PharmaSUG 2015 – Paper QT22

Creating output datasets using SQL (Structured Query Language) only

Andrii Stakhniv, Experis Clinical, Ukraine

ABSTRACT

PROC SQL is one of the most powerful procedures in SAS. With this tool we can easily manipulate data and create a large number of outputs.

I will illustrate how we can create final datasets based on three common types of safety outputs using SQL (Structured Query Language) only:

- Outputs with a fixed structure (like Disposition outputs);
- Outputs with descriptive statistics (like Demographics or Laboratory outputs);
- Outputs with a flexible 2-level hierarchy structure (like Adverse Events outputs).

The approaches and tricks presented here can be utilized in everyday work as they are easy to implement and understand.

Additionally, this information can be a helpful resource for those who are new to statistical programming analysis.

INTRODUCTION

SAS is widely used in the pharmaceutical industry as the main software tool for clinical trial analyses. At the core of SAS is a specific programing language which contains four key elements:

- Data Steps
- Procedures
- Macros
- ODS (output delivery system)

With these key elements, we can easily manipulate data, perform different types of analysis, and produce a large variety of reports. One of the most powerful procedures in SAS is PROC SQL; SQL is very flexible and therefore is popular among different developers.

The purpose of this paper is to demonstrate the facilities of SQL and also provide basic knowledge about the clinical trial output creation process by providing step-by-step instructions followed with examples.

GENERATION INPUT DATA

According to the Clinical Data Interchange Standards Consortium (CDISC) standards, Analysis Data Model (ADaM) datasets are used for generation of statistical outputs for a regulatory submission. They are built on Study Data Tabulation Model (SDTM) datasets which contain data collected during the study and organized by clinical domain.

For illustrating the statistical output creation process, we need to have two ADaM datasets. The first one is ASL an Analysis Subject Level (ASL) dataset and the second one is some analysis dataset (ANL).

The ASL dataset has a one-record-per-subject structure, where the USUBJID variable is a unique identifier for each record. Here is usually stored demographic data, treatment group and population information, baseline characteristics and any other subject or study specific data.

An ANL dataset can have different structures depending on the clinical domain but here we will use one where each subject can have several records and each record is classified to both a main category and a sub category (example, Adverse Events dataset where we have System Organ Class and Preferred Term classification).

For our purpose, we assume that we have 30 subjects (patients) in our study and 3 trial treatment arms. To be sure that all applied methods are not data dependent, we set up these parameters at the very beginning of our program.

```
%let pt_num = 30;    /* Number of patients         */
%let tr_num = 3;     /* Number of treatment groups */
%let asl    = _asl;  /* Subject Level dataset name */
%let anl    = _anl;  /* Analysis dataset name      */
```

Everything is ready now to generate our dummy ASL and ANL datasets. To have a cleaner presentation of them, we will simplify the list of variables.

```
** Generation of ASL (Analysis Subject Level) dataset **;
data &asl(drop = i r);

  do i = 1 to &pt_num by 1;
   ** Random variables **;
   r = rand("Uniform");  /* [0,  1] */
   k = floor(10*r);      /* [0, 10] */

   ** Create subject (patient) id number **;
   USUBJID = 1000 + i;

   ** Setup treatment variables numeric and character **;
   TRTN = round(1 + (&tr_num - 1) * r);
   TRT  = "Treatment - "||compress(put(TRTN, best.));

   ** Setup population / discontinue / completers flags **;
   POPFL  = ifc(k < 9, "Y", "");
   COMPFL = ifc(k < 7, "Y", "");
   DISCFL = ifc(k = 7, "Y", "");

   ** set up 2 numeric & their char variables **;
   ** Example, AGE & AGECAT or some BL value) **;
   NUM_VAR1 = round(20 + (50 - 20) * r);        /*[20,50]*/
   CHR_VAR1 = ifc(NUM_VAR1 <= 35, "<= 35", "> 35");

   NUM_VAR2 = round(-5 + (5 +   5) * r, 0.001); /*[-5,5 ]*/
   CHR_VAR2 = ifc(NUM_VAR2 <= 0, "Negative", "Positive");

   output;
  end;
run;


** Generation of ANL (Analysis) dataset **;
data &anl(drop = i r);

  do i = 1 to 25 by 1;
   ** Random variables **;
    r = rand("Uniform");  /* [0,1] */

    USUBJID = 1000 + round(1 + (&pt_num - 1) * r); /*[1001, 1000 + &pt_num]*/

    MCAT    = "Main Category - "||
              compress(put(round(1 + (5 - 1) * rand("Uniform")),best.));
    SCAT    = "Sub Category - "||
              compress(put(round(1 + (5 - 1) * rand("Uniform")),best.));

    output;
  end;
run;
```

