Paper 120-2011

# SAS® Macro to Consolidate Data from Multiple Cards with Varying Time Validity but Belonging to a Single Account

Sohail Mohammad, Razor – An NSI Company, Addison, Texas, USA

Goutam Chakraborty, Oklahoma State University, Stillwater, OK, USA

## ABSTRACT

Many business-to-business (B2B) marketers use loyalty cards to recognize their loyal customers and give them benefits for their continued patronage. A B2B customer typically has a single corporate account with the seller. But, the customer often wants multiple loyalty cards that it can distribute to its employees who will be using the seller's product and services for earning loyalty points. The loyalty cards points are often accrued over varying time frames and sometimes cards become inactive or lost and need to be replaced. Problems arise in consolidating loyalty points in situations where cards are anonymous and lost or inactive cards are simply replaced by new ones. We develop and report a SAS Macro to solve this problem.

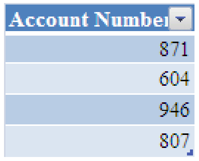
## INTRODUCTION

Loyalty cards are most commonly used as an element of customer relationship management (CRM). A national retailer of goods and services chose to issue *anonymous* loyalty cards to all of its business customers because of customers' privacy concerns with the data. A typical business customer of this retailer had many employees using the retailer's goods and services. Naturally, each employee was therefore issued a loyalty card so that he or she could accrue points and redeem those for benefits. A loyalty program customer's card could be misplaced or lost or given to another employee in the same company. In all such situations, customer would call the retailer who would simply issue a new card but not cancel the old card. Sometimes customers will continue to use only the new card. Other times, customer will continue to use both the old and the new card. If each card was tied to the name of customer's employees or if the retailer could cancel the old card while issuing a new card, then transferring and consolidating all the transactions of a particular customer would have been straightforward. This created a non-trivial problem of mapping back all the transactions done by a customer to just one account irrespective of which card (old or new) she used to do the transaction. This paper discusses about a SAS® Macro Program that solves this problem.

## Cards Transfer Issue

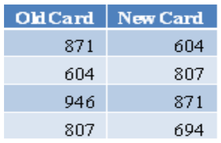
CRM programs these days are characterized by the use of loyalty cards. The customers find the cards easy to use and carry. But a typical problem that arises is that the cards are lost or stolen. The client we were working with would replace a new card carrying a new account number in place of the old one. But all the transactions that a customer performed with the old card were not mapped back to new number. The database did not have a unique customer number to identify a customer rather the customer was identified with the card number he was carrying. If the card was lost and a new card with a different number was issued, in the database the card would be considered as if belonging to a new or different customer. To complicate matters further the old card was not deactivated and the customer could use them again when he found it. The client is in B2B marketing where same account loyalty card may be used by multiple persons working in the company. That is why the old card is not deactivated when a new card is issued. Consequently, a customer would continue using multiple cards over different time frames: some transactions with one card and some transactions with other cards with sometimes overlapping time frames and sometimes non-overlapping time frames. An initial analysis of the data suggested that there were few customers who were using as many as 18 different cards! In order to make sure that proper business metrics are reported for each customer, it was important that all the transactions made using different cards be mapped to a single card. The client did not have any mechanism during data capture to map those transactions to a single account number. All our client had was a text file where the old and new account numbers were noted down. While such primitive arrangements may seem strange in this age of supercomputing, it is not uncommon when cards handling and replacement are cheaply outsourced to a third-party fulfillment company. Because there were multiple instances of card transfers by the same customer it was very difficult to manually map those transactions using the text file.

Below we present an example which will illustrate the problem and the logic that was used to solve the problem. The account numbers shown in the snapshot below belonged to one customer.



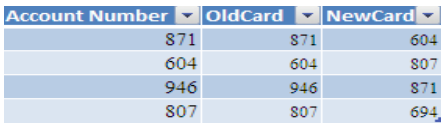
| Account Number |
|---|
| 871 |
| 604 |
| 946 |
| 807 |

The table above is a snapshot from the transaction dataset. The table below is a snapshot from the cards transfer file.



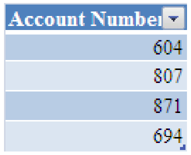
| Old Card | New Card |
|---|---|
| 871 | 604 |
| 604 | 807 |
| 946 | 871 |
| 807 | 694 |

In order to capture the transactions in one account we need to first merge the two datasets on matching AccountNumber and OldCard values. The merged table looks like as shown below. The merging is done in such a way that only the contributing records from transaction dataset are included in the merged dataset.



| Account Number | OldCard | NewCard |
|---|---|---|
| 871 | 871 | 604 |
| 604 | 604 | 807 |
| 946 | 946 | 871 |
| 807 | 807 | 694 |

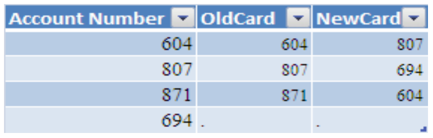
Once the datasets are merged, we needed to replace the AccountNumber with NewCard value whenever the NewCard column is not null. The transaction dataset after replacement would look like as shown below.



| Account Number |
|---|
| 604 |
| 807 |
| 871 |
| 694 |

We realize that a single iteration will not be sufficient because of the possibility of other numbers in the OldCard column which are same as account numbers in transaction dataset. So, we needed to merge and replace again.

