Paper 101-2010

Using PROC SQL to Summarize and Transpose Data

Kevin Chung, Fannie Mae, Washington DC

ABSTRACT

Do you need to create a SAS data set using SUMMARY and TRANSPOSE procedures? Do you want to replace the SUMMARY and TRANSPOSE procedures with a SQL script which is easy to maintain and extend? If your answer is yes, then the PROC SQL tips provided in this paper would be very helpful to improve your data processing routine. This paper demonstrates how to use one single SQL CREATE TABLE statement to perform the count, summarize, and transpose data. The purpose of this paper is to provide an alternative way to manipulate data more efficiently than using traditional methods such as data step, SUMMARY and TRANSPOSE procedures. By using PROC SQL, you can eliminate the use of data steps, proc SUMMARY and TRANSPOSE steps. Since there is a pattern when you use SQL method, you can convert the query to a macro easily. The intended audience is the intermediate SAS users with good knowledge of Base SAS.

INTRODUCTION

The major strategy of the SQL procedure demonstrated here is to use SAS SUM function with logical expression to determine which row to select. Suppose we need to check if the variable TYPE is 3. The expression `1-abs(sign(TYPE-3))` used in reference [1] allows you to evaluate the condition as TRUE or FALSE. The nice thing of the SAS logical expression is to simply use `(TYPE=3)` to evaluate the TRUE or FALSE; therefore, it's very easy to be plugged into the SQL statement. This functionality presents a convenient way to summarize data into specific buckets. Most of the data as illustrated by the tables below can be created by REPORT procedure with OUT= option. If you need to create a report with summary and transposition of the variables, the TABULATE and REPORT procedures might be a convenient approach to simplify steps and improve efficiency. This paper is very useful for those SAS users who need to create data sets using data steps as well as the SUMMARY and TRANSPOSE procedures. Four examples are given below for different scenarios and each example has more than one way to reach the same results.

EXAMPLE 1:

How to create a data set with the loan count and loan amount like the following output? You can use TABULATE procedure to generate the following report easily. But we need to create an intermediate SAS data set which will be used by subsequent steps.

aqsn_dt	DC_Count	DC_Sum	MD_Count	MD_Sum	VA_Count	VA_Sum
201001	4	890000	2	248000	1	118000
201002	1	153000	5	1078000	1	232000
201003	2	425000	1	86000	3	708000

Create the input data set.

```
data loans;
  input aqsn_dt :mmddyy10. state :$2. product :$1. upb @@;
  format aqsn_dt yymmn6.;
cards;
01/01/2010 MD A 124000  02/01/2010 DC A 153000  03/01/2010 VA B 159000
01/01/2010 DC B 182000  02/01/2010 MD A  92000  03/01/2010 VA A 133000
01/01/2010 VA A 118000  02/01/2010 MD B 160000  03/01/2010 DC A 203000
01/01/2010 DC A 219000  02/01/2010 MD A 255000  03/01/2010 MD B  86000
01/01/2010 DC B 227000  02/01/2010 MD A 319000  03/01/2010 VA A 416000
01/01/2010 MD A 124000  02/01/2010 VA A 232000  03/01/2010 DC A 222000
01/01/2010 DC B 262000  02/01/2010 MD B 252000
;
```

Let’s start with the traditional method using SUMMARY procedure first.

```
/* Traditional method */
proc summary data=loans nway;
  class aqsn_dt state;
  var upb;
  output out=out1(drop=_:)
         N=Count sum=UPB;
  format aqsn_dt yymmn6.;
run;
```

```
Obs   aqsn_dt  state  Count     UPB

 1    201001    DC      4      890000
 2    201001    MD      2      248000
 3    201001    VA      1      118000
 4    201002    DC      1      153000
 5    201002    MD      5     1078000
 6    201002    VA      1      232000
 7    201003    DC      2      425000
 8    201003    MD      1       86000
 9    201003    VA      3      708000
```

