Paper FF-06

Merging into Hash:
Some Practical Examples of Converting MERGE Statements into Hash Objects

Ying Liu, Toronto, ON, Canada

ABSTRACT

Merging data from two or more datasets is a common process as we manipulate our data for reporting and analysis. Prior to SAS® version 9, the MERGE statement was the most common approach to accomplish this task for a data step programmer. Although MERGE is effective and robust, there are some potential downsides; the most common is the cost of sorting. To use a MERGE the datasets normally must go through multiple sorts. Each data set has to be sorted by the common variables before the merge step occurs; and the results dataset often has to be sorted in to its most common access order after the merge. The CPU time can significantly increased because of the sorting procedures; in addition, if the datasets are large there is a need for extended disk space to accommodate the sort. With the introduction of a memory search method – the SAS HASH object, merging with a HASH look-up table method substantially improves the data management process, not only increasing efficiency but also improving code transparency. This paper will illustrate some techniques used to convert programmes using a MERGE statement to programmes using a HASH Object in the match-merge table relationship. Both one-to-one and one-to-many relationships will be covered.

INTRODUCTION

For the purposes of the paper match-merging is the process of combining observations from one or more look-up tables to one main table to form a single observation in the new table according to the common variables specified in the BY statement that are available for all matching tables. There are two types of relationships between a main table and look-up tables. They are as follows:

- ONE-to-ONE relationship: one observation in a main table matches only one observation in their look-up tables.

- ONE-to-MANY relationships: one observation in a main table matches two or more observations in their look-up tables.

This paper will use practical examples to illustrate these two type relationships by converting MERGE statements into hash objects.

IMPLEMENTATION

ONE-to-ONE relationship implies that there is exactly one observation in a main table that joins exactly one observation in a look-up table. This paper will exam two types of common joins between the main table and the look-up table: inner join, left outer join. SAS® code will be provided for converting MERGE statements into HASH objects to describe the inner join and left outer join relationship in the following common practical examples:

- Example1: a main table inner joins a look-up table by common variables
- Example 2: a main table left outer joins a look-up table by common variables
- Example 3: special case when a look-up variable returned from a look-up table also exists in a main table
- Example 4: a main table left outer joins more than one look-up tables

EXAMPLE1: A MAIN TABLE INNER JOINS A LOOK-UP TABLE BY COMMON VARIABLES

Definition of Inner join: the resulting table has matched observations that join both the main table and the look-up table by the common variables.

Requirement:
Create a transaction report with patient demographic information: sex, age and postal code. In this report, each observation in a TRANSACTIONS table joins only one observation from a PATIENTS table by the common variable PID. A TRANSACTIONS table has one-to-one inner join relationship with a PATIENTS table.
The description and the sample data for both TRANSACTIONS and PATIENTS tables can be found in the appendix.

Implementation:

Option1: Inner join with MERGE statement.

SAS® code:

```
proc sort data=data.transactions out=transactions;
  by pid;
run;

proc sort data=data.patients(drop=studyid) out=patients;
  by pid;
run;

data trans_merge_patients;
  merge transactions (in=inTran)
        patients (in=inPat);
  by pid;
  if inTran and inPat;
  age = intck("year", dob, visitdate);
run;

*- sort data by the original access order -*;
proc sort data=trans_merge_patients;
  by pid did visitdate feecode;
run;
```

Inner join with MERGE statement goes though three sorts. First TRANSACTIONS and PATIENTS are sorted on the common variable PID before merging step occurs. After merging, the results table TRANS_MERGE_PATIENTS is sorted to match the original access order of TRANSACTIONS by pid, did, visitdate and feecode.
The statement `"if inTran and inPat;"` in above SAS® code forces the inner join; that is, only observations that exist in both TRANSACTIONS table and PATIENTS table are kept in the results table.

Option2: Inner join with a hash table.

SAS® code:

```
data trans_hash_patients(drop=rc);

  if 0 then set data.patients(drop=studyid);
  declare hash hh_pat(dataset:"data.patients(drop=studyid)");
  rc=hh_pat.defineKey("pid");
  rc=hh_pat.defineData("dob", "sex", "postcode");
  rc=hh_pat.defineDone();

  do until(eof);
    set data.transactions end=eof;
    call missing(dob, sex, postcode);
    rc=hh_pat.find();
    age = intck("year", dob, visitdate);
```

```
    if rc=0 then output;
  end;
  stop;
run;
```

By converting the inner join with a MERGE statement into the inner join with a HASH object, there are two obvious advantages:

1. Inner join with a MERGE statement requires two sorts prior to MERGE and one sorting after. Inner join with a hash table has no sorting required. By converting MERGE statement into a HASH object, the CPU processing time will be significantly reduced. Users will receive huge time efficiency improvements especially in the large data processing.
2. The code for a MERGE statement only tells SAS® to return data from a look-up table, but is not clear about what these data elements are. The code for using a HASH table is more transparent. The HASH object not only explicitly defines the key (PID) that is used to match, but also explicitly defines the data - dob, sex, postcode - that will be returned from a look-up table (data.patients).

EXAMPLE2: A MAIN TABLE LEFT OUTER JOINS A LOOK-UP TABLE BY COMMON VARIABLES

Definition of left outer join: the resulting table has all observations from a main table and the matched observations from a look-up table.

Requirement:
Create a transaction report with all transactions from a TRANSACTION table and will attach patient demographic information - sex, age and postal code if a patient is found is a PATIENTS table. In this report, each observation in a TRANSACTIONS table may or may not match an observation from a PATIENTS table. If matched, there is only one observation from a PATIENTS table retrieved. A TRANSACTIONS table has one-to-one left outer join relationship with a PATIENTS table.

