I DIDN'T KNOW YOU COULD DO THAT! REVELATIONS FROM THE ADVANCED SAS® CERTIFICATION EXAM

Patricia Hettinger, Certified SAS® Professional, Oakbrook Terrace, IL

ABSTRACT

Many people have extreme test anxiety. But studying for a test can give you a lot of insight into the subject matter you may not have had before. This is certainly true of the SAS certification exams. Even If you have been a successful SAS professional, there is still a lot to be gained from certification beyond simple credentials. This paper details some features of SAS that were a complete revelation to the author and many be of service to you in your own daily work.

INTRODUCTION

It's nice to have the Base SAS certification under your belt and really do you need any more than that? Most of us SAS analysts are probably steadily employed. We may even be recruited for jobs almost daily. So, is there any benefit for going through another certification besides as something nice to have on a resume? The answer is most definitely 'Yes'! Let's explore a few ways preparing for the advanced SAS certification exam can help you, especially in sparking creativity with what you learned from the process.

BACKGROUND

The author has been steadily employed for the past ten years as a SAS professional with just six months out during the financial crisis. Base SAS certification seemed desirable then because of a comment from more than one prospective employer: "Why don't you have a SAS certification? I would think someone like you would have that at least". Obtaining it definitely helped with getting past the employment system gatekeepers. And there it stood for several years as full employment made any additional certification seem unnecessary. The topic didn't come up again until winning an advanced certification guide in a raffle by the SAS Institute. By that time, a few more projects on the resume increased confidence so why not? This should be so easy. Just go through the study guide to make sure nothing was missing before the exam. It shouldn't take more than a month even with working full-time. But this ended up meaning more than just passing an exam. There were features encountered that the author never saw before, even in reviewing code written by many others. The exam date kept being pushed back until finally taking and passing the exam the spring of 2017.

THE SECTIONS ON THE TEST

Accessing Data Using SQL
Advanced Programming Techniques
Macro Processing

ACCESSING DATA USING SQL

Referencing a Calculated Column

You can create a new column in your query and name it. But you may noticed that you can't refer to it again in your query as is. For example, trying to execute the query below will give you an error:

```
proc sql;
        Select years_with_company*vacation_rate as vacation_allowed,
        vacation_allowed-vacation_taken as vacation_left
        From time_sheets;
```

```
ERROR:  The following columns were not found in the contributing table:
vacation_allowed
```

If you added a WHERE clause to find everyone with 10 days or more of vacation allowed, you would get the same error:

```
        select years_with_company*vacation_rate as vacation_allowed,
        vacation_allowed-vacation_taken as vacation_left
        from time_sheets
        where vacation_allowed => 10;
```

The strange thing is that you could reference the new column when doing an aggregate query attempting to find project teams with vacation days left greater than 10 for future capacity planning:

```
select Sum((years_with_company*vacation_rate) – vacation_taken)) as vacation_left
from time_sheets
group by project_team
having vacation_left > 10;
```

The key word CALCULATED with the new column name will let you use the new column anywhere. A small thing but one with a potentially large impact if you do a more complicated column creation in your query like this:

```
select name,
        case
                when status='G' then .1
                when status='N' then .1
                when status='M' then .12
                when status='B' then .15
                else .175
        end
as multipler_value,
balance*calculated multiplier_value as premium
from balance_table
```

The Problem with Outer Joins

One problem with doing a full outer join is getting missing values for the keys that didn't match. These are our two input data sets:

Account_Table
Acct
1234
5678
0246
9786

Recovery_Table
Acct
1234
6780
0246
1543

We desire to see all of the account numbers even if there is no match:

Account_nbr
0246
1234
1543
5678
6780
9786

We could use a CASE clause to populate the account_nbr column:

```
select a.*,
b.stat_code, b.risk_code ,
case
        when a.acct is null then b.acct
        else a.acct
end as account_nbr
from account_table a
full join recovery_table b
        on a.acct=b.acct
sort by calculated account_nbr;
```

Yes, we can sort on a calculated column too.