DATA PROCESSING

Creation of any statistical output always starts with a review of the specifications. We need to know which template is required and which data will be needed. Earlier, we generated input datasets, so now we can perform data processing. This step is general for all clinical outputs and starts with selecting an appropriate population. This population can be “Safety”, “Intent to treat”, “All patients” or any specific/modified population. To generalize, we assume that the POPFL variable in the ASL dataset is responsible for required population. The ANL dataset contains all available data, so we need to subset it on subjects from the specified population. In practice, we usually need to create additional filtering for the ANL.

```
proc sql noprint;
  ** Subset ASL dataset on population **;
  create table asl_p as
  select a.*
  from &asl as a
  where a.POPFL = "Y";

  ** Subset ANL dataset **;
  create table anl_p as
  select a.*, d.TRTN
  from
    asl_p as d
    join &anl as a on a.USUBJID = d.USUBJID;
quit;
```

Now we can prepare treatment group counts (N/output denominators) and labels which we will use for the header output.

```
proc sql noprint;
  ** Get summary treatment information **;
  create table trt as
  select
     t.TRTN
    ,t.TRT
    ,COUNT(distinct t.USUBJID)                as N   /* N - denominator */
    ,trim(left(t.TRT))||" (N="||
     compress(put(CALCULATED N, best.))||")" as LBL /* treatment label */
  from asl_p as t
  group by t.TRTN, t.TRT
  order by t.TRTN, t.TRT;

  ** Save treatment labels into macro variables **;
  select LBL
  into :lbl1-:lbl%left(&tr_num)
  from trt;
quit;
```

OUTPUTS WITH A FIXED STRUCTURE

Fixed structure outputs display summary information in a pre-defined format and are commonly used for safety analyses. A sample of such output is presented in Table 1, where the value outside the brackets shows number of subjects and the value inside the brackets displays portion (percentage of the total number of subjects in the treatment group).

Table 1: Sample of output with a fixed structure

	Treatment – 1 (N = XX)	**…**	**Treatment – T (N = XX)**
Subjects completed the study	X (XX.X%)	…	X (XX.X%)
Subjects discontinued the study	X (XX.X%)	…	X (XX.X%)
Subjects who have at least one record in the analysis dataset	X (XX.X%)	…	X (XX.X%)

The SQL approach presented below is based on the following steps:

1. Create the template skeleton dataset, with ordering variable for category.
2. Using the Cartesian Product (cross join), extend the skeleton with treatment group information and their counts.
3. For each category, compute numbers and percentages for each treatment group.
4. Format values according to template.
5. Go from vertical structure to horizontal (transpose by treatment).

```
 proc sql noprint;
  ** Define skeleton dataset **;
  create table skel
  (
     ID  num
    ,ORD num
    ,TXT char(100)
   );
  ** Fill our our skeleton dataset with data **;
  insert into skel values(1,1, "Subjects completed the study");
  insert into skel values(1,2, "Subjects discontinued study");
  insert into skel values(1,3, "Subjects who have at least one record in the
analysis dataset");

  ** Add treatment information to our skeleton **;
  create table part1 as
  select *, . as Pcnt
  from
    trt
    cross join skel;

  ** Count those who completed study **;
  update part1 as p
  set Pcnt = (select COUNT(distinct t.USUBJID)
              from asl_p as t
              where t.TRTN = p.TRTN
                and t.COMPFL)
  where ORD = 1;

  ** Count those who have discontinued study **;
  update part1 as p
  set Pcnt = (select COUNT(distinct t.USUBJID)
              from asl_p as t
              where t.TRTN = p.TRTN
                and t.DISCFL)
  where ORD = 2;

  ** Count those who have record in ANL dataset **;
  update part1 as p
  set Pcnt = (select COUNT(distinct t.USUBJID)
              from anl_p as t
              where t.TRTN = p.TRTN)
  where ORD = 3;

  ** Prepare result variable **;
  create table fin1 as
  select
     p.ID
    ,p.ORD
    ,p.TXT
    ,p.TRTN
    ,p.TRT
    ,p.LBL
    ,CASE
       when p.Pcnt ne 0 then
            put(p.Pcnt, 6.)||" ("||put((p.Pcnt/p.N)*100, 5.1)||"%)"
       else put(p.Pcnt, 6.)
     END as RES length=30
  from part1 as p
  order by p.ID, p.ORD, p.TXT, p.TRTN, p.TRT;
quit;
```