The description and the sample of the data of both TRANSACTIONS and PATIENTS table can be found in the appendix.

Implementation:

Option1: Left outer join with MERGE statement.

SAS® code:

```
proc sort data=data.transactions out=transactions;
  by pid;
run;

proc sort data=data.patients(drop=studyid) out=patients;
  by pid;
run;

data trans_outJoin_merge_patients;
  merge transactions (in=inTran)
        patients (in=inPat);
  by pid;
  if inTran;
  age = intck("year", dob, visitdate);
run;

*- sort data by the original access order -*;
proc sort data=trans_outJoin_merge_patients;
  by pid did visitdate feecode;
run;
```

Left outer join with MERGE statement has similar structure to the inner join with MERGE statement. The difference is in the MERGE step. The left outer join uses statement "if inTran;" which means all observations from the main table TRANSACTIONS need to be returned regardless of whether or not they will be matched in the look-up table. However, the inner join uses the statement "if inTran and inPat;, which enforces that only matched observations from both the main table and the look-up table will be retrieved.

Options2: Left outer join with a HASH table.

SAS® code:

```
data trans_outJoin_hash_patients(drop=rc);

  if 0 then set data.patients(drop=studyid);

  declare hash hh_pat(dataset:"data.patients(drop=studyid)");
  rc=hh_pat.defineKey("pid");
  rc=hh_pat.defineData("dob", "sex", "postcode");
  rc=hh_pat.defineDone();

  do until(eof);
    set data.transactions end=eof;
    call missing(dob, sex, postcode);
    rc=hh_pat.find();
    age = intck("year", dob, visitdate);
    output;
  end;
  stop;
run;
```

Left outer join with a HSAH object has similar structure to the inner join with a HSAH object as well. The coding difference is that the subsetting statement "`if rc=0 then output;`" Is removed in the left outer join codes. It means that all observations get returned regardless of whether or not they will be found in the look-up table.

Again by converting left outer join with MERGE statement into a HASH object, users will receive the same advantages as converting inner join with MERGE statement into a HASH object. The advantages are the significant CPU time reduced and more transparency in code.

EXAMPLE3: SPECIAL CASE WHEN A LOOK-UP VARIABLE RETURNED FROM A LOOK-UP TABLE ALSO EXISTS IN A MAIN TABLE

Requirement:

Create a report with all transactions from a TRANSACTIONS_W_PCODE left outer join PARTIAL_PATIENTS table.
The TRANSACTIONS_W_PCODE table has the similar structure to the TRANSACTIONS table, but it includes a variable postcode for the postal code. PARTIAL_PATIENTS table has the same data layout as PATIENTS table, but it does not include patients with PID less than 200.

In this report, each record in a TRANSACTIONS_W_PCODE table may or may not match a record in a PARTIAL_PATIENTS table. If there is a match, allow the value of postcode from a PARTIAL_PATIENTS table to overwrite the value of postcode in the main table. If there is no match, use the value of postcode from the TRANSACTIONS_W_PCODE table.
The description and the sample of the data of TRANSACTIONS_W_PCODE and PARTIAL_PATIENTS table can be found in the appendix.

Implementation:

Option1: with a MERGE statement.

SAS® code:

```
proc sort data=data.transactions_w_pcode out=transactions_w_pcode;
  by pid;
run;

proc sort data=data.partial_patients out=partial_patients;
  by pid;
run;

data trans_wPcode_merge_patients;
  merge transactions_w_pcode (in=inTran)
        partial_patients (in=inPat);
  by pid;
  if inTran;
run;

data test_merge(keep=pid postcode dob sex);
  set trans_wPcode_merge_patients;
  if pid <200;
run;

proc sort data=test_merge nodupkey;
  by pid;
run;
```

The results of TEST_MERGE table for the patients with the missing PID (`if pid <200`) show that these patients keep their original value of postcode from the TRANSACTIONS_W_PCODE table. Please see the results in the below table.

PID	postcode	dob	sex
30	M4S 3H2		
52	M4Y 1N3		
88	M4J 1S6		
119	M4C 4K3		
142	M4L 2T9		
189	M4T 1G6		

Option2: with a HASH object.

SAS® code:

- hash option with missing value assigned to postcode-;

```
data trans_wPcode_hash_patients(drop=rc);

  if 0 then set data.partial_patients(drop=studyid);

  declare hash hh_pat(dataset:"data.partial_patients(drop=studyid)");
  rc=hh_pat.defineKey("pid");
  rc=hh_pat.defineData("dob", "sex", "postcode");
  rc=hh_pat.defineDone();

  do until(eof);
    set data.transactions_w_pcode end=eof;
    call missing(dob, sex, postcode);
    rc=hh_pat.find();
    output;
```

```
  end;
  stop;
run;

data test_hash(keep=pid postcode dob sex);
  set trans_wPcode_hash_patients;
  if pid <200;
run;

proc sort data=test_hash nodupkey;
  by pid;
run;
```

The results of test_hash for the patients with missing PID (`if pid <200`) show that these patients do not have value assigned to postcode. Please see the results in the below table:

PID	postcode	dob	sex
30			
52			
88			
119			
142			
189			

Why does the result come out differently? Because the statement "`call missing(dob, sex, postcode);`" sets all the variables dob, sex and postcode to missing. Therefore if the key is found in a hash look-up table (`hh_pat.find()=0`), the missing values are overwritten by the values returned from the hash look-up table; however, if the key is not found, all values will remain missing. Since the observations with pid <200 are not found in the hash look-up table PARTIAL_PATIENTS, these observations will have missing value for postcode. To fix the problem, a user needs to rename the postcode variable in the main table. During the data step process, if a patient is not found in the hash look-up table, reassign the original value of postcode to the postcode.

