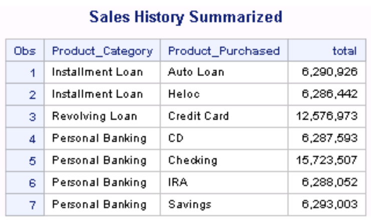


Sales History

Obs	customer_id	open_date	product_category	product_purchased	quantity
1	7071	14JAN1975	Installment Loan	Auto Loan	1
2	8657	14JAN1975	Revolving Loan	Credit Card	1

Output 1. Sales History Table

After you summarize the numeric field named 'quantity' by 'product_category' and 'product_purchased' (character fields), the output will look similar to this:

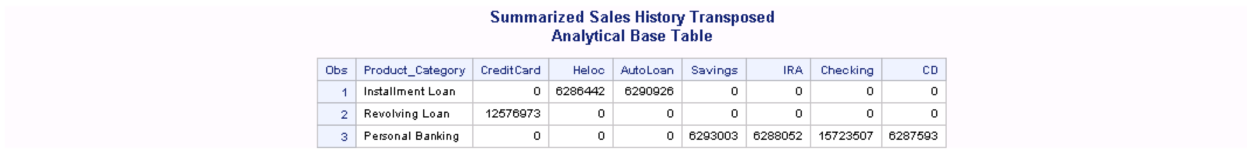


Sales History Summarized

Obs	Product_Category	Product_Purchased	total
1	Installment Loan	Auto Loan	6,290,926
2	Installment Loan	Heloc	6,286,442
3	Revolving Loan	Credit Card	12,576,973
4	Personal Banking	CD	6,287,593
5	Personal Banking	Checking	15,723,507
6	Personal Banking	IRA	6,288,052
7	Personal Banking	Savings	6,293,003

Output 2. Sales History Summarized

While summarization (above) is the first step, the data still needs to be transposed. This means that the former 'product_purchased' column now becomes a distinct category column, and duplicate records are eliminated to produce a unique file, in this case by 'product_category'. After the transposition occurs, the data will appear as follows:



Summarized Sales History Transposed
Analytical Base Table

Obs	Product_Category	CreditCard	Heloc	AutoLoan	Savings	IRA	Checking	CD
1	Installment Loan	0	6286442	6290926	0	0	0	0
2	Revolving Loan	12576973	0	0	0	0	0	0
3	Personal Banking	0	0	0	6293003	6288052	15723507	6287593

Output 3. Summarized Sales History Transposed

Now that we have staged the analytical base table, we can begin the next set of iterative work: namely exploring the transposed data to see how well it supports our proposed predictive analysis. Note that it is common for SAS users to stage analytical base tables that have tens of thousands of columns per subject. The reason for this is to improve the accuracy of the models produced by predictive analytics.

To support our development of an analytical base table we will investigate how SAS® Data Loader for Hadoop will leverage our investment in Hadoop by moving the SAS processes that stage the analytical base table to a MapReduce framework.

CHALLENGES IN PROCESSING HADOOP DATA

Data Volume and Formats

Data volumes stored in Hadoop make the processing of the data challenging. Source tables can easily be in the hundreds of gigabytes or even in terabytes. With these data volumes, you cannot move the data to a different platform. The processing must take place on the Hadoop cluster. Because Hadoop is an open-source environment, there are numerous data formats supported. From a SAS point of view, as long as our source and target tables are support by SAS/ACCESS® Interface to Hadoop or SAS/ACCESS® Interface to Impala, you can process that data in the Hadoop cluster using the MapReduce framework.

Single Input Stream

MapReduce processes are limited to one input stream that makes joining of multiple sources a challenge. To overcome this, we have a few options. If your data is not stored in an SQL-based format, we can use PIG. With the SQL-based formats like Hive and Impala, we can use SQL to accomplish the join. With SQL, we can create a view of the join, and that view can be used as the single input stream to the MapReduce process.

Single Output Stream

MapReduce processes are limited to one output stream. This implies we cannot create multiple output tables in a single pass of the source table.

Transposing Data

Transposing data is a straightforward task that is not trivial to using open-source techniques or SQL. In SAS 9.4 (TS1M3) PROC TRANSPOSE will run via a MapReduce framework. Until then, we can leverage the TRANSPOSE directive in SAS Data Loader for Hadoop when our source tables are stored in Hive. If our source tables are not stored in Hive and they are in a format that the SAS® Embedded Process supports, we can leverage a component of the SAS Data Loader for Hadoop called the SAS In-Database Code Accelerator for Hadoop. This will enable us to write DS2 THREAD and DATA statements that run in a MapReduce framework.