The second step is to apply the TRANSPOSE procedure on the variable Count and UPB (UnPaid Balance). Since two variables were transposed, two rows were generated with each aqsn_dt/state combination. The added variable _NAME_ can be used to identify which original variable in the input data set the value originates. An intermediate data step is required to combine the state and _NAME_ as one new variable IDVAR, which will be used by the second TRANSPOSE procedure as the variable name.

```
proc transpose data=out1
     out=out2;
  by aqsn_dt state;
  var Count upb;
run;


data out2;
  set out2;
  idvar=state||'_'||
        _name_;
  drop state _name_;
run;
```

```
Obs  aqsn_dt  state  _NAME_    COL1

  1  201001    DC    Count         4
  2  201001    DC    UPB      890000
  3  201001    MD    Count         2
  4  201001    MD    UPB      248000
  5  201001    VA    Count         1
  6  201001    VA    UPB      118000
  7  201002    DC    Count         1
  8  201002    DC    UPB      153000
  9  201002    MD    Count         5
 10  201002    MD    UPB     1078000
 11  201002    VA    Count         1
 12  201002    VA    UPB      232000
 13  201003    DC    Count         2
 14  201003    DC    UPB      425000
 15  201003    MD    Count         1
 16  201003    MD    UPB       86000
 17  201003    VA    Count         3
 18  201003    VA    UPB      708000
```

```
Obs  aqsn_dt     COL1    idvar

  1  201001         4  DC_Count
  2  201001    890000  DC_UPB
  3  201001         2  MD_Count
  4  201001    248000  MD_UPB
  5  201001         1  VA_Count
  6  201001    118000  VA_UPB
  7  201002         1  DC_Count
  8  201002    153000  DC_UPB
  9  201002         5  MD_Count
 10  201002   1078000  MD_UPB
 11  201002         1  VA_Count
 12  201002    232000  VA_UPB
 13  201003         2  DC_Count
 14  201003    425000  DC_UPB
 15  201003         1  MD_Count
 16  201003     86000  MD_UPB
 17  201003         3  VA_Count
 18  201003    708000  VA_UPB
```

A second TRANSPOSE procedure was applied with the IDVAR value as the new variable name for each corresponding COL1 and the output OUT3 is the data set we want.

```
proc transpose data=out2
   out=out3(drop=_name_);
  by aqsn_dt;
  id idvar;
  var col1;
run;
```

```
Obs   aqsn_dt   DC_Count   DC_UPB   MD_Count    MD_UPB   VA_Count    VA_UPB

 1    201001        4      890000      2        248000       1       118000
 2    201002        1      153000      5       1078000       1       232000
 3    201003        2      425000      1         86000       3       708000
```

The alternative way is to use a data step with two dimentional array to perform the summary and transposition operations and create the identical data set. The example below is a hard-coded version because it’s easy for illustration. Since the BY-group processing will be used in the data step, the input data set LOANS must be sorted by AQSN_DT and STATE. Let’s go over the process for AQSN_DT=201001 and see how the summary and transposition operations are performed in the data step. At ❷, the field r represents the *n*th state and it’s reset to 0 at the first obs of the primary BY-variable AQSN_DT. Then r increased by 1 at the first obs of the secondary By-variable STATE at ❸. The summary operation is performed at ❹ and ❺. A 3x2 array A is declared at ❶ to hold the cumulative values from ❹ and ❺. Step ❻ and ❼ transpose a two dimentional array to a one dimentional array when the data step reaches the last obs of the 201001 group. The value 3 in DO loop is the number of distinct STATE. The entire process repeats for AQSN_DT=201002 and 201003.

```
/* Data step method */
proc sort data=loans; by aqsn_dt state; run;
data one;
  set loans;
  by aqsn_dt state;
  array out(*) DC_Count DC_UPB MD_Count
               MD_UPB VA_Count VA_UPB;
  array a(3,2); ❶

  if first.aqsn_dt then do;
    r=0; ❷
    call missing(of a(*));
  end;
  if first.state then r+1; ❸
  a(r,1)+1; ❹
  a(r,2)+UPB; ❺

  if last.aqsn_dt then do;
    do i=1 to 3;
      out(2*i-1)=a(i,1); ❻
      out(2*i)=a(i,2); ❼
    end;
    output;
  end;
  keep aqsn_dt DC_Count DC_UPB MD_Count
       MD_UPB VA_Count VA_UPB;
run;
```