Revised Option2: Fixed codes with a HASH object.

SAS® code:

- hash option with keeping value of postcode in the main table when not found in a lookup table-;

```
data trans_wPcode_hash_patients(drop=rc origPcode);

  if 0 then set data.partial_patients(drop=studyid);

  declare hash hh_pat(dataset:"data.partial_patients(drop=studyid)");
  rc=hh_pat.defineKey("pid");
  rc=hh_pat.defineData("dob", "sex", "postcode");
  rc=hh_pat.defineDone();

  do until(eof);
    set data.transactions_w_pcode (rename=postcode=origPcode)end=eof;
    call missing(dob, sex, postcode);
    rc=hh_pat.find();
    if rc ne 0 then postcode=origPcode;
    output;
  end;
  stop;
run;

data test_hash(keep=pid postcode dob sex);
  set trans_wPcode_hash_patients;
  if pid <200;
```

```
run;

proc sort data=test_hash nodupkey;
  by pid;
run;
```

The following is the results for the observations with patient ID not found in a look-up table using revised option2:

PID	postcode	dob	sex
30	M4S 3H2		
52	M4Y 1N3		
88	M4J 1S6		
119	M4C 4K3		
142	M4L 2T9		
189	M4T 1G6		

With the fixed codes, the results show the same from the results using a MERGE statement.

EXAMPLE 4: A MAIN TABLE LEFT OUTER JOINS MORE THAN ONE LOOK-UP TABLES

Requirement:
Create a transaction report with all patient information and doctor information. In this report, each observation in a TRANSACTIONS table will match an observation in a PATIENTS table and an observation in a DOCTORS table. TRANSACTIONS table has one-to-one left outer join relationship with a PATIENTS and a DOCTORS tables.

The description and the sample of the data of TRANSACTIONS, PATIENTS and DOCTORS table can be found in the appendix.

Implementation:

Option1: Left outer join with MERGE statements in multiple look-up tables.

SAS® code:

```
proc sort data=data.transactions out=transactions;
  by pid;
run;

proc sort data=data.patients(drop=studyid
                             rename=(dob=patBirthDate
                                     sex=patSex
                                     postcode=patPcode)
                             )
  out=patients;
  by pid;
run;

data trans_merge_patients;
  merge transactions (in=inTran)
        patients (in=inPat);
  by pid;
  if inTran and inPat;
```

```
run;

proc sort data=trans_merge_patients;
  by did;
run;

proc sort data=data.doctors (rename=(dob=docBirthDate
                                     sex=docSex
                                     postcode=docPcode)

                                            )
  out=doctors;
  by did;
run;

*- retain statement keeps the order of variables in the output -*;
data trans_merge_patients_doctors;
  retain pid did visitdate feecode patPcode patBirthDate patSex docPcode
docBirthDate docSex;

  merge trans_merge_patients (in=inTrPat)
        doctors (in=inDoc);
  by did;
  if inTrPat;
run;

proc sort data=trans_merge_patients_doctors;
  by pid did visitdate feecode patPcode patBirthDate patSex docPcode docBirthDate
docSex;
run;
```

The sample of results from using MERGE statements is listed in the below:

PID	DID	visitdate	feecode	patPcode	patBirthDate	patSex	docPcode	docBirthDate	docSex
30	3540	03/06/2008	G310	M4S 3H2	24/08/1949	M	M5B1W8	01/01/1945	M
30	3540	03/06/2008	G700	M4S 3H2	24/08/1949	M	M5B1W8	01/01/1945	M
30	11367	08/12/2008	A903	M4S 3H2	24/08/1949	M	L3T4A3	23/05/1933	M
30	11367	08/12/2008	G310	M4S 3H2	24/08/1949	M	L3T4A3	23/05/1933	M
30	11367	08/12/2008	G489	M4S 3H2	24/08/1949	M	L3T4A3	23/05/1933	M
30	13197	28/08/2008	G313	M4S 3H2	24/08/1949	M	M2J1V1	07/07/1942	M
30	13197	08/12/2008	G313	M4S 3H2	24/08/1949	M	M2J1V1	07/07/1942	M
30	16125	03/06/2008	G313	M4S 3H2	24/08/1949	M	M2J1V1	07/07/1939	M
30	17768	04/04/2008	A001	M4S 3H2	24/08/1949	M	M5N1N2	07/07/1930	M
30	17768	15/07/2008	A007	M4S 3H2	24/08/1949	M	M5N1N2	07/07/1930	M
30	17768	26/08/2008	A003	M4S 3H2	24/08/1949	M	M5N1N2	07/07/1930	M
30	17985	09/12/2008	S323	M4S 3H2	24/08/1949	M	L6H6K7	15/03/1952	M
30	18644	19/01/2009	A007	M4S 3H2	24/08/1949	M	M6H3L8	02/01/1949	M
30	18799	28/08/2008	A035	M4S 3H2	24/08/1949	M	L3T4A3	18/05/1947	M
30	18799	28/08/2008	G310	M4S 3H2	24/08/1949	M	L3T4A3	18/05/1947	M
30	18799	28/08/2008	G489	M4S 3H2	24/08/1949	M	L3T4A3	18/05/1947	M
30	18799	09/12/2008	G224	M4S 3H2	24/08/1949	M	L3T4A3	18/05/1947	M
30	18799	09/12/2008	S323	M4S 3H2	24/08/1949	M	L3T4A3	18/05/1947	M
30	18799	10/12/2008	C032	M4S 3H2	24/08/1949	M	L3T4A3	18/05/1947	M

Each observation combines patient information and doctor information into an observation from the TRANSACTION table to form a new table TRANS_MERGE_PATIENTS_DOCTORS.
Merging with multiple look-up tables repeats the same process twice as merging with single look-up table. In this example, TRANSACTIONS table and PATIENTS table require being sorted by PID before merged together. Then

the merged table TRANS_MERGE_PATIENTS and DOCTORS table need to be sorted by DID before next merging step. The sorting procedures and merging procedures require each observation from each table being processed four times: two times in sorting procedures and two times in merging procedures.