SAS IN-DATABASE CODE ACCELERATOR FOR HADOOP

SAS In-Database Code Accelerator for Hadoop enables us to run DS2 code in a MapReduce framework. DS2 is a new, SAS proprietary programming language that is appropriate for advanced data manipulation, large transpositions, computationally complex programs, scoring models, and BY group processing. DS2 is included with Base SAS® and intersects with the SAS DATA step. It also includes additional data types, ANSI SQL types, programming structure elements, and user-defined methods and packages:

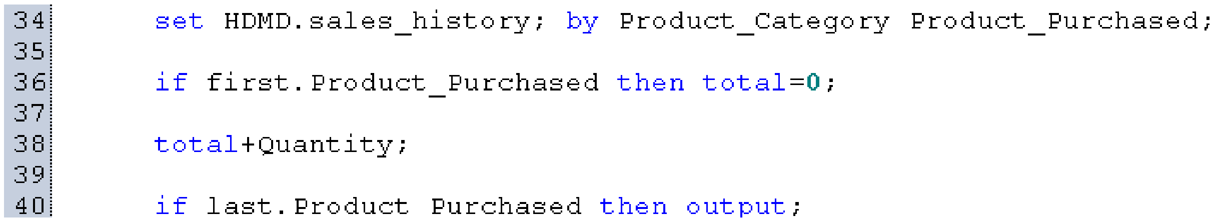
DS2 Data Type	Description
BIGINT	stores a large signed, exact whole number, with a precision of 19 digits. The range of integers is -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. Integer data types do not store decimal values. Fractional portions are discarded
TINYINT	stores a very small signed, exact whole number, with a precision of three digits. The range of integers is -128 to 127. Integer data types do not store decimal values. Fractional portions are discarded.
VARBINARY(n)	stores varying-length binary data, where n is the maximum number of bytes to store. The maximum number of bytes is not required to store each value. If varbinary(10) is specified and the binary string uses only five bytes, only five bytes are stored in the column.
VARCHAR(n)	stores a varying-length character string, where n is the maximum number of characters to store. The maximum number of characters is not required to store each value. If varchar(10) is specified and the character string is only five characters long, only five characters are stored in the column.

Table 1. Partial Listing of DS2 Data Types. (For a complete list, see the *SAS 9.4 DS2 Language Reference*.)

To accomplish the task defined in the introduction of this paper, we will use two approaches. When our source tables are stored in Hive, our preferred approach is to use the SUMMARY and TRANSPOSE directives in SAS Data Loader for Hadoop. When our data is not stored in Hive and it is in a format supported by the SAS Embedded Process, our approach will be to use DS2.

Summary – DS2 Code

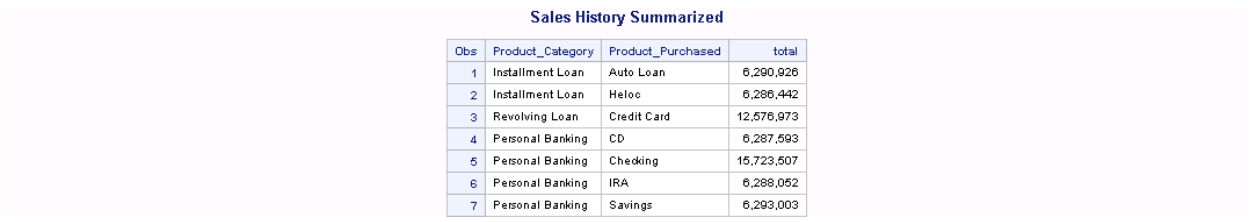
Summarizing data using DS2 is very similar to using a DATA step.



```
34        set HDMD.sales_history; by Product_Category Product_Purchased;
35
36        if first.Product_Purchased then total=0;
37
38        total+Quantity;
39
40        if last.Product Purchased then output;
```

Figure 1. DS2 Code to Summary Sales History

To summarize our data we are using a RETAIN statement for the variable ‘total’. In addition, we are using FIRST. LAST. processing with the BY group ‘product_purchaed’. Review line 36 of Figure 1. We will set the value for ‘total’ to zero for each new occurrence of the BY group. Review line 40 of Figure 1. When we hit the last occurrence of the BY group variable ‘product_purchased’, we will output one row. When this process completes, we will have our summary table:

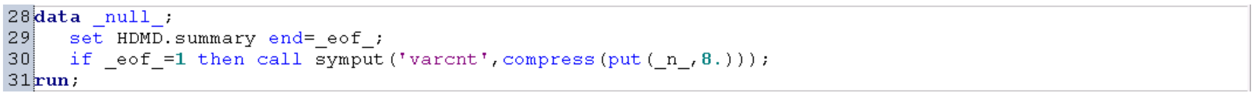


Sales History Summarized

Obs	Product_Category	Product_Purchased	total
1	Installment Loan	Auto Loan	6,290,926
2	Installment Loan	Heloc	6,286,442
3	Revolving Loan	Credit Card	12,576,973
4	Personal Banking	CD	6,287,593
5	Personal Banking	Checking	15,723,507
6	Personal Banking	IRA	6,288,052
7	Personal Banking	Savings	6,293,003

Output 4. Summary Table

Next, we need to set a global macro variable to the total number of rows in our summary table. This macro variable is used in the rename step, which is detailed later in this paper When we transpose our data, we will generate 7 new columns. The names of those columns are the values of 'product_purchased' in Output 4. To obtain the number of rows in the summary table, we will use a DATA step:



