PharmaSUG 2019 - Paper AP-47

Confessions of a PROC SQL Instructor

Charu Shankar, SAS® Institute Inc.

ABSTRACT

After teaching at SAS for over 10 years to thousands of learners, this instructor has collected many best practices from helping customers with real-world business problems. Hear all about her confessions on making life easy with mnemonics to recall the order of statements in SQL. Learn about the data step diehard user who now loves SQL thanks to this little-known secret gem in PROC SQL. Hear about the ways in which ANSI SQL falls short and PROC SQL picks up the slack. In short, there are many confessions and so little time. session is open to all interested in improving their SQL knowledge and performance.

CONFESSION #1 - PROC SQL SYNTAX ORDER: SO FEW WORKERS GO HOME ON TIME

Every computer language has syntax order that is uniquely its own. Trying to remember the syntax is sometimes not easy for those fluent in multiple languages: human or computer. For some help in memory recall, try my mnemonic to remember the syntax order of SQL.

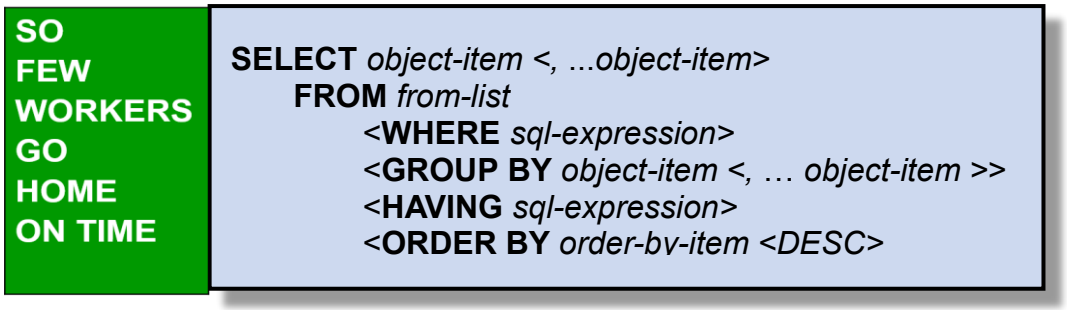


Figure 1: PROC SQL Mnemonic

Here is a PROC SQL query in its entirety. The SELECT and FROM are mandatory statements in any SQL query. Anything in triangular brackets is optional.

```
PROC SQL;
    SELECT object-item <, ...object-item>
        FROM from-list
            <WHERE sql-expression>
                <GROUP BY object-item <, … object-item >>
                    <HAVING sql-expression>
                        <ORDER BY order-by-item <DESC>
                            <, …order-by-item>>;
QUIT;
```

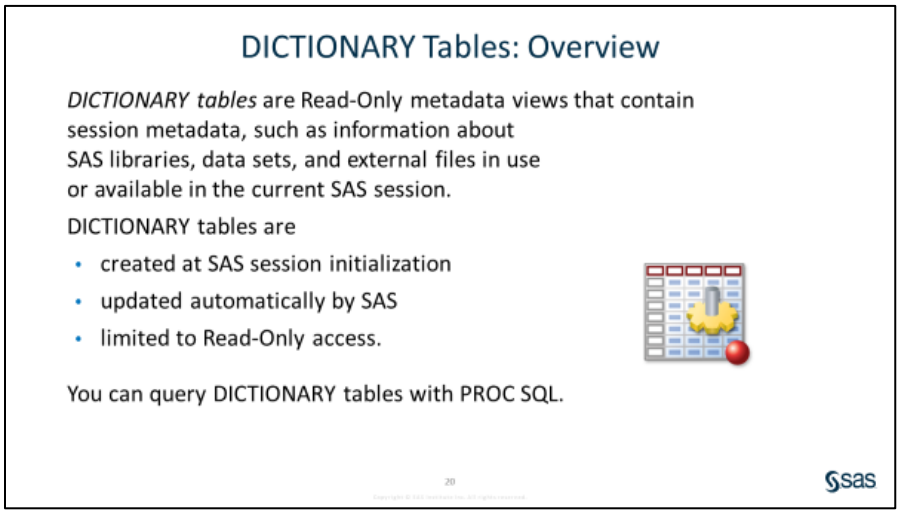
Figure 2: PROC SQL Syntax order

A *SELECT statement* is used to query one or more tables.
The FROM clause specifies the tables that are required for the query
The WHERE clause specifies data that meets certain conditions.
The GROUP BY clause groups data for processing.
The HAVING clause specifies groups that meet certain conditions.
The ORDER BY clause specifies an order for the data.

CONFESSION #2 - KNOW THY DATA: DICTIONARY TABLES

There is no magic pill that will forgive us for not knowing our data. ”Know thy data” must be the most fundamental principle that cannot be ignored. In fact I am going to go out on a limb here and say, this is the only rule that data workers must know. Everything else is SAS!