The COALESCE function lets us do this a bit more easily:

```
Select a.*,b.stat_code, b.risk_code ,
coalesce ( a.acct , b.acct ) as account_nbr
from account_table a
full join recovery_table b,
on a.acct=b.acct
sort by calculated account_nbr;
```

Dropping and Keeping Variables in PROC SQL

If you've ever created a new SAS data set by joining two SAS data sets with the same variable names, you have probably seen a message like this: variable xxx already exists on table yyyyy. This is because PROC SQL will write a variable to a new data set on a 'first come, first served' basis, taking the column from the first table in the join. This is fine if a) you don't mind messages like this in your log and b) you wanted the columns in the first table. But what if you didn't? You can specify variables to be dropped or kept in PROC SQL much like in the DATA step? You would write your query like:

```
proc sql;
        create table new_values
        as select A.*,B.stat_code, b.risk_code
        from recovery_table (drop=stat_code risk_code) B
        left join
        reset_table B
        on a.acct=b.acct;
```

The above code would create a new table with status and risk codes from the reset_table data set instead of from recovery_table, the first specified when there was a match between the two tables on the acct variable and give missing values on the nomatch condition. Dropping the stat_code and risk_code columns from the recovery table also removes the 'already exist' message.

ADVANCED PROGRAMMING TECHNIQUES – A TALE OF TWO VIEWS

Did you know that setting the MSGLEVEL option to I will write index usage notes to the log whenever PROC SQL or the DATA step executes? This can be very helpful in understanding what is happening when querying SAS datasets with indexes.

One common situation is multiple datasets with the same structure containing subsets of the data. For the example below, we will refer to customer data sets containing customers for different regions, in this case north, south, east and west. There is a unique index on customer_id in each of them. The data sets are all updated daily.

Suppose we want to extract a record for customer 1234 but we don't know which dataset it is in. We could query the datasets one by one or to make things easier, set up a view combining them. Perhaps a SAS view combining the data sets will do the trick? Let's turn fullstimer on, set msglevel to I and see this in action.

Here's our SAS data view of the four data sets:

```
        Data sasregions/view=sasregions;
        Set
                cdir.north
                cdir.south
                cdir.east
                cdir.west
        ;
```

Then, let's extract the record using the DATA step:

```
        DATA extract_one;
        Set sasregions;
        Where customer_id = 1234;
```

You will get something like this in the log, depending of course, on which day you run it:

```
NOTE:  There were 3218963 observations read from the data set cdir.north.
NOTE:  There were 2976337 observations read from the data set cdir.south
NOTE:  There were 3076337 observations read from the data set cdir.east
NOTE:  There were 3146337 observations read from the data set cdir.west
NOTE:  There were 1 observations read from the data set work.sasregions
       WHERE customer_id = 1234
```

Stimer output was:

```
       Real time                    7:24.22
       User cpu time                3:48.55
       System cpu time              17.60
       Memory                       112579.00k
       OS Memory                    138944.00k
       Timestamp                    09/13/2017 05:45:00 PM
       Page Faults                         0
```

The index was not used, obviously – all of the observations were read sequentially, very inefficient in this case. Why is this so? Carefully reading of the study guide indicates that no SAS view will ever be able to use the underlying indexes. Is there a way to get around this? Let's create an equivalent SQL view as another option:

```
        proc sql;
                create view sqlregions
                        as select *
                        from cdir.north
                union
                        select *
                        from cdir.south
                union
                        select *
                        from cdir.east
                union
                        select *
                        from cdir.west
                ;
```

Now, let's try extracting the same record again with an SQL query first:

```
proc sql;
        create table extract_one_sql1
        as select *
        from sqlregions
        where customer_id = 1234;
```