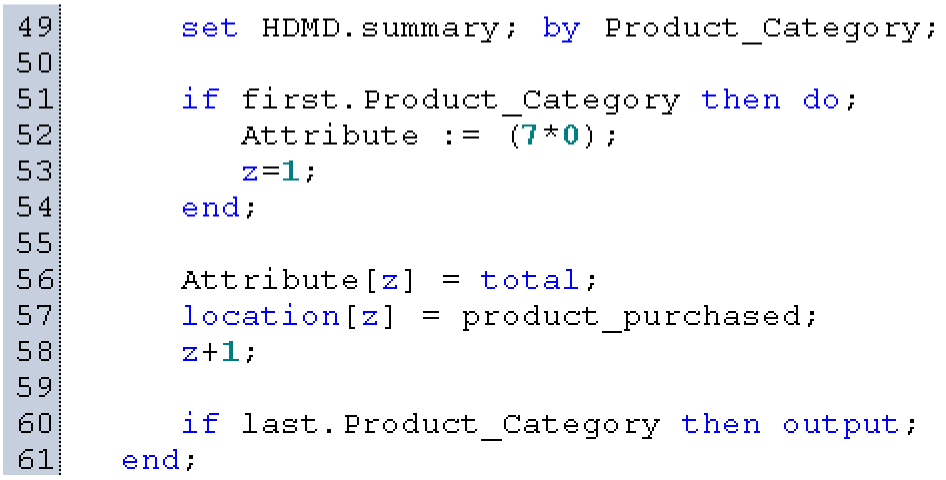
```
28 data _null_;
29    set HDMD.summary end=_eof_;
30    if _eof_=1 then call symput('varcnt',compress(put(_n_,8.)));
31 run;
```

Figure 2. DATA Step to Set Macro Variable &varcnt

Review line 30 of Figure 2. We test for the last record, and, when we find it, we use the CALL SYMPUT statement to create the macro variable '&varcnt'.

Transpose - DS2 array processing

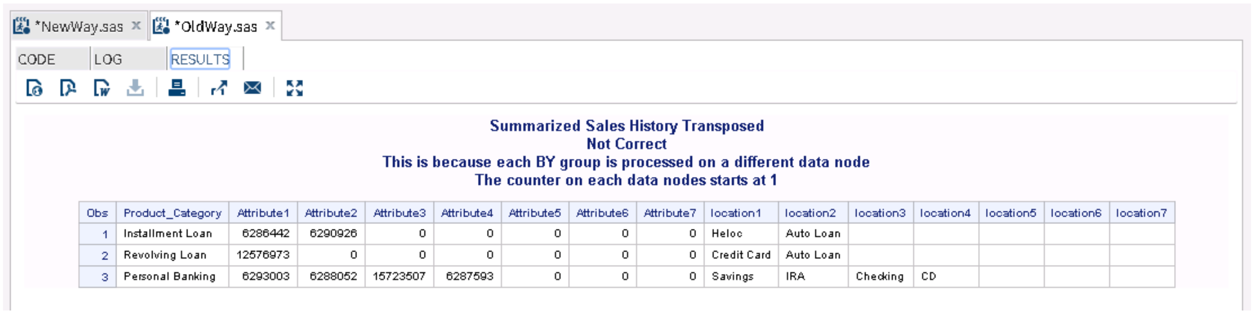
With Hadoop, we must remember that our DS2 code will be processed in a MapReduce framework. Each data node of the Hadoop cluster will execute the DS2 code in parallel and without sharing any information between the data nodes. This impacts array processing, which is required when transposing data using DS2 code. To illustrate this, we will use an approach used by Base SAS programmers to transpose our data. In the DS2 code shown in Figure 3, 'z' is the variable that contains the array position for our two arrays 'attribute' and 'location':



```
      set HDMD.summary; by Product_Category;

      if first.Product_Category then do;
         Attribute := (7*0);
         z=1;
      end;

      Attribute[z] = total;
      location[z] = product_purchased;
      z+1;

      if last.Product_Category then output;
   end;
```

Figure 3. DS2 Code

Using this approach works in a non-distributed computing environment. However, in a distributed computing environment, our transposed table is incorrect:



Summarized Sales History Transposed
Not Correct
This is because each BY group is processed on a different data node
The counter on each data nodes starts at 1

Obs	Product_Category	Attribute1	Attribute2	Attribute3	Attribute4	Attribute5	Attribute6	Attribute7	location1	location2	location3	location4	location5	location6	location7
1	Installment Loan	6286442	6290926	0	0	0	0	0	Heloc	Auto Loan					
2	Revolving Loan	12576973	0	0	0	0	0	0	Credit Card	Auto Loan					
3	Personal Banking	6293003	6288052	15723507	6287593	0	0	0	Savings	IRA	Checking	CD			