To help navigate through the inherited, sometimes messy data, my go to suggestion is dictionary tables. With the amount of heavy-duty metadata scouring, data workers do to get data intelligence, this is one confession I simply must make. I LOVE dictionary tables and cannot imagine life without them. When you see this confession revealed, I’m positive you will also feel the same way.



Know your dictionary tables

There can be over 30 Dictionary tables that provide metadata information. Our focus in this presentation will be on using data from three of the tables

DICTIONARY.TABLES

- detailed information about tables

DICTIONARY.COLUMNS

- detailed information about all columns in all tables

DICTIONARY.MEMBERS

- general information about SAS library members
-

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

To get to know the columns and what they stand for, query the dictionary table first using the following code.

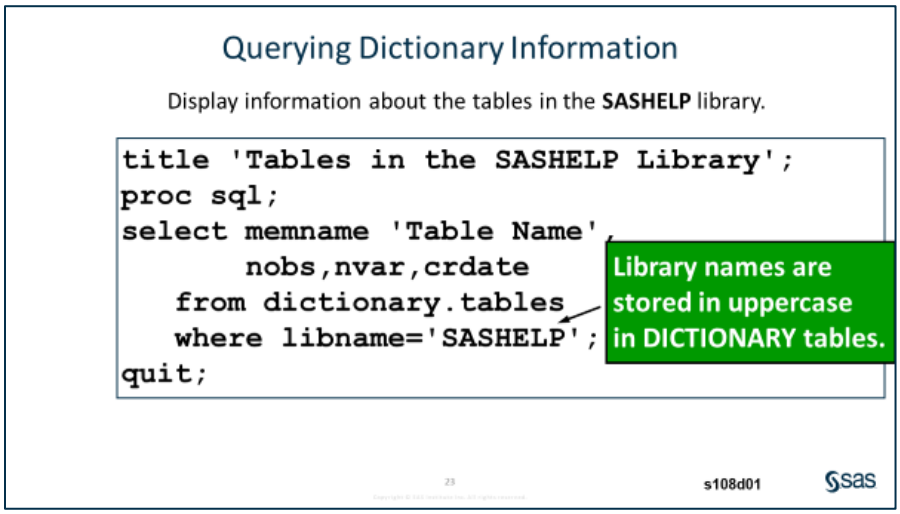
Code to describe dictionary tables

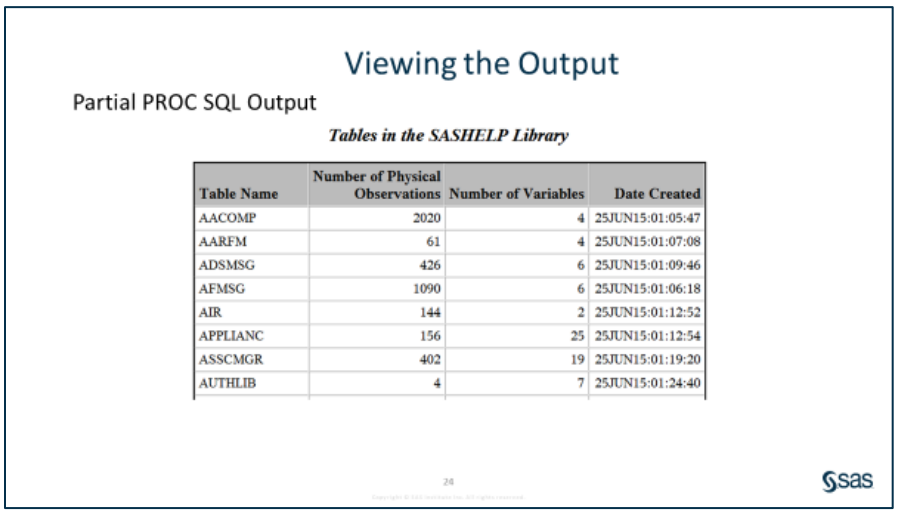
```
describe table dictionary.tables;
```

Log

```
NOTE: SQL table DICTIONARY.TABLES was created like:
create table DICTIONARY.TABLES
  (libname char(8) label='Library Name',
   memname char(32) label='Member Name',
   ...
   crdate num format=DATETIME informat=DATETIME label='Date Created',
   modate num format=DATETIME informat=DATETIME label='Date Modified',
   nobs num label='Number of Physical Observations',
   obslen num label='Observation Length',
   nvar num label='Number of Variables', ...);
```