Result after merging:



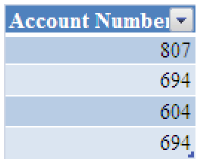
| Account Number | OldCard | NewCard |
|---|---|---|
| 604 | 604 | 807 |
| 807 | 807 | 694 |
| 871 | 871 | 604 |
| 694 | . | . |

It can be seen that the last row has missing values in ‘OldCard’ and ‘NewCard’ columns as the corresponding ‘AccountNumber’ value was not found in the ‘OldCard’ column of the cards transfer file to do the match merge.

We follow the same rule of merging and replacing until all the transactions are captured under one account number. Ideally, the stopping criteria would be when the number of non-missing values after merging would go down to zero but in our case it would not e because of the back and forth looping between the old card and new card values. That is, suppose a record had a value of 1 in old card and 2 in new card then another record had a value of 2 in old card and 1 in new card. So, the stopping criteria in our case would be when the number of non-missing values starts increasing from the previous iteration as that would mean that all the card numbers which had to be mapped to one particular account was achieved in the previous iteration and in the current iteration it has gone back to original stage.

The merging and replacing, for the example we were discussing above, would look like as shown below until all the numbers are captured in one account number.

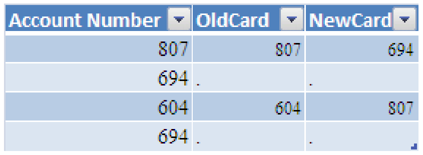
Result after replacing the matching values with the new card values



| Account Number |
|---|
| 807 |
| 694 |
| 604 |
| 694 |

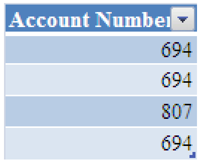
Similarly the results after successive iterations will look as below:

After merging



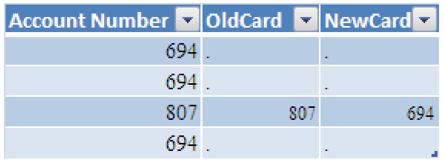
| Account Number | OldCard | NewCard |
|---|---|---|
| 807 | 807 | 694 |
| 694 | . | . |
| 604 | 604 | 807 |
| 694 | . | . |

After replacement



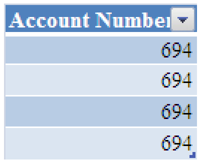
| Account Number |
|---|
| 694 |
| 694 |
| 807 |
| 694 |

After merging



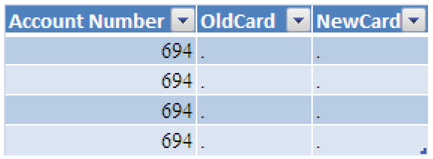
| Account Number | OldCard | NewCard |
|---|---|---|
| 694 | . | . |
| 694 | . | . |
| 807 | 807 | 694 |
| 694 | . | . |

After replacement



| Account Number |
|---|
| 694 |
| 694 |
| 694 |
| 694 |

After merging



| Account Number | OldCard | NewCard |
|---|---|---|
| 694 | . | . |
| 694 | . | . |
| 694 | . | . |
| 694 | . | . |

As can be seen from the above tables, we needed to write five code blocks of merging and four code blocks for replacement for this example. After each replacement we would need to sort the data again for merging. After each merge we would need to count the number of non-missing values and compare it with the previous non-missing values. All these when one customer had just four extra cards which were not deactivated. As we pointed out earlier, there were instances in the data where customers were using as many as 18 cards. Fixing that would require quite a lot of time and a lot of manual intervention if done with normal SAS Program.