Your log will now look much like this:

```
INFO:  Index CUSTOMER_ID selected for where clause optimization
INFO:  Index CUSTOMER_ID selected for where clause optimization
INFO:  Index CUSTOMER_ID selected for where clause optimization
INFO:  Index CUSTOMER_ID selected for where clause optimization
```

What happens if we extract the record in the DATA step?

```
data extract_one_sql2;
set sqlregions;
where customer_id eq 1234;
```

We get the same information message as we did with the SQL query PLUS some more details on how the new data set was created:

```
NOTE:  There were 1 observations read from the data set cdir.north
       WHERE CUSTOMER_ID = 1234
NOTE:  There were 0 observations read from the data set cdir.south
       WHERE CUSTOMER_ID = 1234
NOTE:  There were 0 observations read from the data set cdir.east
       WHERE CUSTOMER_ID = 1234
NOTE:  There were 0 observations read from the data set cdir.west
       WHERE CUSTOMER_ID = 1234
NOTE:  There were 1 observations read from the data set WORK.sqlregions
       WHERE CUSTOMER_ID = 1234
```

The stats from Stimer look much better too:

```
       Real time                  1:11.75
       User cpu time              37.79 seconds
       System cpu time            0.07 seconds
       OS Memory                  182189.53k
       Timestamp                  09/13/2017 04:29:10 PM
       Page Faults                       0
       Page Reclaims                     2353
       Page Swaps                        0
       Voluntary Context Switches        124
       Involuntary Context Switches      10883
       Block Input Operations            0
       Block Output Operations           0
       Page Reclaims                     1005
       Page Swaps                        0
       Voluntary Context Switches        46256
       Involuntary Context Switches      89906
       Block Input Operations            0
       Block Output Operations           0
```

You could even create another SQL view based off the sqlregions view and still have the underlying index available to you.

Perhaps not all the variables are needed so we will create a view with just a few of them. Instead of recreating the sqlregions view, we can create a new view using it as a source:

```
        proc sql;
                create view short_version
                as select customer_id, payment_history36, current_balance, last_payment
                from sqlregions;
```

But creating a SAS view from the SQL view will still not allow use of the index:

```
        data short_version_sas/view=short_version_sas;
        set sqlregions (keep=customer_id payment_history36 current_balance last_payment);
```

```
NOTE:  There were 3218963 observations read from the data set cdir.north.
NOTE:  There were 2976337 observations read from the data set cdir.south
NOTE:  There were 3076337 observations read from the data set cdir.east
NOTE:  There were 3146337 observations read from the data set cdir.west
NOTE:  There were 1 observations read from the data set work.short_version_sas
       WHERE customer_id = 1234
```

MACRO PROCESSING – TO THE DATA STEP AND BEYOND

Macros can make your life easier by making it possible to write code just once and run as needed with different parameters. You might have used it to execute different data steps or procedures. But you can actually execute macros within a DATA step or even a procedure like PRINT. For example, you might want to print some variables in a certain order and limit the ones you want printed. This works best for variables that are named as members of a group such as week1 – week52 in the next example. The source data set contains weekly weigh-ins (weeky_record). Suppose you want to report your weight in a given week going forward three weeks and going backwards two weeks. You could code a report like this for the period starting with week3 for the next three weeks after that and then the two weeks before in reverse order:

```
Proc print data=weekly_record;
Var week3-week6 week2 week1;
```

Your output would be:

```
Obs     Week3   Week4   Week5   Week6   Week2   Week1
1       190.0   189.5   188.0   187.0   191.0   191.5
```