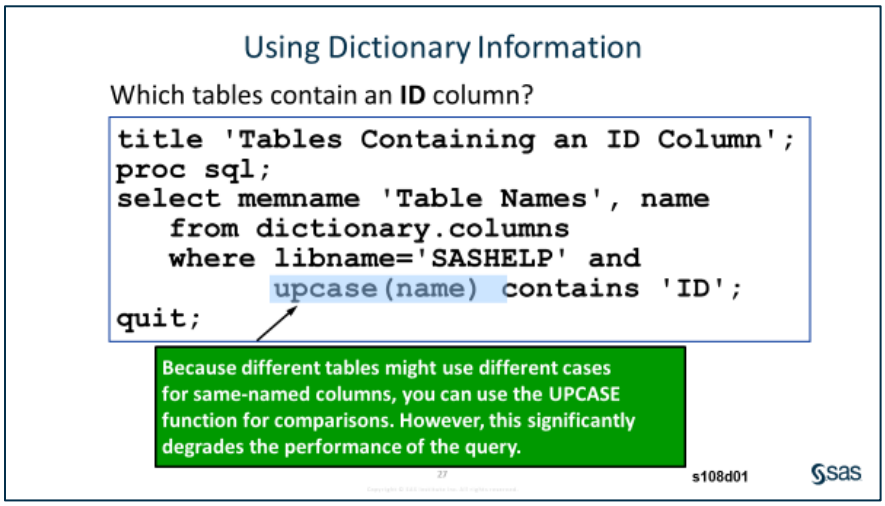
Display information about tables in SASHELP

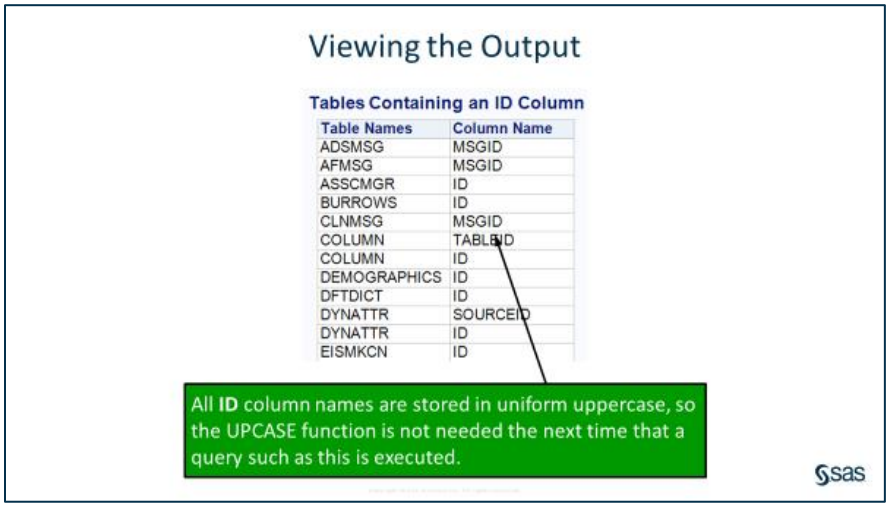




Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Finding same named columns





However, you may have observed, that this is something that Proc contents can easily do. Its not something that impresses us about a niche value that dictionary tables can add.

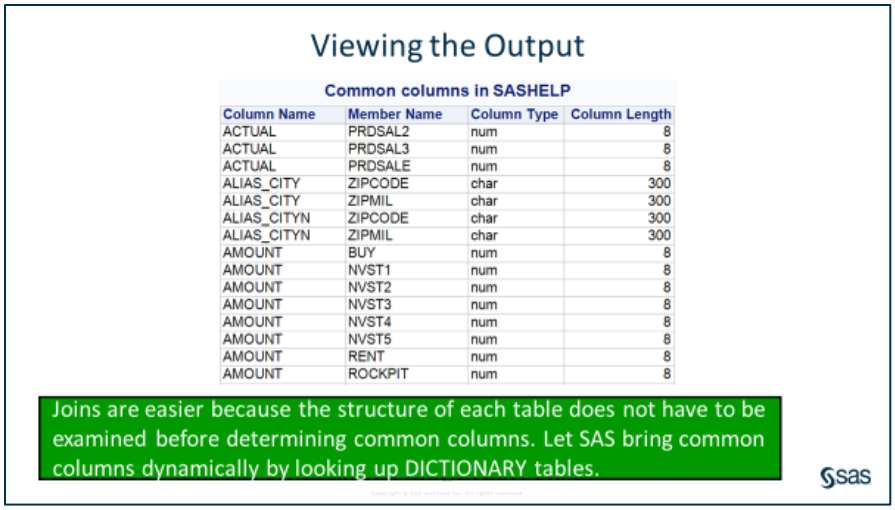
Also these past techniques to explore DICTIONARY tables work when you know the names of columns. What happens if you do not know your data, and you want SAS to retrieve all same-named columns in a library.

Are you ready for my confession #2? Use the following code to eliminate any manual work.