Output 5. Incorrect Transposed Table

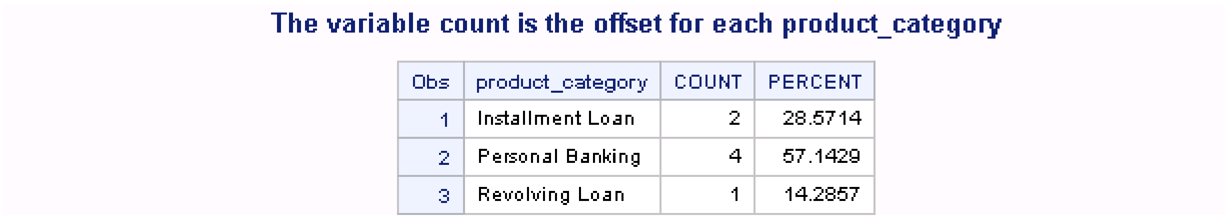
The reason that the table is incorrect is that each BY group is processed on a different data node, and this impacts our array position variable ‘z’. The value of ‘z’ will start at 1 on each data node and produce the incorrect results shown in Output 5.

To generate a correct result, we must alter our DS2 code to add an offset to the variable ‘z’. So each BY group has the correct starting position in the arrays. To accomplish this, we will determine the number of unique values of our variable ‘product_purchased’ for each ‘product_category’:

```
19 filename offset '/opt/LSF_Share/sasss1/SGF2015/offset.sas';
20
21 proc delete data=HDMD.FreqCount;
22 run;
23
24 proc freq data=HDMD.summary noprint ;
25    table product_category / out=HDMD.FreqCount ;
26 run;
27
28 data _null_;
29    length q $32.;
30    file offset;
31    set HDMD.FreqCount;
32    q= "'" || Trim(product_category) || "'";
33    line = 'IF z eq 0 and product_category eq ' || q || ' then Z=' || COUNT ||';';
34    put line;
35 run;
```

Figure 4. Base SAS Code to Calculate the Offset for Array Processing in a Distributed Environment

Review line 24-26 in Figure 4. We are using PROC FREQ to process the summary table and write the results to a table:



The variable count is the offset for each product_category

Obs	product_category	COUNT	PERCENT
1	Installment Loan	2	28.5714
2	Personal Banking	4	57.1429
3	Revolving Loan	1	14.2857

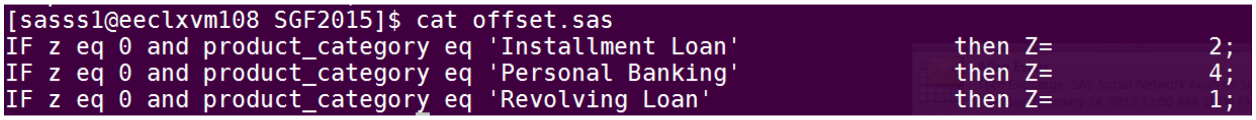
Output 6. Table Created by PROC FREQ

We will now process the table created by PROC FREQ and create a file that contains the IF statements that we need to control which array location we are to populate:

```
data _null_;
   length q $32.;
   file offset;
   set HDMD.FreqCount;
   q= "'" || Trim(product_category) || "'";
   line = 'IF z eq 0 and product_category eq ' || q || ' then Z=' || COUNT ||';';
   put line;
run;
```

Figure 5. Base SAS Code to Generate a File to Control the Offset of the Array Position for Each ‘product_category’

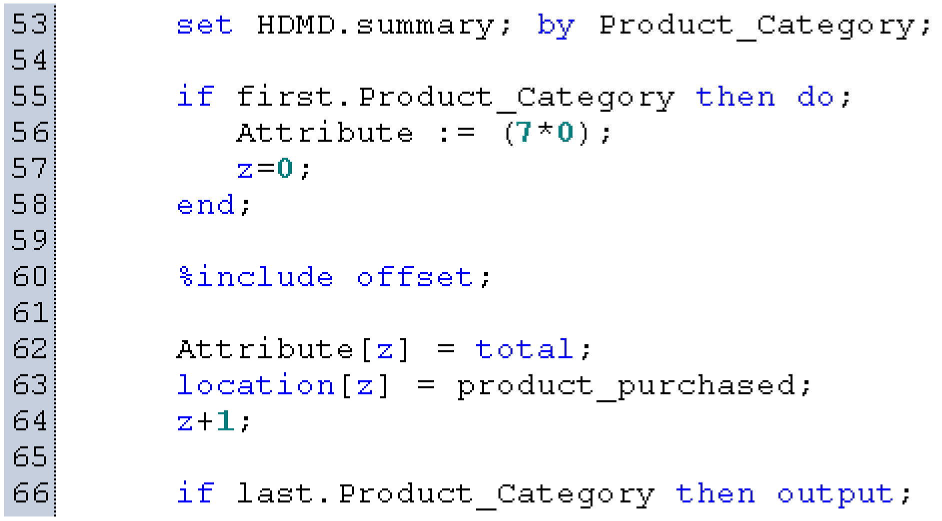
The code in Figure 5 produced the following:



```
[sasss1@eeclxvm108 SGF2015]$ cat offset.sas
IF z eq 0 and product_category eq 'Installment Loan'                  then Z=          2;
IF z eq 0 and product_category eq 'Personal Banking'                  then Z=          4;
IF z eq 0 and product_category eq 'Revolving Loan'                    then Z=          1;
```