Assigning treatment labels can be performed in a macro loop which allows more flexible code, but for simplicity we will maintain pure SQL.

```
proc sql noprint;
  ** Transpose dataset by treatment **;
  create table final1 as
  select
     f.ID
    ,f.ORD
    ,f.TXT as TXT1
    ,""    as TXT2    length=100
    ,MAX(CASE when TRTN = 1 then RES else "" END) as _1 label="&lbl1"
    ,MAX(CASE when TRTN = 2 then RES else "" END) as _2 label="&lbl2"
    ,MAX(CASE when TRTN = 3 then RES else "" END) as _3 label="&lbl3"
  from fin1 as f
  group by f.ID, f.ORD, f.TXT
  order by f.ID, f.ORD, f.TXT;
quit;
```

Using the skeleton update method allows us to have simple maintainable code which easily can be extended if we would like to add more categories. Also keeping vertical structure gives us possibility to have working code in case when we have more than 3 treatment groups in study.

Another way to perform a data transpose is to apply the PROC TRANSPOSE procedure. In this case we do not need to define and keep treatment labels in special macro variables as they would be obtained directly from the LBL variable.

```
proc transpose data=fin1 out=tfinal1(drop=_NAME_);
  by ID TXT ORD;
  var RES;
  id TRTN;
  idlabel LBL;
run;
```

OUTPUTS WITH DESCRIPTIVE STATISTICS

Such outputs are very common in statistical analysis. They appear in different variations which required a more complicated template for demonstration. In the ASL dataset we generated two pairs of variables (numeric and its character equivalent), and we will calculate statistics of the NUM_VAR1 variable across each value from the CHR_VAR2 variable. A sample of this output is presented in Table 2.

Table 2: Sample of output with descriptive statistics

		Treatment – 1 (N = XX)	**…**	**Treatment – T (N = XX)**
<CHR_VAR2 value X>	N	XX	…	XX
	Mean	XX.X	…	XX.X
	SD	XX.XX	…	XX.XX
	Median	XX.X	…	XX.X
	Min – Max	XX.X – XX.X	…	XX.X – XX.X

The SQL approach applied here contains the following steps:

1. With summary functions such as “MEAN”, “STD”, “MIN”, “MAX” under aggregation we can calculate the required statistics. For MEDIAN, we need to use special subquery, as it is a row function in SAS 9.3.
2. Go from horizontal structure to vertical (transpose our statistics) additionally generating an ordering variable, formatting values, and adding treatment information.
3. Go from a vertical structure to horizontal (transpose by treatment).

Creating output datasets using SQL (Structured Query Language) only

```
proc sql noprint;
  ** Calculate requested stats **;
  ** MEDIAN is a row function in SAS 9.3, so we need to apply a trick **;
  create table stat1 as
  select
     t.TRTN
    ,t.TRT
    ,t.CHR_VAR2
    ,COUNT(*)            as n
    ,MEAN(t.NUM_VAR1)    as MEAN
    ,STD(t.NUM_VAR1)     as STD
    ,MIN(t.NUM_VAR1)     as MIN
    ,MAX(t.NUM_VAR1)     as MAX
    ,(select
        AVG(m.NUM_VAR1)
      from
        (select
            a.NUM_VAR1
         from
            asl_p as a
            cross join asl_p as b
         where a.TRTN = t.TRTN
           and b.TRTN = t.TRTN
           and a.CHR_VAR2 = t.CHR_VAR2
           and b.CHR_VAR2 = t.CHR_VAR2
         group by a.NUM_VAR1
         having SUM(CASE when a.NUM_VAR1 = b.NUM_VAR1 then 1 else 0 END) >=
            abs(SUM(sign(a.NUM_VAR1 - b.NUM_VAR1)))) as m
      ) as MEDIAN
  from  asl_p as t
  group by t.TRTN, t.TRT, t.CHR_VAR2
  order by t.TRTN, t.TRT, t.CHR_VAR2;
quit;
```

Descriptive statistics also can be calculated using the PROC MEANS procedure on sorted data.

```
proc sort data=asl_p out=asl_ps;
  by TRTN TRT CHR_VAR2;
run;

proc means data=asl_ps noprint;
  by TRTN TRT CHR_VAR2;
  var NUM_VAR1;
  output out = stat2 (drop = _type_ _freq_)
  n      = n
  mean   = mean
  min    = min
  max    = max
  median = median
  std    = std;
run;
```