Code to find common column names dynamically

```
title 'Common columns in SASHELP';
proc sql;
select name, type, length, memname
   from dictionary.columns
   where libname='SASHELP'
   group by name
   having count(name) > 1;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



Using dictionary tables from the SASHELP Library

SAS provides views based on the DICTIONARY tables in the **SASHELP** library. Most of the **SASHELP** library DICTIONARY view names are similar to DICTIONARY table names, but they are shortened to eight characters or less. They begin with the letter **v** and do not end in **s**.

For example:

dictionary.tables = sashelp.vtable

Code to query dictionary tables in the SASHELP library

```
title 'Tables in the SASHELP Library';
proc print data=sashelp.vtable NOOBS ;
   var memname nobs nvar;
   where libname='SASHELP';
run;
```

An Efficiency question: PROC SQL or PRINT?

Code to compare PROC SQL with PROC PRINT

```
options fullstimer;
proc sql;
    select libname, memname, name, type, length
    from dictionary.columns
    where upcase(name) contains 'ID'
    and libname='SASHELP' and type='num';
quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.73 seconds
      user cpu time       0.42 seconds
      system cpu time     0.29 seconds
      memory              5584.18k
      OS Memory           24672.00k
      Timestamp           05/22/2018 01:52:52 PM
      Step Count          4  Switch Count  36
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Why is PROC SQL more efficient?

While querying a DICTIONARY table, SAS launches a discovery process. Depending on the DICTIONARY table being queried, this discovery process can search libraries, open tables, and execute views.

The PROC SQL step runs much faster than other SAS procedures and the DATA step. This is because PROC SQL can optimize the query before the discovery process is launched. It has to do with the processing order. The PROC SQL step runs much faster because the WHERE clause is processed before the tables referenced by the SASHELP.VCOLUMN view are opened. Therefore, it is more efficient to use PROC SQL instead of the DATA step or SAS procedures to query DICTIONARY tables.

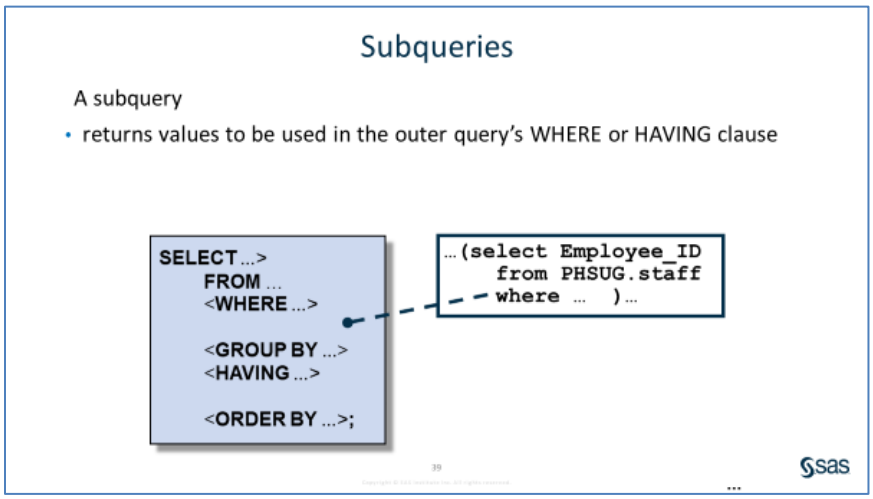
CONFESSION # 3 – STACK DATA HORIZONTALLY

Subqueries: Best Practices, Dangers of Correlated

There are many ways to stack data horizontally in SQL. I would like to show you the underlying behavior of two ways: a non-correlated subquery and a correlated subquery.

The reason this became my third confession is due to the dangerous zone that a correlated subquery can lead into. A query that is running for days must be examined closely to see if there was a correlated subquery somewhere in the lines of code that were submitted. My confession stems from a customer query whose SQL code was indeed running for days. When they sent me the code, I could spot the correlated subquery in there. Taking it out resolved the problem and the code ran healthy in a matter of minutes. I'm not saying that every long running SQL query can be attributed to a correlated subquery. But its definitely a moment for pause and reflection and examination. My goal in this confession of mine is simply to share these practices that can halt the smooth running of your query. And show you the inner workings of both correlated(bad idea generally) and a non-correlated(best practice generally).