Figure 6. Offset for Array Processing in a Distributed Environment

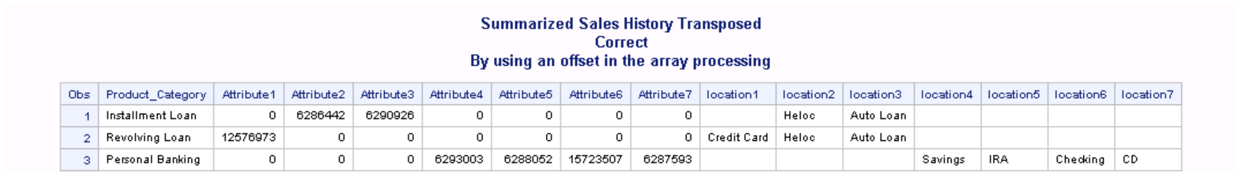
We will now alter our DS2 code to use offsets. Review line 57 of Figure 7. We are initializing the variable 'z' to zero for each BY group (that is, 'product_category'). Review line 60 of Figure 7. This is how we will include the IF statements created in the code shown in Figure 5.



```
53        set HDMD.summary; by Product_Category;
54
55        if first.Product_Category then do;
56           Attribute := (7*0);
57           z=0;
58        end;
59
60        %include offset;
61
62        Attribute[z] = total;
63        location[z] = product_purchased;
64        z+1;
65
66        if last.Product_Category then output;
```

Figure 7. Altered DS2 Code Using Offsets for Array Processing in a Distributed Environment

When the code in Figure 7 is processed, it produces a correct transpose table:



Summarized Sales History Transposed
Correct
By using an offset in the array processing

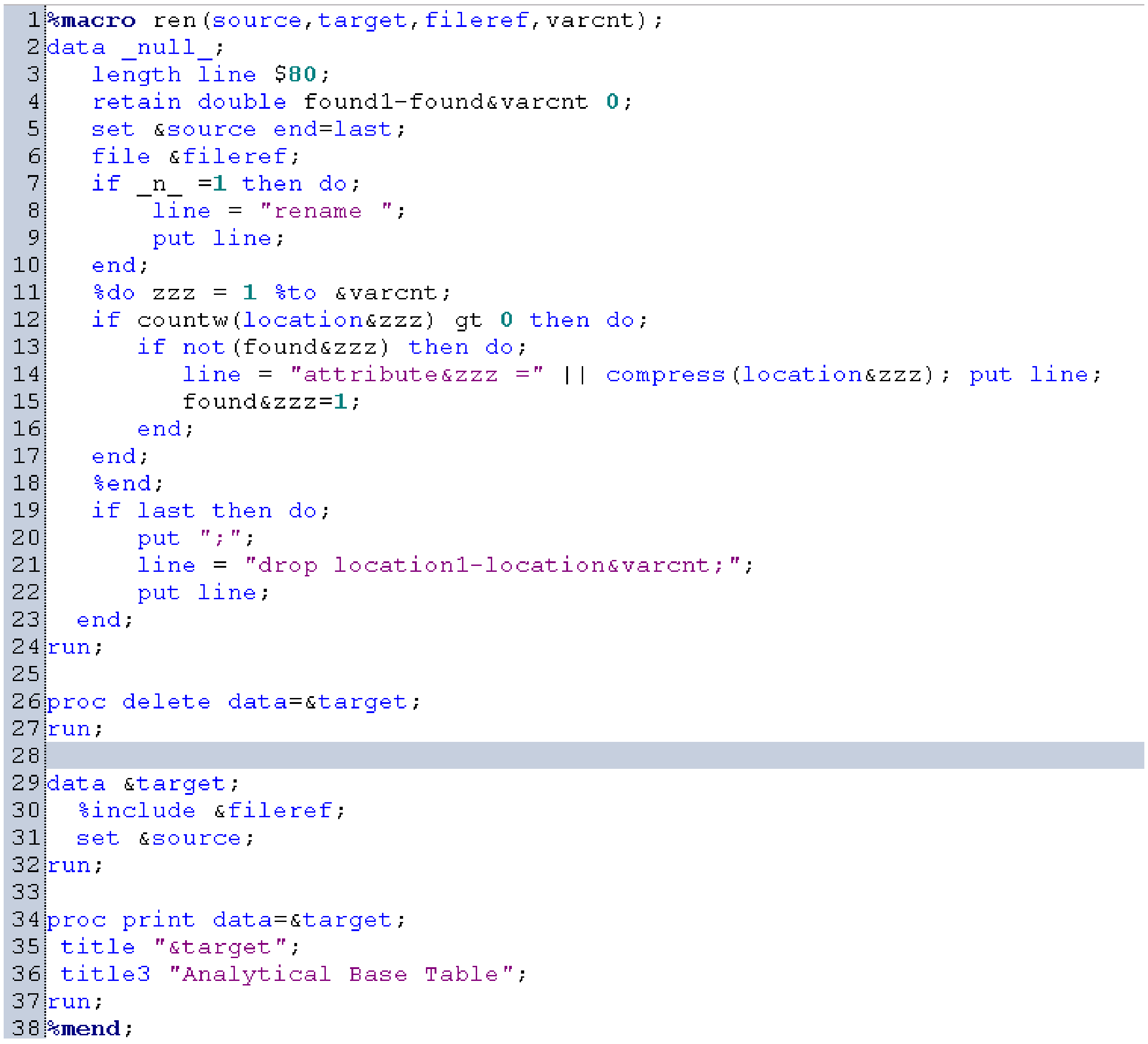
Obs	Product_Category	Attribute1	Attribute2	Attribute3	Attribute4	Attribute5	Attribute6	Attribute7	location1	location2	location3	location4	location5	location6	location7
1	Installment Loan	0	6286442	6290926	0	0	0	0		Heloc	Auto Loan				
2	Revolving Loan	12576973	0	0	0	0	0	0	Credit Card	Heloc	Auto Loan				
3	Personal Banking	0	0	0	6293003	6288052	15723507	6287593				Savings	IRA	Checking	CD

