Top 10 Most Powerful Functions for PROC SQL

Chao Huang. Oklahoma State University
Yu Fu. Oklahoma State University

ABSTRACT

PROC SQL is not only one of the many SAS procedures and also a distinctive subsystem with all common features from SQL (Structured Query Language). Equipped with PROC SQL, SAS upgrades to a full-fledging relational database management system. PROC SQL provides alternative ways to manage data other than the traditional DATA Step and SAS procedures. In addition, SAS's built-in functions are the add-on tools to increase the power of PROC SQL. In this paper, we illustrate ten popular SAS functions, which facilitate the capacity of PROC SQL in data management and descriptive statistics.

INTRODUCTION

Structured Query Language (SQL) is a universal computer language for all relational database management systems. PROC SQL is the implementation of the SQL syntax in SAS. It first appeared in SAS 6.0, and since then has been widely used for SAS users. PROC SQL greatly increases SAS's flexibility in handling data, especially for multiple-table joining and database access. There are a number of comparisons between the DATA Step and the SQL procedure in SAS [1]. A majority of SAS functions can be directly used in the SQL procedure. And the PROC procedure also enjoys a few unique functions. In this paper, we select the 10 SAS functions, and show their usage in the SQL procedure. For demonstration purpose, we simulate a Social Security Number (SSN) dataset with two entries from different sources. Each entry of the 1000 records should be identical but some values are missing.

```
*****(0) Simulate two datasets for demonstration*******************;
*****(0.1) Simulate a dataset for two SSN entries******************;

data ssn_data;
    do i = 1 to 1000;
      ssn1 = ceil((ranuni(1234)*1E9));
      ssn2 = ssn1;
      if ssn1 le ceil((ranuni(1000)*1E9)) then call missing(ssn1);
      if ssn2 le ceil((rannor(2000)*1E9)) then call missing(ssn2);
      drop i;
      output;
   end;
   format ssn1 ssn2 ssn11.;
run;
```

We also simulate a patient-visiting dataset with three patient IDs. Every patient receives three different treatments at each visit. The effects of the treatments (1 means effective; 0 means not effective) and the cost for each visit are recorded. Other than the two simulated datasets, two datasets shipped with SAS, SASHELP.CLASS and SASHELP.CARS, are also used in the paper.

```
*****(0.2) Simulate a dataset for hospital visits *****************;

 data hospital_data;
      input id visit treat1 treat2 treat3 cost;
      format cost dollar8.2;
      cards;
      1 1 0 0 0 520
      1 2 1 0 0 320
      1 3 0 1 0 650
      2 1 1 0 0 560
      2 2 1 0 0 360
      3 1 1 0 0 500
      3 2 0 0 1 350
      ;;;
 run;
```

TOP 10 FUNCTIONS FOR THE SQL PROCEDURE IN SAS

1. The MONOTONIC function

The MONOTONIC function is quite similar to the internal variable _N_ in DATA Step. We can use it to select the records according to their row number. For example, we choose the SSNs from the 501th line to the 888th line in the SSN dataset.

```
****(1) MONOTONIC: specify row numbers*****************************;

proc sql;
   select *
   from ssn_data
   where monotonic() between 501 and 800
;quit;
```

2. The COUNT, N and NMISS functions

These counting functions are especially useful in data cleaning. By using them, the detailed missing status is shown in only one output table. For the SSN dataset, we can display the total numbers of the missing and non-missing values for each SSN entry.

```
****(2) COUNT/N/NMISS: find total and missing values***************;

proc sql;
select count(*) as n 'Total number of the observations',
       count(ssn1) as m_ssn1 'Number of the non-missing values for ssn1',
       nmiss(ssn1) as nm_ssn1 'Number of the missing values for ssn1',
       count(ssn2) as m_ssn1 'Number of the non-missing values for ssn2',
       nmiss(ssn2) as nm_ssn2 'Number of the missing values for ssn2'
from ssn_data;
quit;
```

3. The COALESCE function

The COALESCE function does the magic to combine multiple rows into a single one with any non-missing value. In this example, there are two rows of SSNs, and supposedly they should be identical each other. However, some of them are missing due to input errors or other reason. The COALESCE function in the SQL statement below checks the value of the two rows and returns the first non-missing value, which maximizes the SSN information.

```
****(3) COALESCE: combine values among columns*********************;
proc sql;
     select monotonic() as obs, coalesce(ssn1, ssn2) as ssn format = ssn11.
     from ssn_data
;quit;
```

4. The MISSING function

The MISSING function returns a Boolean value for a variable (0 when non-missing; 1 when missing). In the example below, the missing status of the values in the SSN dataset is displayed row by row.

```
****(4) MISSING: return Boolean for missing value******************;
proc sql ;
   select monotonic() as obs,
     (case sum(missing(ssn1), missing(ssn2))
           when 0 then 'No missing'
           when 1 then 'One missing value'
             else 'Both missing values'
      end) as status 'Missing status'
   from ssn_data;
quit;
```