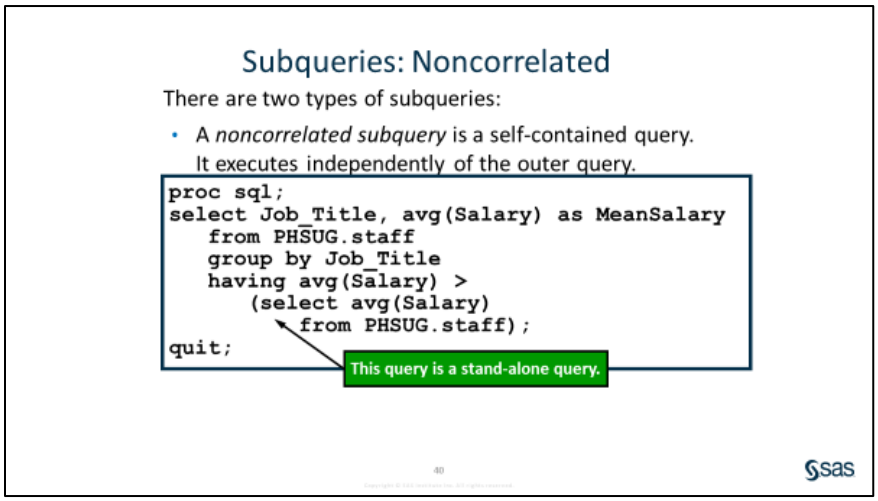
A subquery is a a query within a query. So it must make sense, if we look at the statements in SQL that it would sit on the WHERE or HAVING clause.



Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

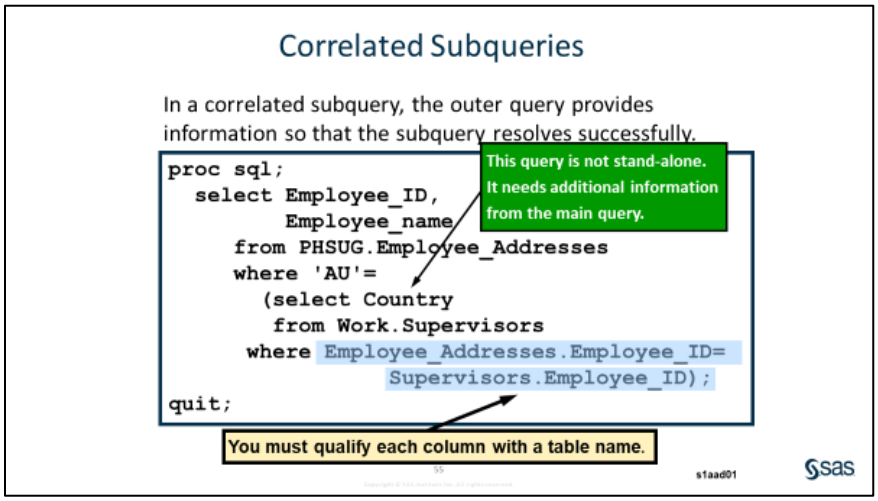
Here is the big difference between the two:

In a non-correlated subquery, the inner query or sub-query is independent and self-contained. It passes information to the outer query and doesn't depend on the outer query for data.



In a correlated subquery, the outer query passes information to the inner query or subquery. The subquery cannot resolve without the data provided by the outer query.

Select 1st row from supervisors and search the entire outer table for a match, if we don't get a match we did the work for nothing. So We will keep on doing this: reading the outer table each time for every single row in the inner query. If I had a million rows in my inner query, outer table gets read 1 million times. When it comes to processing data, slow is universally bad. So avoid correlated as much as you can.



Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

CONFESSION # 4 – WHERE ANSI FALLS SHORT AND PROC SQL STEPS IN

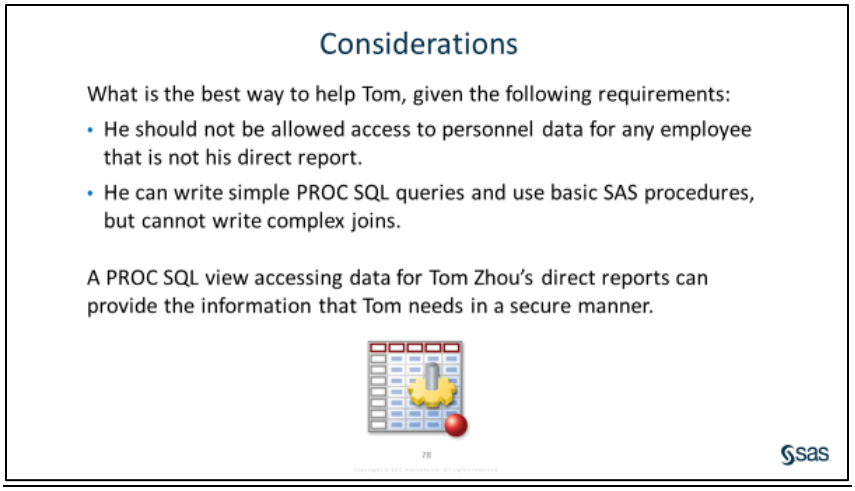
Making a view portable

Like me, you have probably wondered about the many different installations of SQL and how they could help. Here is a great confession: PROC SQL actually comes to the rescue where ANSI falls short.