Partial output of LOANS data set

```
Obs  aqsn_dt  state    upb

 1   201001    DC    182000
 2   201001    DC    219000
 3   201001    DC    227000
 4   201001    DC    262000
 5   201001    MD    124000
 6   201001    MD    124000
 7   201001    VA    118000
```

Values stored in array A at ❹ and ❺ for 201001

(1,1) 4	(1,2) 890000
(2,1) 2	(2,2) 248000
(3,1) 1	(3,2) 118000

The step ❻ and ❼ transpose a two dimentional array to a one dimentional array.

DC_Count	DC_UPB	MD_Count	MD_UPB	VA_Count	VA_UPB
(1,1) ➔ (1) 4	(1,2) ➔ (2) 890000	(2,1) ➔ (3) 2	(2,2) ➔ (4) 248000	(3,1) ➔ (5) 1	(3,2) ➔ (6) 118000

The above data step method might be difficult to understand for SAS beginners. Let’s take a look at the following SQL method and see how easy the summary and transposition operations performed in a single SQL statement. The SQL procedure does not require the input data set to be sorted. How does this SQL statement work? Suppose the first obs is read with AQSN_DT=200901. The logical expression, state=’MD’ is evaluated as TRUE and returns 1 only if the obs contains the value ‘MD’ in variable STATE. So, 1 contributes to MD_Count and UPB is added to MD_Sum at ❷. Everything else gets 0, i.e., STATE=’DC’ and STATE=’VA’ at ❶ and ❸ are evaluated as FALSE and returns 0. The next obs is read with AQSN_DT=201002 and STATE=’DC’, DC_Count is increased by 1 and UPB is added to DC_Sum at ❶. The process keeps adding 1 to XX_Count and UPB to XX_Sum if the logical expression is evaluated as TRUE for STATE=’XX’. Since GROUP BY clause is specified, the data set is created with six fields for each month. You might ask one question, since we always use select count(*) … SQL statement to count the number of obs that meets a particular condition, why not COUNT(state=’DC’)? The answer is the SUM function is used to add up each TRUE (1) value from those observations that meet the condition specified in the argument. If COUNT function is used, the XX_Count is the same for each month because COUNT function adds up 1 for each obs even a FALSE (0) value is evaluated.

```
/* SQL method */
proc sql;
  create table state_sum as
  select aqsn_dt format=yymmn6.,
         sum((state='DC')) as DC_Count, sum((state='DC')*upb) as DC_Sum, ❶
         sum((state='MD')) as MD_Count, sum((state='MD')*upb) as MD_Sum, ❷
         sum((state='VA')) as VA_Count, sum((state='VA')*upb) as VA_Sum  ❸
  from loans
  group by aqsn_dt;
quit;
```

How can I convert the SQL script to a macro?

```
%macro m1;
  proc sql noprint;
    select distinct state, count(distinct state)
    into :list separated by ' ',
         :n
    from loans;
    %let n=&n;
    %put list=&list n=&n;

    create table state_sum as
    select aqsn_dt format=yymmn6.
      %do i=1 %to &n;
        %let st=%scan(&list,&i);
        ,sum((state="&st")) as &st._Count, sum((state="&st")*upb) as &st._Sum
      %end;
    from loans
    group by aqsn_dt;
  quit;
%mend m1;
option mprint;
%m1
```

EXAMPLE 2:

Same input data as Example 1. How can I count the number of products under each state. This example demonstrates the usage of more than one logical expression. The field DC_Prod_A stands for the count of observations with STATE=’DC ’ and PRODUCT=’A ’.