Option2: Left outer join with HASH objects in multiple look-up tables

SAS® code:

```
*- retain statement keeps the order of variables in the output -*;
data trans_hash_patients_doctors (drop=rc_pat rc_doc);
  retain pid did visitdate feecode patPcode patBirthDate patSex docPcode
docBirthDate docSex;
  if 0 then do;
    set data.patients(drop=studyid
                      rename=(dob=patBirthDate
                              sex=patSex
                                          postcode=patPcode))
        data.doctors(rename=(dob=docBirthDate
                             sex=docSex
                             postcode=docPcode));
  end;

  declare hash hh_pat(dataset:"data.patients(rename=(dob=patBirthDate
                              sex=patSex
                                          postcode=patPcode))");
  rc_pat=hh_pat.defineKey("pid");
  rc_pat=hh_pat.defineData("patBirthDate", "patSex", "patPcode");
  rc_pat=hh_pat.defineDone();

   declare hash hh_doc(dataset:"data.doctors(rename=(dob=docBirthDate
                              sex=docSex
                              postcode=docPcode))");
  rc_doc=hh_doc.defineKey("did");
  rc_doc=hh_doc.defineData("docBirthDate", "docSex", "docPcode");
  rc_doc=hh_doc.defineDone();


  do until(eof);
    set data.transactions end=eof;
    call missing(patBirthDate, patSex, patPcode, docBirthDate, docSex, docPcode);
    rc_pat=hh_pat.find();
    rc_doc=hh_doc.find();
    output;
  end;
  stop;
run;
```

The following table lists the sample results from using HASH objects:

pid	did	visitdate	feecode	patPcode	patBirthDate	patSex	docPcode	docBirthDate	docSex
30	3540	03/06/2008	G310	M4S 3H2	24/08/1949	M	M5B1W8	01/01/1945	M
30	3540	03/06/2008	G700	M4S 3H2	24/08/1949	M	M5B1W8	01/01/1945	M
30	11367	08/12/2008	A903	M4S 3H2	24/08/1949	M	L3T4A3	23/05/1933	M
30	11367	08/12/2008	G310	M4S 3H2	24/08/1949	M	L3T4A3	23/05/1933	M
30	11367	08/12/2008	G489	M4S 3H2	24/08/1949	M	L3T4A3	23/05/1933	M
30	13197	28/08/2008	G313	M4S 3H2	24/08/1949	M	M2J1V1	07/07/1942	M

30	13197	08/12/2008	G313	M4S 3H2	24/08/1949	M	M2J1V1	07/07/1942	M
30	16125	03/06/2008	G313	M4S 3H2	24/08/1949	M	M2J1V1	07/07/1939	M
30	17768	04/04/2008	A001	M4S 3H2	24/08/1949	M	M5N1N2	07/07/1930	M
30	17768	15/07/2008	A007	M4S 3H2	24/08/1949	M	M5N1N2	07/07/1930	M
30	17768	26/08/2008	A003	M4S 3H2	24/08/1949	M	M5N1N2	07/07/1930	M
30	17985	09/12/2008	S323	M4S 3H2	24/08/1949	M	L6H6K7	15/03/1952	M
30	18644	19/01/2009	A007	M4S 3H2	24/08/1949	M	M6H3L8	02/01/1949	M
30	18799	28/08/2008	A035	M4S 3H2	24/08/1949	M	L3T4A3	18/05/1947	M
30	18799	28/08/2008	G310	M4S 3H2	24/08/1949	M	L3T4A3	18/05/1947	M
30	18799	28/08/2008	G489	M4S 3H2	24/08/1949	M	L3T4A3	18/05/1947	M
30	18799	09/12/2008	G224	M4S 3H2	24/08/1949	M	L3T4A3	18/05/1947	M
30	18799	09/12/2008	S323	M4S 3H2	24/08/1949	M	L3T4A3	18/05/1947	M
30	18799	10/12/2008	C032	M4S 3H2	24/08/1949	M	L3T4A3	18/05/1947	M

By using hash objects, the two look-up tables PATIENTS and DOCTORS table are read into the memory as two hash objects. Each observation in the main table TRANSACTIONS only needs to be accessed once. The whole matching process is completed in one data step. By converting MERGE statements into HASH objects, users will get two benefits that are as follows:

- From the coding perspective, the program using HASH objects in matching multiple look-up tables is more concise and transparent than the program using MERGE statements.
- From the CPU process, the program using HASH objects will be executed at least four times faster than the program using MERGE statements.

ONE-to- MANY relationships: One observation in a main table may match one or many observations in a look-up table based on the common variables.