Are you ready for my next revelation?

This technique has to do with portability of a view. ANSI SQL expects the view and table associated with the view to be co-located or stored in the same location. Many users prefer to store different objects in different locations. If you feel hampered by this ANSI expectation, then this confession may be just what you are looking for.

Data that Tom needs comes from 3 tables: employee_addresses, employee_payroll, employee_organization



Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Code to build view

```
proc sql;
create view PHSUG.tom_zhou as
   select Employee_Name as Name format=$25.0,
          Job_Title as Title format=$15.0,
          Salary 'Annual Salary' format=comma10.2,
          int((today()-Employee_Hire_Date)/365.25)
             as YOS 'Years of Service'
      from employee_addresses as a,
           employee_payroll as p,
           employee_organization as o
      where a.Employee_ID=p.Employee_ID and
            o.Employee_ID=p.Employee_ID and
            Manager_ID=120102;
quit;
```

Code to run view

```
title "Tom Zhou's Direct Reports";
title2 "By Title and Years of Service";
select *
   from PHSUG.tom_zhou
   order by Title desc, YOS desc;
```

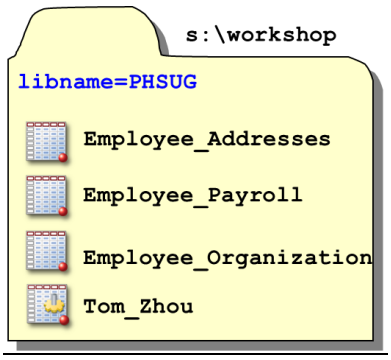
Output

Tom Zhou's Direct Reports
By Title and Years of Service

Name	Title	Annual Salary	Years of Service
Nowd, Fadi	Sales Rep. IV	30,660.00	40
Hofmeister, Fong	Sales Rep. IV	32,040.00	35
Phoumirath, Lynelle	Sales Rep. IV	30,765.00	28
Platts, Alexei	Sales Rep. IV	32,490.00	16
Kletschkus, Monica	Sales Rep. IV	30,890.00	7
Hayawardhana, Caterina	Sales Rep. III	30,490.00	40
Comber, Edwin	Sales Rep. III	28,345.00	40
Kaiser, Fancine	Sales Rep. III	28,525.00	35

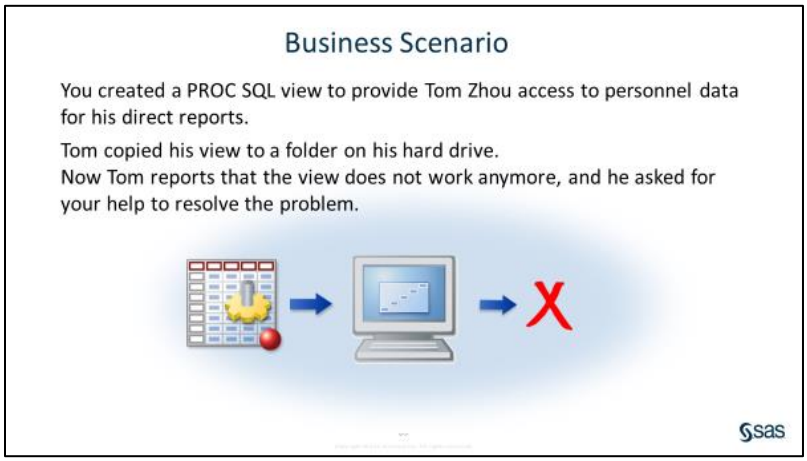
Ansi standards

ANSI standards specify that the view must reside in the same library as the contributing tables.



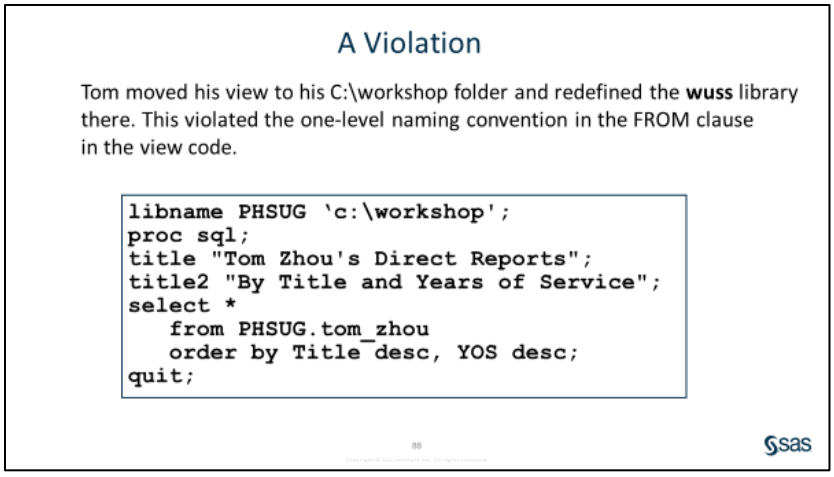
Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

This becomes a problem when Tom like most users wishes to move the view to his personal folder.