	DC_Prod_A	DC_Prod_B	MD_Prod_A	MD_Prod_B	VA_Prod_A	VA_Prod_B
201001	1	3	2	0	1	0
201002	1	0	3	2	1	0
201003	2	0	0	1	2	1

```
/* Traditional method */
proc summary data=loans nway completetypes;
  class aqsn_dt state product;
  var upb;
  output out=out1(drop=_:) N=Count;
  format aqsn_dt yymmn6.;
run;

proc transpose data=out1 out=out2;
  by aqsn_dt state product;
  var Count;
run;

data out2;
  set out2;
  idvar=state||'_Prod_'||product;
  drop state product _name_;
run;

proc transpose data=out2 out=out3(drop=_:);
  by aqsn_dt;
  var col1;
  id idvar;
run;
```

```
/* SQL method */
proc sql;
  create table state_count as
  select aqsn_dt format=yymmn6.,
     sum((state='DC')*(product='A')) as DC_Prod_A, sum((state='DC')*(product='B')) as
DC_Prod_B,
     sum((state='MD')*(product='A')) as MD_Prod_A, sum((state='MD')*(product='B')) as
MD_Prod_B,
     sum((state='VA')*(product='A')) as VA_Prod_A, sum((state='VA')*(product='B')) as
VA_Prod_B
  from loans
  group by aqsn_dt;
quit;
```

How to convert the above statement to a macro?

```
%macro m2;
  proc sql noprint;
    select distinct state, count(distinct state)
    into :state_list separated by ' ',
         :n1
    from loans;

    select distinct product, count(distinct product)
    into :prdct_list separated by ' ',
         :n2
    from loans;
    %let n1=&n1;
    %let n2=&n2;

    create table state_sum as
    select aqsn_dt format=yymmn6.
      %do i=1 %to &n1;
        %let st=%scan(&state_list,&i);
        %do j=1 %to &n2;
          %let p=%scan(&prdct_list,&j);
          ,sum((state="&st")*(product="&p")*1) as &st._Prod_&p
        %end;
      %end;
    from loans
    group by aqsn_dt;
  quit;
%mend m2;
option mprint;
%m2
```

EXAMPLE 3:

A monthly report is created based on the sales amount and broken out by region. The report contains the total sales amount for current month, previous month, quarter to date (QTD), and year to date (YTD). Assume the program is run at the first day of the month. The hard-coded version below is used to illustrate the process.

Region	Curr	Prev	QTD	YTD
01	19200	29800	19200	119600
02	32200	31400	32200	107200
03	21500	41400	21500	109900

Create the input data set.

The input data contains the sales data between January 2010 and April 2010.

```
data sales;
  dt0=mdy(1,1,2010);
  do id=101 to 300;
    date=dt0+ceil(ranuni(101)*120);
    n=ceil(ranuni(101)*3);
    if mod(n,3)=1 then region='01';
    else if mod(n,3)=2 then region='02';
    else region='03';
    amount=ceil(ranuni(101)*30)*100;;
    output;
  end;
  format date mmddyy10.;
  keep id date region amount;
run;


/* Traditional method */
data one;
  set sales;
  grp=4;
  output;    /* YTD */

  if (month(date)=4) then do;
    grp=1;
    output;  /* Current month */
  end;
  else if (month(date)=3) then do;
    grp=2;
    output;  /* Previous month */
  end;

  if (qtr(date)=2) then do;
    grp=3;
    output;  /* QTD */
  end;
run;

proc summary data=one nway;
  class region grp;
  var amount;
  output out=out1(drop=_:) sum=;
run;

proc transpose data=out1 out=out2(drop=_name_ rename=(_1=Curr _2=Prev _3=QTD _4=YTD));
  by region;
  var amount;
  id grp;
run;
proc print data=out2; run;


/* SQL method */
proc sql;
  create table monthly_rpt as
  select region,
         sum((month(date)=4)*amount) as Curr,
         sum((month(date)=3)*amount) as Prev,
         sum((qtr(date)=2)*amount) as QTD,
         sum(amount) as YTD
  from sales
  group by region;
quit;
```