Example 5: a main table inner joins a look-up table with duplicate observations in the key

Requirement:

The FEESCHEDULE table has duplicate feeCode. For example, feecode A201 has two fees: one charges $1.55 for a service that occurs on and after 2008-04-01, but before 2008-10-01; and the other fee charges $5.5 for any service that occurs on and after 2008-10-01. Please see the data layout of the FEESCHEDULE in the below table:

FeeCode	EffectiveDate	EndDate	ProviderFee
A201	2008-04-01	99999999	1.55
A201	2008-10-01	99999999	5.5
A202	2008-04-01	99999999	18.1
A203	2008-04-01	99999999	12
A203	2008-10-01	99999999	16.54
A203	2009-01-01	99999999	20.68
A204	2008-04-01	99999999	1.55
A205	2007-04-01	99999999	10
A205	2008-04-01	99999999	14
A205	2008-10-01	99999999	23.27
A205	2009-01-01	99999999	30
A206	2008-04-01	99999999	15.51
A207	2008-04-01	99999999	62.04
A208	2008-04-01	99999999	7.76
A303	2008-04-01	99999999	11.45
A304	2008-04-01	99999999	7

A305	2008-04-01	99999999	16.4
A310	2008-04-01	99999999	11
A310	2009-01-01	99999999	28.44

The main table TRANSAMPLE has 60 observations. The data layout of TRANSAMPLE can be found in the appendix. Each observation from TRANSAMPLE table will be matched with one or more observations in the FEESCHEDULE table by the common variable feecode. The report requires assigning the correct fee to each feecode in the TRANSAMPLE table.

Implementation:

Option1: match duplicates using a MERGE statement.

The following SAS® code merges the main table TRANSAMPLE with the above look-up table FEESCHEDULE by the common variable feecode.

SAS® code:

```
data trans_merge_feeS;
  merge tranSample (in=inTran)
        feeSchedule(in=inFeeS)
  ;
  by feecode;
  if inTran and inFees;
run;
```

The following table lists the sample results of TRANS_EMRGE_FEES table:

PID	DID	visitdate	feecode	EffectiveDate	EndDate	ProviderFee
402	7343	07/01/2009	A203	01/04/2008	99999999	12
4707	4371	30/04/2008	A203	01/10/2008	99999999	16.54
5172	19967	25/09/2008	A203	01/01/2009	99999999	20.68
5650	15900	17/12/2008	A203	01/01/2009	99999999	20.68
5690	9218	19/11/2008	A203	01/01/2009	99999999	20.68
5690	9218	09/01/2009	A203	01/01/2009	99999999	20.68
5798	17855	11/03/2009	A203	01/01/2009	99999999	20.68
7401	3631	25/02/2009	A203	01/01/2009	99999999	20.68
8548	5974	14/06/2008	A203	01/01/2009	99999999	20.68
9185	8271	11/02/2009	A203	01/01/2009	99999999	20.68
9691	18246	09/12/2008	A206	01/04/2008	99999999	15.51
9691	18246	16/01/2009	A206	01/04/2008	99999999	15.51
1516	12631	31/07/2008	A310	01/04/2008	99999999	11
7792	12747	23/12/2008	A310	01/01/2009	99999999	28.44
7792	12747	27/01/2009	A310	01/01/2009	99999999	28.44

From the above table, it shows that a user does not retrieve the correct fee from FEESCHEDULE table to the corresponding service code.

For example, feecode A203 has three fees:

- $12 for the service time from the 1st of April in 2008 to the 30th of September in 2008;
- $16.54 for the service time from the 1st of October in 2008 to 31st of December in 2008;
- $20.68 for the service starting from the 1st of January in 2009.

However this merging process does not take visitdate into consideration. The fees for feecode A203 are retrieved in sequence from FEESCHEDULE table: the first matched observation from the main table will be assigned to $12. The second matched observation will get $16.54. The third matched observation will be given to $20.68. Since $20.68 is

the last fee for feecode A203 in the FEESCHEDULE table, all the other matched observations with A203 feecode in the main table will be assigned to $20.68. This process will be applied to the same for all the duplicate feecode such as A201, A203, A205, A310 highlighted in yellow in the FEESCHEDULE table.

For the single feecode such as feecode A206 in the FEESCHEDULE table, since there is only one fee associated with the feecode, the single fee will be retrieved each time when an observation in the main table has a feecode matched with them.

In order to assign the correct fee for the service that a patient receives, a user needs to take an extra step to create a unique key - feecode before performing merging. A new look-up table needs to be created with the unique key from the original look-up table FEESCHEDULE. This process requires creating two arrays FeeAmt and feeEffDate to hold fee - providerFee and the corresponding fee effective date - effectiveDate.

The SAS® code to create a look-up table is as follows:

```
data data.feeCodes (keep=feecode feeAmt1-feeAmt4 effDate1-effDate4);
  set data.feeSchedule(keep=feecode effectiveDate providerFee);
  by feecode ;

  array feeAmt(4) feeAmt1 - feeAmt4;
  array feeEffDate(4)  effDate1 - effDate4;
  format effDate1-effDate4 yymmdd10.;

  retain feeAmt1-feeAmt4;
  retain effDate1 - effDate4;
  retain cnt;

  if first.feecode then do;
    do i = 1 to 4;
      feeAmt(i) = .;
      feeEffDate(i) = .;
    end;
    cnt=0;
  end;

  cnt=cnt + 1;
  feeAmt(cnt) = providerFee;
  feeEffDate(cnt) = EffectiveDate;

  if last.feecode then output;

run;
```

The results for the new look-up table FEECODES are listed in the below table:

FeeCode	feeAmt1	feeAmt2	feeAmt3	feeAmt4	effDate1	effDate2	effDate3	effDate4
A201	1.55	5.5			01/04/2008	01/10/2008		
A202	18.1				01/04/2008			
A203	12	16.54	20.68		01/04/2008	01/10/2008	01/01/2009	
A204	1.55				01/04/2008			
A205	10	14	23.27	30	01/04/2007	01/04/2008	01/10/2008	01/01/2009
A206	15.51				01/04/2008			
A207	62.04				01/04/2008			
A208	7.76				01/04/2008			
A303	11.45				01/04/2008			
A304	7				01/04/2008			
A305	16.4				01/04/2008			
A310	11	28.44			01/04/2008	01/01/2009		

After the new look-up table FEECODES is created, the main table TRANSAMPLE merges with the FEECODES by the unique key feeCode. During the merging process, a patient's visiting date (visitdate) is checked against fee effective dates - feeEffDate in order to retrieve the correct fee.