Code submitted that failed

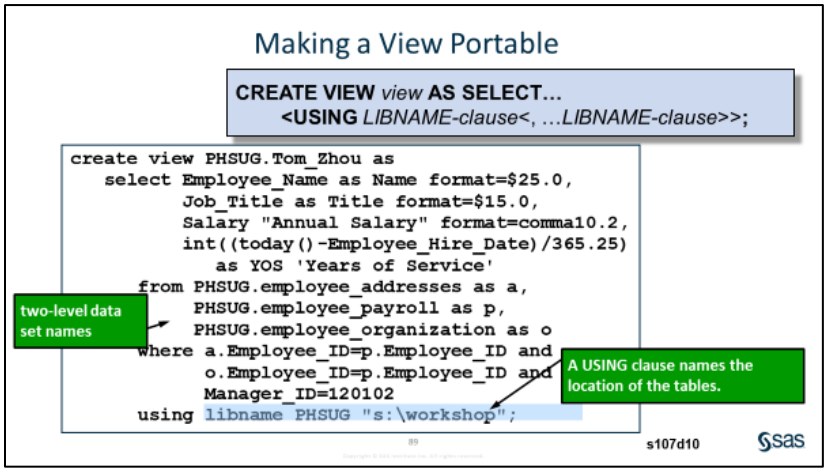
```
libname PHSUG 'c:\workshop';
NOTE: Libref PHSUG was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\workshop
   proc sql;
   title "Tom Zhou's Direct Reports";
   title2 "By Title and Years of Service";
   select *
      from PHSUG.tom_zhou
      order by Title desc, YOS desc;
ERROR: File PHSUG.EMPLOYEE_ADDRESSES.DATA does not exist.
ERROR: File PHSUG.EMPLOYEE_PAYROLL.DATA does not exist.
ERROR: File PHSUG.EMPLOYEE_ORGANIZATION.DATA does not exist.
   quit;
   title;
NOTE: The SAS System stopped processing this step because of errors.
```



Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Making the view portable with PROC SQL

Use the Using clause to make the view portable.



The view Works and it can be saved on a separate location thus making it truly portable.

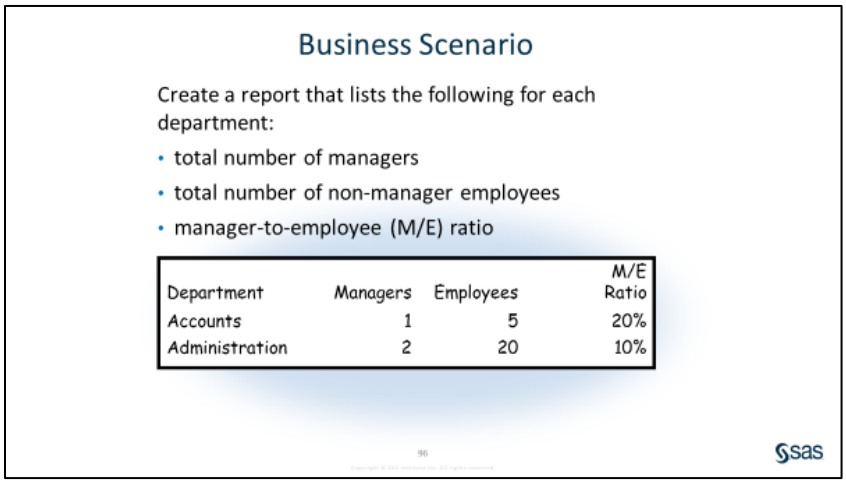
CONFESSION # 5 – SUMMARIZING DATA USING THE BOOLEAN GATE

Hands-down, summarizing data using the Boolean gate in PROC SQL has to be my all-time favorite technique. When I fell in love with its elegance, I captioned my blog captioned "No 1 Best programming technique for 2012". It was easily my #1 best technique for life, but I thought I would keep myself open to new learning! Read on, and you might forgive me for this confession.

Summarizing data

The Boolean is simply the digital computing world's way of converting everything to 0s and 1s. A yes, is a one and a no a zero.