5. The SPEDIS and SOUNDEX functions

The two functions can fulfill fuzzy matching. For example, if we want to examine the first entry of the SSN dataset to see if there is any possible duplicate, we can use the SPEDIS function in the SQL statement to look up any pair of the records. Here we set the argument to be 25 in order to detect any singlet [2].

```
****(5)SPEDIS/SOUNDEX: fuzz matching*******************************;
****(5.1)SPEDIS: find spelling mistakes****************************;

proc sql;
   select a.ssn1 as x, monotonic(a.ssn1) as x_obs,
          b.ssn1 as y, monotonic(b.ssn1) as y_obs
   from ssn_data as a, ssn_data as b
   where (x gt y) and (spedis(put( x, z11.), put( y, z11.)) le 25);
quit;
```

For human names, we can check similarities by the SOUNDEX function to avoid duplicates [3]. The SASHELP.CLASS has 19 names. Phonically, John and Jane look similar according to the SOUNDEX function.

```
****(5.2)SOUNDEX: find phonic similarity*****************************;

proc sql;
     select a.name as name1, b.name as name2
     from sashelp.class as a, sashelp.class as b
     where soundex(name1) = soundex(name2) and (name1 gt name2);
quit;
```

6. The RANUNI function

This function does simple random sampling like PROC SURVEYSELECT. We can specify the OUTOBS option at the beginning to choose the sample size.

```
****(6)RANUNI: simple random sampling*******************************;

proc sql outobs=30;
     select *
     from ssn_data
     order by ranuni(1234);
quit;
```

7. The MAX function

The MAX function returns the maximum value and sometimes simplifies column-wise aggregation. For the patient-visiting dataset, if we need to know if each treatment is effective for the patients, it may take some time to code the RETAIN statement and temporary variables at DATA Step, while the MAX function at PROC SQL is quite straightforward.

```
****(7)MAX: find the maximum value for each column*****************;

proc sql;
      select id, max(treat1) as effect1 'Effect after Treatment 1',
      max(treat2) as effect2 'Effect after Treatment 2',
      max(treat3) as effect3 'Effect after Treatment 3'
      from hospital_data
      group by id
;quit;
```

8. The IFC and IFN functions

The two functions play a role like the CASE-WHEN-END statements in typical SQL syntax, if the condition is about a binary selection. The IFC function deals with character variables, while the IFN function is for numbers. For the patient-visiting dataset, we can use the two functions together to find the total cost, the discounted cost (a 15% discount is applied if the total cost is greater than $1,000), and whether the first treatment is effective for each patient.

```
****(8)IFC/IFN: binary selection for either character and number****;

proc sql;
   select id, ifc(max(treat1) = 1, 'Yes', 'No') as overall_effect
      length = 3 'Any effect after treatments',
      sum(cost) as sum_cost format = dollar8.2 'Total cost',
      ifn(calculated sum_cost ge 1000,
          calculated sum_cost*0.85,
          calculated sum_cost*1) as discounted_cost
      format=dollar8.2 'Total cost after discount if any'
   from hospital_data
   group by id;
quit;
```

9. The UNIQUE function

This function is very convenient to show the number of the levels for every categorical variable.

```
****(9)UNIQUE: find the levels of categorical variables***********;

proc sql;
      select count(unique(make)) as u_make 'Number of the car makers',
      count(unique(origin)) as u_origin 'Number of the car origins',
      count(unique(type)) as u_type 'Number of the car types'
      from sashelp.cars;
quit;
```

10. The PUT function

We can apply the PUT function with a user-defined format by PROC FORMAT in the WHERE statement to create filters. For the SASHELP.CARS dataset, this strategy is used to choose only the high or medium priced cars.

```
****(10)PUT: create an filter by user-defined format**************;

proc format;
  value range
      40000-high='High'
      26000 -< 40000='Medium'
      other ='Low';
run;

proc sql;
     select model, make, msrp,
            msrp as range 'Price Range' format = range.
      from sashelp.cars
      where put(msrp, range.) in ('High', 'Medium');
quit;
```

CONCLUSION

The combination of SAS's powerful functions and the SOL procedure will benefit SAS users in data management and descriptive statistics.

REFERENCES

1. Christianna S. Williams. 'PROC SQL for DATA Step Die-hards'. SAS Global Forum Proceeding 2008.

http://www2.sas.com/proceedings/forum2008/185-2008.pdf
2. Yefim Gershteyn.'Use of SPEDIS Function in Finding Specific Values'. SAS Users Group International 25.

http://www2.sas.com/proceedings/sugi25/25/cc/25p086.pdf
3. Amanda Roesch.'Matching Data Using Sounds-Like Operators and SAS® Compare Functions'. Northeast SAS Users Group 2011.

http://www.nesug.org/Proceedings/nesug11/ap/ap07.pdf