The following is the SAS® code using a MERGE statement:

```
proc sort data=data.tranSample out=tranSample;
  by feecode;
run;

proc sort data=data.feecodes out=feecodes;
  by feecode;
run;


data tran_merge_fee (keep=pid did visitdate feecode baseFeeDate baseFee);
  *- keep variables in a specific order -*;
  retain pid did visitdate feecode baseFeeDate baseFee;
  merge tranSample (in=inTrs)
        feecodes (in=inFee)
  ;
  array feeAmt(4) feeAmt1 - feeAmt4;
  array feeEffDate(4) effDate1 - effDate4;
  format baseFeeDate yymmdd10.;

  by feecode;
  if inTrs and inFee;
  i = 1;
  do until (feeEffDate(i) = . );

    if visitdate < feeEffDate(i) then leave;
    i=i + 1;
    if i = 5 then leave;
  end;
  baseFee=feeAmt(i-1);
  baseFeeDate=feeEffDate(i-1);
run;
```

Option2: match duplicates using a HASH object.

The SAS® code using a HASH object is as follows:

```
data tran_hash_fee(keep=pid did visitdate feecode baseFeeDate baseFee);

  *- keep variables in a specific order -*;
  retain pid did visitdate feecode baseFeeDate baseFee;

  if 0 then set data.feeschedule(keep=feecode effectiveDate providerFee);
  length r 8;
  declare hash h_fee(dataset:"data.feeschedule(keep=feecode effectiveDate
providerFee)",
                     multidata:"Y");
  rc=h_fee.defineKey("feecode");
  rc=h_fee.defineData("effectiveDate", "providerFee");
  rc=h_fee.defineDone();

  format effectiveDate baseFeeDate yymmdd10.;
```

```
  do until(eof);
    set tranSample  end=eof;
    call missing(effectiveDate, providerFee);

    rc=h_fee.find();


    if rc = 0 then do;
      baseFee=providerFee;
      baseFeeDate=effectiveDate;

        h_fee.has_next(result:r);
      do while (r ne 0);
           rc=h_fee.find_next();
            if visitdate < effectiveDate then leave;
           else do;
             baseFee=providerFee;
            baseFeeDate=effectiveDate;
              h_fee.has_next(result:r);
           end;
      end;
    end;

    output;
  end;
  stop;
run;
```

SAS®9.2 allows creating a hash object with multiple data items for each key by indicating "Y" to the hash component "multidata" in the hash declare statement. The hash object h_fee is loaded with data from original look-up table FEESCHEDULE.

The data step uses the following hash methods to perform the matching process:

- h_fee.find() determines if the key feecode exists in the hash object. If the key is found, h_fee.find() returns 0 to rc.
- h_fee.has_next(result:r) determines whether there is a next item in the current key's multiple data item list. If r ne 0, it implies there is another data item in the data item list.
- H_fee.find_next(): retrieves the data items(effectiveDate and providerFee) from the hash object(h_fee()).

The patient's visitdate is checked against effectiveDate in the do while loop until the appropriate fee is found.

In order to verify the conversion from a MERGE statement into a HASH object, the user needs to sort variables in both data sets with the same variable order before executing PROC COMPARE procedure.

The SAS® codes for PROC COMPARE procedure are as follows:

```
proc sort data=tran_merge_fee;
  by pid did visitdate feecode baseFeeDate baseFee ;
run;
```

```
proc sort data=tran_hash_fee;
  by pid did visitdate feecode  baseFeeDate baseFee;
run;

proc compare base=tran_hash_fee compare=tran_merge_fee;
run;
```

The comparing results are as follows:

```
                                  Observation Summary

                              Observation      Base  Compare

                              First Obs           1        1
                              Last  Obs          60       60

                    Number of Observations in Common: 60.
                    Total Number of Observations Read from WORK.TRAN_HASH_FEE: 60.
                    Total Number of Observations Read from WORK.TRAN_MERGE_FEE: 60.

                    Number of Observations with Some Compared Variables Unequal: 0.
                    Number of Observations with All Compared Variables Equal: 60.
```

NOTE: No unequal values were found. All values compared are exactly equal.

The following is the sample results from either TRAN_MERGE_FEE or TRAN_HASH_FEE for the feecode - A203, A206 and A310. Data shows that each observation from TRANSAMPLE is assigned the correct fee.

pid	did	visitdate	feecode	baseFeeDate	baseFee
4707	4371	30/04/2008	A203	01/04/2008	12
5172	19967	25/09/2008	A203	01/04/2008	12
8548	5974	14/06/2008	A203	01/04/2008	12
5650	15900	17/12/2008	A203	01/10/2008	16.54
5690	9218	19/11/2008	A203	01/10/2008	16.54
402	7343	07/01/2009	A203	01/01/2009	20.68
5690	9218	09/01/2009	A203	01/01/2009	20.68
5798	17855	11/03/2009	A203	01/01/2009	20.68
7401	3631	25/02/2009	A203	01/01/2009	20.68
9185	8271	11/02/2009	A203	01/01/2009	20.68
9691	18246	09/12/2008	A206	01/04/2008	15.51
9691	18246	16/01/2009	A206	01/04/2008	15.51
1516	12631	31/07/2008	A310	01/04/2008	11
7792	12747	23/12/2008	A310	01/04/2008	11
7792	12747	27/01/2009	A310	01/01/2009	28.44

Note: baseFeeDate: effectiveDate;
baseFee: ProviderFee.