Output 7. Transpose Table using Offsets for Array Processing

Before we give this transpose table to the data scientists we have one more thing to take care of because the names of our transposed columns 'attribute1-attribute7' are meaningless. What we want are the values shown in 'location1-location7'—that is, Credit Card, Heloc, Auto Loan, Savings, IRA, Checking, and CD.

Rename – DATA step

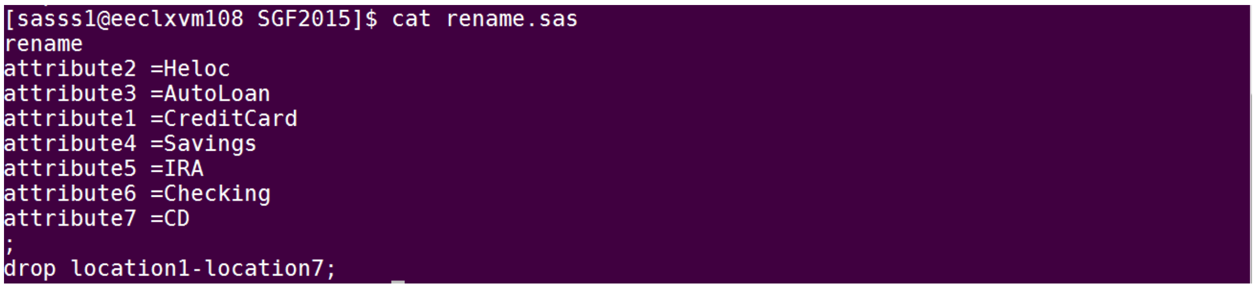
The values in the array 'location1-location7' in Output 7 are the unique values of the variable 'product_purchased'. The best way to accomplish the rename in a distributed environment is to use a DATA step with a RENAME statement. Like DS2, a DATA step is also eligible to run via a MapReduce framework. The statements used in the DATA step will determine if this code is eligible for the MapReduce framework. In addition to the statements, the source and target tables must be stored in Hadoop for this code to be eligible. To accomplish the rename we will create a SAS macro:



```
 1 %macro ren(source,target,fileref,varcnt);
 2 data _null_;
 3    length line $80;
 4    retain double found1-found&varcnt 0;
 5    set &source end=last;
 6    file &fileref;
 7    if _n_ =1 then do;
 8        line = "rename ";
 9        put line;
10    end;
11    %do zzz = 1 %to &varcnt;
12    if countw(location&zzz) gt 0 then do;
13       if not(found&zzz) then do;
14          line = "attribute&zzz =" || compress(location&zzz); put line;
15          found&zzz=1;
16       end;
17    end;
18    %end;
19    if last then do;
20       put ";";
21       line = "drop location1-location&varcnt;";
22       put line;
23   end;
24 run;
25
26 proc delete data=&target;
27 run;
28
29 data &target;
30   %include &fileref;
31   set &source;
32 run;
33
34 proc print data=&target;
35  title "&target";
36  title3 "Analytical Base Table";
37 run;
38 %mend;
```

Figure 8. DATA Step Code to Rename our Transposed Variables

Like our offset code, the macro in Figure 8 will create a file that contains the RENAME and DROP statements. To do this, we will use the macro variable '&varcnt'. (See Figure 2.) Review lines 11-17 in Figure 8. In this step, we will loop through each array location (that is, 'location1-location7'). When we find a word in that array position, we generate the RENAME statement and set a flag to indicate that the RENAME statement has been generated and written to our file. If we do not set this flag, we could generate multiple RENAME statements for the same variable, and that would cause an error when this code is processed.



```
[sasss1@eeclxvm108 SGF2015]$ cat rename.sas
rename
attribute2 =Heloc
attribute3 =AutoLoan
attribute1 =CreditCard
attribute4 =Savings
attribute5 =IRA
attribute6 =Checking
attribute7 =CD
;
drop location1-location7;
```