Received statistics can be checked by PROC COMAPRE procedure.

```
proc compare base = stat1 compare = stat2
             method = absolute criterion = 0.0000001
             %str(warn)ings listall MAXPRINT=(200,4000);
run;
```

Now in one SQL script we will perform several operations to prepare data for a treatment transpose. For this we will use one big subquery to construct a vertical structure from our horizontal statistics. Also, here we need to take care about order for newly created items (the ORD variable is added for this purpose) and format received statistics (the RES variable will keep the result value). When this subquery is done, we can add treatment information.

```
proc sql noprint;
  ** Go from horizontal structure to vertical **;
  ** Add treatment information                 **;
  ** Prepare result variable                   **;
  create table fin2 as
  select
     d.*
    ,t.LBL
  from
    (select
        2                          as ID
       ,1                          as ORD
       ,s.CHR_VAR2                 as TXT1 length = 100
       ,"n"                        as TXT2 length = 100
       ,s.TRTN
       ,s.TRT
       ,put(s.n, 5.)               as RES length=30
     from stat1 as s
     union all
     select
        2                          as ID
       ,2                          as ORD
       ,s.CHR_VAR2                 as TXT1 length = 100
       ,"Mean"                     as TXT2 length = 100
       ,s.TRTN
       ,s.TRT
       ,put(s.mean, 7.1)           as RES length=30
     from stat1 as s
     union all
     select
        2                          as ID
       ,3                          as ORD
       ,s.CHR_VAR2                 as TXT1 length = 100
       ,"SD"                       as TXT2 length = 100
       ,s.TRTN
       ,s.TRT
       ,put(s.std, 8.2)            as RES length=30
     from stat1 as s
     union all
     select
        2                          as ID
       ,4                          as ORD
       ,s.CHR_VAR2                 as TXT1 length = 100
       ,"Median"                   as TXT2 length = 100
       ,s.TRTN
       ,s.TRT
       ,put(s.median, 7.1)         as RES length=30
     from stat1 as s
     union all
     select
        2                          as ID
       ,5                          as ORD
       ,s.CHR_VAR2                 as TXT1 length = 100
       ,"Min - Max"                as TXT2 length = 100
       ,s.TRTN
       ,s.TRT
       ,compress(put(s.MIN, 7.1))||" - "||
        compress(put(s.MAX, 7.1)) as RES length=30
     from stat1 as s
    ) as d
    join trt as t on t.TRTN = d.TRTN
  order by d.ID, d.TXT1, d.ORD, d.TRTN, d.TRT;
quit;
```

Similar to what we have done for fixed outputs, we perform a final transpose by treatment.

```
proc sql noprint;
  ** Transpose dataset by treatment **;
  create table final2 as
  select
     f.ID
    ,f.ORD
    ,CASE when f.ORD = 1 then f.TXT1 else "" END as TXT1
    ,f.TXT2
    ,MAX(CASE when TRTN = 1 then RES else "" END) as _1 label="&lbl1"
    ,MAX(CASE when TRTN = 2 then RES else "" END) as _2 label="&lbl2"
    ,MAX(CASE when TRTN = 3 then RES else "" END) as _3 label="&lbl3"
  from fin2 as f
  group by f.ID, f.TXT1, f.ORD, f.TXT2
  order by f.ID, f.TXT1, f.ORD, f.TXT2;
quit;
```

OUTPUTS WITH A FLEXIBLE 2-LEVEL HIERARCHY

Flexible 2-level hierarchy outputs are also very common in statistical analysis. As an example, we can take Adverse Event output, where the 1st level would be the System Organ Class (SOC) variable and the 2nd level would be the related Preferred Term (PT). This output is flexible because the list of displayed SOCs and PTs is unknown at the beginning and can change after the data changes. For illustration in ANL dataset, we have created two dependent variables, the Main category (MCAT) and the Sub category (SCAT) which are generic equivalents of SOC and PT.

Sample of this output is presented in Table 3, where the number in brackets shows the number of records (events) defined by the MCAT/SCAT values and the number outside the brackets displays the number of unique subjects.

Table 3: Sample of output with a flexible 2-level hierarchy

Main category (MCAT)	Sub category (SCAT)	Treatment – 1 (N = XX)	…	Treatment – T (N = XX)
<MCAT value M1>		XX (XX)	…	XX (XX)
	<SCAT value M1S1>	XX (XX)	…	XX (XX)
	<SCAT value M1S…>	XX (XX)	…	XX (XX)
	<SCAT value M1SK>	XX (XX)	…	XX (XX)
<MCAT value MX>		XX (XX)	…	XX (XX)
	<SCAT value MXS1>	XX (XX)	…	XX (XX)
	<SCAT value MXS…>	XX (XX)	…	XX (XX)
	<SCAT value MXSN>	XX (XX)	…	XX (XX)

The SQL approach applied here contains the following steps:

1. Using two subqueries as input tables, we get all of the required data in a vertical view, where the first subquery is responsible for constructing a skeleton of 2-level structure output (combination of MCAT/SCAT based on existing data) and the second subquery prepares subject and event counts.
2. Go from a vertical structure to horizontal (transpose by treatment).