CONCLUSION

This paper demonstrates five practical examples by converting MERGE statements into HASH objects.

In a program with MERGE statements, the common variables in BY statement for both tables must be sorted or indexed prior to merging. When merging with multiple tables or a table with duplicate keys, the match-merge process requires several data steps to complete.

On the other hand, a program with HASH objects loads all look-up tables into memory and no sorting or processing of these tables is required. The search is a memory-index search. In addition, the main table can be processed in its natural order; since the main table is often a very large table the ability to process this table only one time in its natural order will usually result in considerable savings in time and CPU usage. In the end, the whole match-merge process can be completed in one data step.

By converting a program with MERGE statements into a program with HASH objects, a user will benefit the following advantages:

- Less coding. A program with HASH objects has more concise and transparent SAS® code.
- More efficiency. A program with HASH objects is executed significantly faster.
- Less memory and less amounts disk space required.

REFERENCES

Eberhardt, Peter "The SAS Hash Object: It's Time to .find() Your Way Around". *Proceedings of annual Pharmaceutical Industry SAS® Users Group Conference*, 2010

Liu, Ying "SAS® Hash Objects: An Efficient Table Look-Up in the Decision Tree". *Proceedings of sixteenth Annual Southeast SAS® Users Group Conference*, 2008

ACKNOWLEDGMENTS

First and foremost, I would like to thank Mr. Peter Eberhardt for his valuable advice. He has greatly inspired me to work on writing SAS papers with years of working experience using SAS. His support has motivated me and has contributed tremendously to this paper.

CONTACT INFORMATION

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

Ying Liu
12 Lisa Cres.
Richmond Hill, ON
L4B 3J4 Canada
Phone: (416) 897-7551
E-mail: y22liu@yahoo.com

APPENDIX

The following tables are used in this paper:

Table: Transactions

Table attributes:

- PID: patient's identification #
- DID: doctor's identification #
- Visitdate: the date when a patient visits a doctor
- Feecode: the code that a doctor uses to charge the service provided to a patient

Key: a combination of PID, DID and visitdate.

Sample data of transactions table:

PID	DID	visitdate	feecode
903	22006	01/04/2008	A034
3318	3646	01/04/2008	A003
3318	3646	01/04/2008	E430
3318	3646	01/04/2008	G010
3318	3646	01/04/2008	G310
3318	3646	01/04/2008	G313
3318	3646	01/04/2008	G489
3758	2662	01/04/2008	P007
3773	5070	01/04/2008	W003
4384	8975	01/04/2008	A007
5690	18789	01/04/2008	A244
5690	18789	01/04/2008	Z296
5737	21005	01/04/2008	A025
7037	20743	01/04/2008	C124
7574	13267	01/04/2008	X091
7574	13267	01/04/2008	X091
8363	9314	01/04/2008	G441
8363	9314	01/04/2008	G442
8363	9314	01/04/2008	G526

Table: Doctors

Table attributes:

- DID: doctor's identification #
- Postcode: post code of the clinic where a doctor practises
- Dob: doctor's birth date
- Sex: doctor's gender (M-Male; F-Female)

Key: DID.

Sample data of doctors table:

DID	postcode	dob	sex
1088	61761	03/03/1947	M
17036	75093	05/11/1969	M
2	K0A1A0	10/02/1942	M
338	K0A1A0	20/12/1970	M
717	K0A1A0	01/02/1963	M
4126	K0A1A0	11/04/1952	F
4816	K0A1A0	29/11/1969	F
6821	K0A1A0	08/03/1966	F
7927	K0A1A0	11/04/1951	M
10012	K0A1A0	29/01/1952	M
10388	K0A1A0	15/08/1950	M
13847	K0A1A0	17/07/1967	F
15026	K0A1A0	28/01/1948	M
15209	K0A1A0	03/04/1956	F
20843	K0A1A0	20/03/1956	M
7104	K0A1B0	01/10/1956	F
69	K0A1L0	02/11/1972	F
615	K0A1L0	21/06/1980	F
1531	K0A1L0	05/10/1969	F

Table: Patients

Table attributes:

- PID: patient's identification #
- Postcode: post code of a patient's residence
- Dob: patient's birth date
- Sex: patient's gender (M-Male; F-Female)

Key: PID

Sample data of patients table:

PID	postcode	dob	sex
148975	K0K 1C0	07/06/1953	M
193575	L0N 1S4	12/09/1927	M
275667	N0G 2W0	08/04/1970	F
278728	K0A 1T0	21/04/1966	F
296583	P0T 2P0	25/10/1997	F
305558	P0H 1Z0	16/03/1987	F
405856	L0M 1S0	20/03/1996	F
408678	L0N 1S1	26/07/1964	F
424492	K0L 1C0	16/11/1938	F
495503	K0J 2J0	26/12/2007	F
524858	K0K 1C0	29/03/1954	M
698672	K0L 1C0	10/09/1962	F
870433	N0H 2L0	06/01/1956	M
898724	N0R 1G0	15/03/1946	M
1052342	L0S 1N0	22/07/1975	F
1078899	L0M 1A0	25/05/1927	F
1121035	K0M 1S0	18/07/1939	F
1156741	K0A 1L0	16/03/1960	M
1185735	K0A 1R0	21/01/1955	F

Table: transactions_w_pcode

Table attributes:

- PID: patient's identification #
- Postcode: post code of a patient's residence
- DID: doctor's identification #
- Visitdate: the date when a patient visits a doctor
- Feecode: the code that a doctor uses to charge the service provided to a patient

Key: a combination of PID, DID and visitdate.