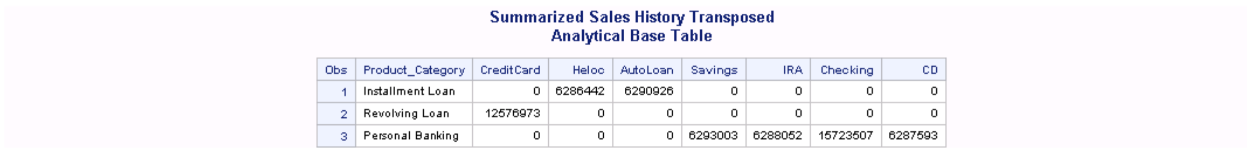
Figure 9. Rename Statements Generated by the Code in Figure 8.

Review line 30 of Figure 8. We are including the file shown in Figure 9. A very positive side effect (meaning that the run time is very fast) of doing the rename with a DATA step is what happens under the covers. A Hadoop method is used to simply copy the source into the target with the new variable names. Figure 10 shows the SAS log information using the SAS OPTION "options sastrace=',,d,d' sastraceloc=saslog;".

```
HADOOP_76: Executed: on connection 4 2038 1424127168 no_name 0 DATASTEP
LOAD DATA INPATH '/tmp/sasdata-2015-02-16-17-52-48-498-e-00012.dlv' OVERWRITE INTO TABLE `TRANSPOSED_ABT` 2039
1424127168 no_name 0 DATASTEP 2040 1424127168 no_name 0 DATASTEP
```

Figure 10. SAS Log Showing the Hadoop Method to Create the Final Transposed Table

We are now ready to hand off our analytical base table to the data scientist team who will apply predictive analytics to it so that they can make well-informed decisions.



Summarized Sales History Transposed
Analytical Base Table

Obs	Product_Category	CreditCard	Heloc	AutoLoan	Savings	IRA	Checking	CD
1	Installment Loan	0	6286442	6290926	0	0	0	0
2	Revolving Loan	12576973	0	0	0	0	0	0
3	Personal Banking	0	0	0	6293003	6288052	15723507	6287593

Output 8. Analytical Base Table

SAS DATA QUALITY ACCELERATOR FOR HADOOP

SAS Data Quality Accelerator for Hadoop (a component of SAS Data Loader for Hadoop) enables us to apply world-class data quality routines to data stored in Hadoop using a MapReduce framework.

Data Quality Accelerator for Hadoop Operations	**Description**	**Input**	**Output**
Casting	Casing applies context-sensitive case rules to text. It operates on character content, such as names, organizations, and addresses.	SAS INSTITUTE DQ_LOWERCASE DQ_UPPERCASE DQ_PROPERCASE	 sas institute SAS INSTITUTE SAS Institute
Extraction	Extraction returns one or more extracted text values, or tokens, as output.	Blue men's long-sleeved button-down collar denim shirt	Color: Blue Material: Denim Item: Shirt
Gender Analysis	Gender analysis evaluates the name or other information about an individual to be determined.	Jane Smith Sam Adams P. Jones	F M U
Identification Analysis	Identification analysis returns a value that indicates the category of the content in an input character string.	John Smith SAS Institute Inc.	NAME ORGANIZATION
Matching	Matching analyzes the input data and generates a matchcode for the data.	Gidley, Scott A Scotty Gidleigh Mr Scott Gidlee Jr. Mr Robert J Brauer Bob Brauer	XYZ$$$ XYZ$$$ XYZ$$$ ABC$$$ ABC$$$