Pcnt variable will be responsible for subject counts and for event counts we will create an Ecnt variable. Obviously, the Pcnt value (number of unique subjects) can't be greater than the Ecnt value (number of occurred events) for any MCAT/SCAT combinations.

The ordering variable ORD will be set to missing because we want to apply sorting of items based on alphabetical order of the MCAT and SCAT variables.

Also, the method we applied here can be easily extended to 3-level hierarchy outputs. For example, in the case where an existing AE output Severity variable will be added.

```
proc sql noprint;
  ** Prepare skeleton and counts using subqueries **;
  create table fin3 as
  select
     b.*
    ,3 as ID
    ,CASE when ~missing(t.Pcnt) then t.Pcnt else 0 END  as Pcnt
    ,CASE when ~missing(t.Ecnt) then t.Ecnt else 0 END  as Ecnt
    ,CASE when ~missing(b.SCAT) then "" else b.MCAT END as TXT1 length = 100
    ,b.SCAT                                             as TXT2 length = 100
    ,put(CALCULATED Pcnt, 4.)||
     " ("||put(CALCULATED Ecnt, 4.)||")"                as RES  length=30
  from
    (select *
     from
       trt
       cross join (select distinct MCAT, "" as SCAT from anl_p)
     union all
     select *
     from
       trt
       cross join (select distinct MCAT, SCAT from anl_p)) as b
    left join
    (select
       p.TRTN
      ,p.MCAT
      ,""  as SCAT
      ,COUNT(distinct p.USUBJID) as Pcnt
      ,COUNT(*)                  as Ecnt
    from anl_p as p
    group by p.TRTN, p.MCAT
    union all
    select
       p.TRTN
      ,p.MCAT
      ,p.SCAT
      ,COUNT(distinct p.USUBJID) as Pcnt
      ,COUNT(*)                  as Ecnt
    from anl_p as p
    group by p.TRTN, p.MCAT, p.SCAT) as t on b.TRTN = t.TRTN
      and b.MCAT = t.MCAT
      and b.SCAT = t.SCAT
  order by b.TRTN, b.MCAT, b.SCAT;


  ** Transpose dataset by treatment **;
  create table final3(drop = MCAT SCAT) as
  select
     f.ID
    ,. as ORD
    ,f.MCAT
    ,f.SCAT
    ,f.TXT1
    ,f.TXT2
    ,MAX(CASE when TRTN = 1 then RES else "" END) as _1 label="&lbl1"
    ,MAX(CASE when TRTN = 2 then RES else "" END) as _2 label="&lbl2"
    ,MAX(CASE when TRTN = 3 then RES else "" END) as _3 label="&lbl3"
  from fin3 as f
  group by f.ID, f.MCAT, f.SCAT, f.TXT1, f.TXT2
  order by f.ID, f.MCAT, f.SCAT, f.TXT1, f.TXT2;
quit;
```

CONCLUSION

A huge number of statistical analysis outputs are based on these three types of templates. This paper demonstrates that PROC SQL is a powerful tool to create a wide variety of outputs. However, from the illustrated examples, SQL code might appear to become more complicated when multiple nested subqueries are utilized. This means that you may need to split your queries into smaller parts or possibly use a solution outside of SQL. PROC SQL sometimes can be a good alternative to SAS data step and SAS procedures.

REFERENCES

SAS® 9.4 SQL Procedure User's Guide. SAS Institute Inc. 2013. Available at URL: http://support.sas.com/documentation/cdl/en/sqlproc/65065/PDF/default/sqlproc.pdf

Chao Huang. Top 10 SQL Tricks in SAS®. Available at URL: http://support.sas.com/resources/papers/proceedings14/1561-2014.pdf

CONTACT INFORMATION

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

Name: Andrii Stakhniv
Enterprise: Experis Clinical
Address: 23 Baggoutovskaya street
City, State ZIP: Kyiv 04107, Ukraine
Work Phone: +1 407 512 10 06 ext. 2407
E-mail: andrii.stakhniv@intego-group.com