You'd have to code this over and over again for selecting different weeks. And what happens if you ask for weeks outside of the range available? An error, of course. But you could write a macro that would be repeatable for any combination AND make sure anyone executing the macro can't go past 52 weeks on the high end nor end up at less than week 1 with this logic:

```
        %macro startit(start,forward,backward);

                %if %eval(&start + &forward) > 52 %then %let last_month=52; /*do not allow values greater than 52*/
                %else %let last_month=%eval(&start + &forward);
                %if %eval(&start - &backward) < 1 %then %let first_month=1 ; /*do not allow values less than 1*/
                %else %let first_month=%eval(&start - &backward);

                %do I = &start %to &last_month; /*output starting month plus the requested months going forward*/
                        Week&i  /*no semi_colons!*/
                %end;

                %do I = %eval(&start-1) %to &first_month %by -1; /*previous months requested, increment by -1*/
                        Week&n /*no semi_colons!*/
                %end;

        %mend startit;
```

You could then use this macro in a print procedure, say to report on someone's progress starting with week 3, going forward three weeks from and putting the prior weeks 1 and 2 at the end. You could even put all of the requested weeks into the title. If you do this, remember that as any macro variable, put the expression in double quotes or you will get only %startit(3,3,3) in your title:

```
title “Report for %startit(3,3,3)”;
proc print data=weekly_record;
var
%startit(3,3,3);
run;
```

Your report now looks like:

```
Report for Week3 Week4 Week5 Week6 Week1 Week2

Obs       Week3     Week4     Week5     Week6     Week2     Week1
1         190.0     189.5     188.0     187.0     191.0     191.5
;
```

You could use also this macro in a DATA step to extract only the variables above in the same manner:

```
data Getweeks;
set weekly_report(keep=%startit(3,3,3));
run;
```

In either case, the variables inside the macro are local to it, meaning the first_month and last_month are not available to the rest of your session unless you declare these as global first.

Getting the Current Date for Your Report

Sometimes you might have a long running batch process that crosses dates. If you want the actual run date in your report, the automatic variable SYSDATE will give the date you started the session or job, not the current date. We can get around that by using the macro function %SYSFUNC. %SYSFUNC can be used to call almost any data step function with the notable exceptions of PUT and INPUT. Fortunately, %SYSFUNC has an optional parameter specifying the desired format: %SYSFUNC(<function>,<optional format>), making the PUT and INPUT functions unnecessary. Let's say that we want to put the current date into MM-DD-YYYY format on our report, regardless of when the job was submitted. We can accomplish this by setting a variable to %sysfunc(today(),MMDDYYD10.) to get a date that you can use to timestamp your output. This expression too can go in the title as is if you don't want to create next another variable in your session:

```
Title “Report for %startit(3,3,3),  Run Date %sysfunc(today(),MMDDYYD10.)”;
```

If you ran this report on September 15, 2017, the title would appear as:

```
Report for Week3 Week4 Week5 Week6 Week1 Week2, Run Date 09-15-2017.
```

CONCLUSION

These are just a few of the things covered on the exam that may make your life easier. Or as the author did, you might find some of the topics unlikely to find practical use in your environment. Two that come to mind are SAS audit data sets that track changes to a particular data set and generation data sets that create a new generation within SAS whenever a data set is replaced. If you do use either concept, you may find them impractical in a development environment. Production environments will probably use some other software to manage changes.

But you are bound to find something you didn't know before that is directly useful. It might be a small thing but small things can have a big impact.

> “For the want of a nail the shoe was lost,
> For the want of a shoe the horse was lost,
> For the want of a horse the rider was lost,
> For the want of a rider the battle was lost,
> For the want of a battle the kingdom was lost,
> And all for the want of a horseshoe-nail.” – Benjamin Franklin

Good luck on the exam!

ACKNOWLEDGEMENTS

Thank you, Joe Butkovich for reviewing this paper and presentation

RECOMMENDING READING
SAS Certification Prep Guide – Advanced Programming for SAS 9 Third Edition – SAS Institute

CONTACT INFORMATION

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

Patricia Hettinger
Oakbrook Terrace, IL 60523
Phone: 331-462-2142, cell 630-309-3431
Email: patricia_hettinger@att.net