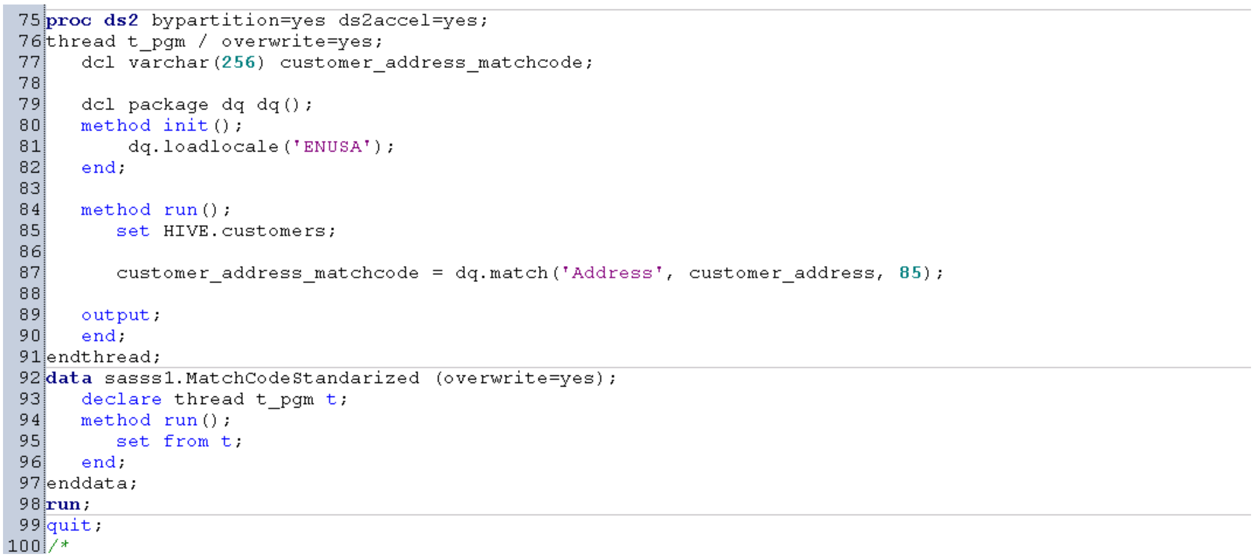
Data Quality Accelerator for Hadoop Operations	Description	Input	Output
Parsing	Parsing segments a string into semantically atomic tokens.	Mr. Roy G. Biv Jr	Prefix: Mr. Given Name: Roy Middle Name: G. Family Name: Biv Suffix: Jr
Pattern Analysis	Pattern analysis returns a simple representation of a text string's character pattern, which can be used for pattern frequency analysis in profiling jobs.	919-677-8000 NC	999-999-999 AA
Standardization	Standardization generates a preferred standard representation of data values.	N car 919.6778000 Smith, Mister James	NC (919) 677–8000 Mr. James Smith

Table 2. SAS Data Quality Accelerator for Hadoop Operations

Data Quality Operations

Let's review an example of the SAS Data Quality Accelerator for Hadoop. Like SAS In-Database Code Accelerator for Hadoop the SAS In-Database Data Quality Accelerator uses DS2 to run data quality operations in a MapReduce framework.

For our example, we will use the "Matching" operator. (Review Matching in Table 2). Matching analyzes the input data and generates a match code for the data. Match codes are encoded representations of character values that are used for analysis, transformation, and standardization of data. Review lines 79-82 in Figure 11. On line 79, we are declaring to our DS2 program everything that we need to cleanse our data in a MapReduce framework. On line 81, we are loading the Quality Knowledge Base (QKB). The QKB contains all of the defined rules, criteria, and data by which analysis and cleansing can be performed for data. Review line 87 of Figure 11; DQ.MATCH is the data quality operation that we are applying to the variable 'Address". DQ.MATCH is using the rules defined in the Quality Knowledge Base for "customer address" with a sensitivity setting of 85. The match code generated by this operation is returned to the variable 'customer_address_matchcode'.



```
 75 proc ds2 bypartition=yes ds2accel=yes;
 76 thread t_pgm / overwrite=yes;
 77    dcl varchar(256) customer_address_matchcode;
 78
 79    dcl package dq dq();
 80    method init();
 81        dq.loadlocale('ENUSA');
 82    end;
 83
 84    method run();
 85       set HIVE.customers;
 86
 87       customer_address_matchcode = dq.match('Address', customer_address, 85);
 88
 89    output;
 90    end;
 91 endthread;
 92 data sasss1.MatchCodeStandarized (overwrite=yes);
 93    declare thread t_pgm t;
 94    method run();
 95       set from t;
 96    end;
 97 enddata;
 98 run;
 99 quit;
100 /*
```

Figure 11. SAS In-Database Data Quality Accelerator Example

CONCLUSION

The SAS In-Database Code Accelerator and the SAS Data Quality Accelerator execute the DS2 DATA and THREAD statements in Hadoop to drive BY group processing and in-database data quality operations. This paper discussed the techniques for writing SAS code to prepare data. These same coding techniques are also leveraged by SAS Data Loader for Hadoop. This product provides directives and an easy to use interface that enables a SAS user to quickly leverage their Hadoop cluster without the need to write code.

RESOURCES

- SAS Institute, Inc. 2015. *SAS® 9.4 DS2 Language Reference, Fourth Edition*. Available at http://support.sas.com/documentation/onlinedoc/base/index.htm
- SAS Institute, Inc. 2015. *SAS® 9.4 In-Database Products: User’s Guide, Fifth Edition*. Available at http://support.sas.com/documentation/onlinedoc/indbtech/index.html
- SAS Institute, Inc. DS2 Programming: Essentials (training course). Available at https://support.sas.com/edu/schedules.html?id=1798

ACKNOWLEDGMENTS

The author would like to thank Carlos Lara and Phil Weiss for their insights on staging analytical base tables for predictive analytics.

CONTACT INFORMATION

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

Steven Sober

100 SAS Campus Drive

Cary, NC 27513

SAS Institute Inc.

919-677-8000

Steven.Sober@sas.com