Below we present a SAS® macro program that would do the entire process described above using very few lines of code and with minimal manual intervention. The whole logic is preserved in the code and there is no need to count and compare the number of non-missing values after each merge and replace block and then decide if an extra merge and replace code block is needed or not.

```
%macro preloop(lib, data, cardsdata);
PROC SORT Data = &lib..&cardsdata;
By AccountNumber;
Run;
PROC SORT Data = &lib..&data;
By AccountNumber;
Run;
DATA &lib..&data._cards0;
Merge &lib..&data (in = intype) &lib..&cardsdata (in = incards);
By AccountNumber;
If in&type;
Run;
%mend preloop;
%preloop(paper, transaction, cardstransfer_final)


%macro loop(lib, data, cardsdata);
       %local i;
       %let i = 0;
       %do %until (&countafter gt &countbefore);
              PROC SQL;
              Select count(distinct(NewAccountNumber)) into :countbefore
              From &lib..&data._cards&i.
              Where NewAccountNumber ne .;
              Quit;

              %let i = %eval(&i + 1);
              %put i = &i;
              Data &lib..&data._cards&i;
              %let i = %eval(&i - 1);
              %put i = &i;
              Set &lib..&data._cards&i;
              If NewAccountNumber ne . then AccountNumber = NewAccountNumber;
              Drop NewAccountNumber;
              Run;

              %let i = %eval (&i + 1);
              %put i = &i;
              Proc sort data = &lib..&data._cards&i;
              By AccountNumber;
              Run;
              %let i = %eval(&i + 1);
              %put i = &i;
```

```
             Data &lib..&data._cards&i;
             %let i = %eval(&i - 1);
             %put i = &i;
             Merge &lib..&data._cards&i (in = intype) &lib..&cardsdata (in = incards);
             By AccountNumber;
             If intype;
             Run;
             %let i = %eval(&i + 1);
             %put i = &i;
             PROC SQL;
             Select count(distinct(NewAccountNumber)) into :countafter
             From &lib..&data._cards&i.
             Where NewAccountNumber ne .;
             Quit;
             %end;
%mend loop;
%loop(paper, transaction, cardstransfer_final)
```

## Explanation of Macro Logic

In order to compare the counts of non-missing values of NewAccountNumber after each iteration, we are using a %DO %UNTIL loop. The %DO %UNTIL will check for the condition **after** the loop has run. The parameters for the loop are two macro variables ("&countbefore" and "&countafter") which we are creating inside the loop using the "into :" clause of PROC SQL. The counts before the replacement and after the replacement are stored in these two macro variables. But during the first iteration, the values of non-missing NewAccountNumber can be calculated only if the data is merged. For this purpose we have written the 'preloop' macro. The merged data is suffixed with the name "_cards0". We have created a local macro variable "&i" in the 'loop' macro and initialized it to zero so that we can use it as a suffix in the dataset name (to match with the merged data created in the 'preloop' macro) when we are counting the number of distinct non-missing NewAccountNumber values for the first time. The value of local macro variable is increased or decreased appropriately, based on the need, using %EVAL. The %PUT helps in the debugging process and will help in determining which unwanted datasets (created as part of the macro) could be safely deleted. For example, at the end of the macro execution if you find that the value of 'i' is 12, then all datasets except whose suffix is 10 may be deleted.

## Conclusion

Loyalty programs are an essential part of any customer retention initiative and these programs are characterized by loyalty cards. In order to measure the effectiveness of the loyalty program it is important that accurate business metrics are reported. The typical cards transfer issue which was presented here show one of the many hurdles in performing accurate analyses. The SAS® Macro Program, making use of PROC SQL and DO UNTIL loop shows one of the many ways in which the problem of multiple cards usage by same customer can be handled. This simple method requires minimal manual intervention and scales up very well.

## References

1. Ronald Fehd. Do Which? Loop, Until or While? A Review of Data Step and Macro Algorithms. In Proceedings of the SAS® Global Forum, 2007. URL: http://www2.sas.com/proceedings/forum2007/067-2007.pdf
2. SAS® online documentation. URL: http://support.sas.com/documentation/onlinedoc/91pdf/index.html

## Contact Information and Brief Bios

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

Name: Sohail Mohammad
Company: Razor – An NSI Company
Address: 15851 Dallas Parkway
City, State ZIP: Addison, Texas, 75001
Work Phone: 972-663-1209
E-mail: Sohail.mohammad@razordriven.com
Web: www.razordriven.com

Sohail Mohammad is a Decision Science Analyst in the Marketing Science department at Razor - an NSI Company. Razor's Insights, Strategy and Database groups collaborate to provide a breadth and depth of business and customer analytics and insights for each client.

Name: Goutam Chakraborty
Enterprise: Oklahoma State University
Address: 419A Spears School of Business
City, State ZIP: Stillwater, OK, 74074
Work Phone: 405-744-7644
E-mail: Sohail.mohammad@razordriven.com
Web: http://spears.okstate.edu/home/goutamc/

Dr. Goutam Chakraborty is a professor of marketing and founder of SAS® and OSU data mining certificate program at Oklahoma State University. He has published in many journals like Journal of Interactive Marketing, Journal of Advertising Research, Journal of Advertising, Journal of Business Research, etc. He has chaired the national conference for direct marketing educators for 2004 and 2005 and co-chaired M2007 data mining conference. He is also a Business Knowledge Series instructor for SAS®