The following macro variables can be derived and used for all cases including the cross year case (current month is January and previous month is December). For testing, the function TODAY() at ❶ can be replaced with mdy(5,1,2010).

```
/* Assume the program is run at 1st day of the month */
%let today=%sysfunc(today()); ❶
%let curr_month=%sysfunc(intnx(month,&today,-1),yymmn6.);
%let prev_month=%sysfunc(intnx(month,&today,-2),yymmn6.);
%let qtr=%sysfunc(ceil(%sysfunc(month(%sysfunc(intnx(month,&today,-1))))/3));

proc sql;
  select region,
         sum((put(date,yymmn6.)="&curr_month")*amount) as Curr,
         sum((put(date,yymmn6.)="&prev_month")*amount) as Prev,
         sum((qtr(date)=&qtr)*amount) as QTD,
         sum(amount) as YTD
  from sales
  group by region;
quit;
```

EXAMPLE 4:

How can I calculate the weighted average score with the credit as the weight. The purpose of this example is to demonstrate the logical expression with missing checking. It's nothing to do with the TRANSPOSE procedure. Let's create the test data first.

```
data scores;
  input ID @;
  do i=1 to 4;
    input score credit @@;
    output;
  end;
  drop i;
cards;
101 90 4 100 2 .  3 80 2
102 80 4  90 2 70 3 90 2
103 85 4  80 2 80 3 .  2
;
```

```
 Obs     ID     score     credit

   1     101      90         4
   2     101     100         2
   3     101       .         3
   4     101      80         2
   5     102      80         4
   6     102      90         2
   7     102      70         3
   8     102      90         2
   9     103      85         4
  10     103      80         2
  11     103      80         3
  12     103       .         2
```

The values differ between _FREQ_ and Count for ID 101 and 103. This is because the missing analysis variable, SCORE in this exampe, is ignored by SUMMARY procedure.

```
/* SUMMARY procedure */
proc summary data=scores nway;
  class ID;
  var score;
  weight credit;
  output out=wgt_score(drop=_type_)
         N=Count mean=;
run;
```

```
Obs   ID   _FREQ_   Count    score

 1   101     4        3     90.0000
 2   102     4        4     80.9091
 3   103     4        3     82.2222
```

In order to get the same results as output wgt_score, the logical expression `score ne .` has to be applied to exclude those observations with missing SCORE.

```
/* SQL method */
proc sql;
  create table wgt_score2 as
  select ID, sum(score ne .) as Count, sum(score*credit)/sum((score ne .)*credit) as
score
  from scores
  group by ID;
quit;
```

CONCLUSION

Hope you have enjoyed the journey to the world of the SQL tip that deals with the summary and transposition. Without understanding this tip, you still can use traditional ways to manipulate the data. Although data manipulation can be a pain, using the SQL tip appropriately will make your life easier!

REFERENCES

[1] Optimizing Transact-SQL : Advanced Programming Techniques
by David Rozenshtein, Anatoly Abramovich, and Eugene Birger (October 1997)

[2] SAS OnlineDoc® 9.1.3, SAS Institute Inc. Cary, NC.
http://support.sas.com/onlinedoc/913/docMainpage.jsp

ACKNOWLEDGMENTS

I would like to thank my colleague Khaled Merhebi for his review and helpful comments on this paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Feel free to contact the author at:

Kevin Chung
Fannie Mae
4000 Wisconsin Ave., NW
Mail Stop: 2H-4S/07
Washington, DC 20016
Work Phone: 202-752-1568
E-mail: kevin_chung@fanniemae.com
kchung01@hotmail.com

You can download all source codes and presentation slides from www.kevin-chung.com/SGF2010