Sample data of transactions table:

PID	postcode	DID	visitdate	feecode
30	M4S 3H2	3540	03/06/2008	G310
30	M4S 3H2	3540	03/06/2008	G700
30	M4S 3H2	11367	08/12/2008	A903
30	M4S 3H2	11367	08/12/2008	G310
30	M4S 3H2	11367	08/12/2008	G489
30	M4S 3H2	13197	28/08/2008	G313
30	M4S 3H2	13197	08/12/2008	G313
30	M4S 3H2	16125	03/06/2008	G313
30	M4S 3H2	17768	04/04/2008	A001
30	M4S 3H2	17768	15/07/2008	A007
30	M4S 3H2	17768	26/08/2008	A003
30	M4S 3H2	17985	09/12/2008	S323
30	M4S 3H2	18644	19/01/2009	A007
30	M4S 3H2	18799	28/08/2008	A035
30	M4S 3H2	18799	28/08/2008	G310
30	M4S 3H2	18799	28/08/2008	G489
30	M4S 3H2	18799	09/12/2008	G224
30	M4S 3H2	18799	09/12/2008	S323
30	M4S 3H2	18799	10/12/2008	C032

Table: FeeSchedule

Table attributes:

- Feecode: the code that a doctor uses to charge the service provided to a patient
- EffectiveDate: the date when the providerFee is effective
- EndDate: default to 99999999
- ProviderFee: the amount of money charged on the corresponding service code

Key: a combination of FeeCode and EffectiveDate.

Data in FeeSchedule:

FeeCode	EffectiveDate	EndDate	ProviderFee
A201	2008-04-01	99999999	1.55
A201	2008-10-01	99999999	5.5
A202	2008-04-01	99999999	18.1
A203	2008-04-01	99999999	12
A203	2008-10-01	99999999	16.54
A203	2009-01-01	99999999	20.68
A204	2008-04-01	99999999	1.55
A205	2007-04-01	99999999	10
A205	2008-04-01	99999999	14
A205	2008-10-01	99999999	23.27
A205	2009-01-01	99999999	30
A206	2008-04-01	99999999	15.51
A207	2008-04-01	99999999	62.04
A208	2008-04-01	99999999	7.76
A303	2008-04-01	99999999	11.45
A304	2008-04-01	99999999	7
A305	2008-04-01	99999999	16.4
A310	2008-04-01	99999999	11
A310	2009-01-01	99999999	28.44

Table: tranSample

Table attributes:

- PID: patient's identification #
- DID: doctor's identification #
- Visitdate: the date when a patient visits a doctor
- Feecode: the code that a doctor uses to charge the service provided to a patient

Key: a combination of PID, DID and visitdate.

All data in tranSample:

PID	DID	visitdate	feecode
402	7343	16/04/2008	A204
402	7343	07/01/2009	A203
1516	12631	31/07/2008	A310
1531	2256	22/04/2008	A204
1531	2256	22/07/2008	A204
1531	2256	20/01/2009	A204
1797	8636	10/02/2009	A205
3029	3547	04/11/2008	A204
3513	244	25/11/2008	A205
3957	22107	27/03/2009	A205
4448	648	11/09/2008	A205
4707	4371	30/04/2008	A203
4707	4371	05/06/2008	A204
4707	4371	18/08/2008	A204
4707	4371	07/10/2008	A204
4707	4371	26/11/2008	A204
4707	7158	13/12/2008	A205
5172	19967	28/08/2008	A205
5172	19967	25/09/2008	A203
5172	19967	19/12/2008	A204
5172	19967	03/03/2009	A204
5650	15900	12/11/2008	A205
5650	15900	17/12/2008	A203
5650	20089	27/02/2009	A205
5690	9218	23/09/2008	A205
5690	9218	19/11/2008	A203
5690	9218	09/01/2009	A203
5798	17855	29/01/2009	A205
5798	17855	15/02/2009	A204
5798	17855	17/02/2009	A204
5798	17855	24/02/2009	A204
5798	17855	11/03/2009	A203
6263	13604	13/05/2008	A205
6671	19470	18/11/2008	A205
6846	20089	14/11/2008	A205
6846	20089	24/03/2009	A204
6846	22107	31/03/2009	A204

7250	15718	20/03/2009	A205
7379	1880	25/02/2009	A204
7379	8113	22/02/2009	A204
7401	3631	25/02/2009	A203
7792	12747	23/12/2008	A310
7792	12747	27/01/2009	A310
7959	8978	02/07/2008	A205
8330	13604	01/12/2008	A205
8330	13604	22/01/2009	A205
8548	5974	14/06/2008	A203
8548	8978	09/06/2008	A205
8859	9218	21/08/2008	A205
9168	18377	28/10/2008	A205
9168	18377	24/11/2008	A204
9168	18377	16/12/2008	A204
9185	8271	06/06/2008	A204
9185	8271	29/09/2008	A204
9185	8271	11/02/2009	A203
9691	18246	21/10/2008	A205
9691	18246	09/12/2008	A206
9691	18246	16/01/2009	A206
11241	2256	05/05/2008	A204
11455	20900	25/09/2008	A204