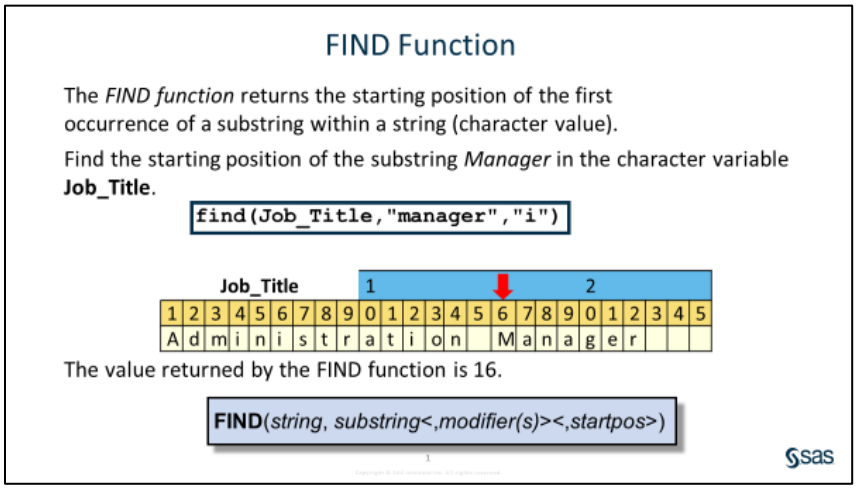
Have you ever been challenged with a business scenario where you had to subset data to return both the haves and the have nots?



Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

How will you go about extracting both the managers and the employees and stick them all on the same line? Never fear, this is what my last confession is all about.

First, we will use the FIND function to find all managers.



Here is the classic Boolean put to good use to determine whether an employee is a manager. If job_title contains Manager, the value is 1 and if it doesn't contain Manager, the value is 0.

Code to write a Boolean expression

```
proc sql;
select Department, Job_Title,
       (find(Job_Title,"manager","i")>0)
       "Manager"
   from PHSUG.employee_information;
quit;
```

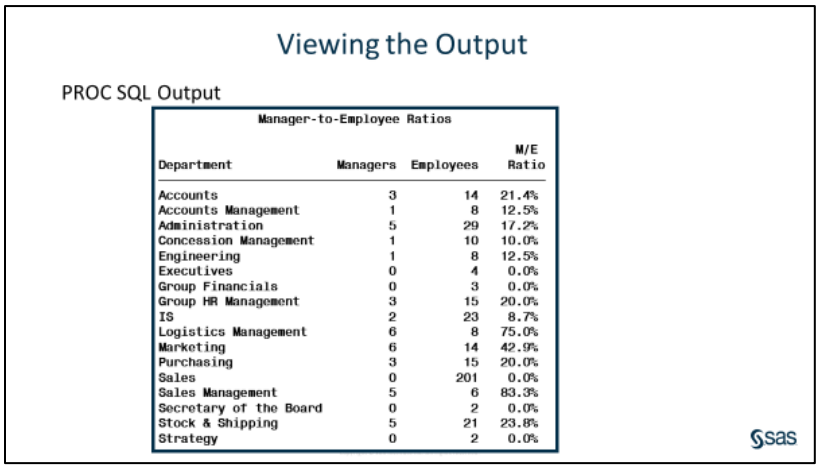
Now simply calculate the statistics by wrapping the Boolean expressions with the Sum function.

Code to summarize data using the boolean

```
proc sql;
title "Manager-to-Employee Ratios";
select Department,
       sum((find(Job_Title,"manager","i")>0))
         as Managers,
       sum((find(Job_Title,"manager","i")=0))
         as Employees,
       calculated Managers/calculated Employees
         "M/E Ratio" format=percent8.1
   from PHSUG.employee_information
   group by Department;
quit;
```

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Output



Isn't that a confession worth waiting for? You can use the Boolean in many expressions and make the expressions as complex as need be.

1. REFERENCES

SAS 9.4 Proc sql user's guide
https://go.documentation.sas.com/?docsetId=sqlproc&docsetTarget=titlepage.htm&docsetVersion=9.4&locale=en

Proc sql syntax order
Go home on time with these 5 PROC SQL tips. Shankar, Charu
https://blogs.sas.com/content/sastraining/2012/04/24/go-home-on-time-with-these-5-proc-sql-tips/

"Working with Subquery in the SQL procedure" Zhang, Lei. Yi, Danbo
https://www.lexjansen.com/nesug/nesug98/dbas/p005.pdf

"Proc sql views"
https://go.documentation.sas.com/?docsetId=sqlproc&docsetTarget=n0nolnbokay91in1gouzgw3xzl5e.htm&docsetVersion=9.4&locale=en

Boolean in SQL.
#"1 best programming tip for 2012". Shankar, Charu
https://blogs.sas.com/content/sastraining/2012/05/10/1-sas-programming-tip-for-2012/

ACKNOWLEDGMENTS

The author is grateful to the many SAS users that have entered her life. Charu is grateful to the Pharmasug committee for the opportunity to present this paper. She would also like to express her gratitude to her manager, Stephen Keelan without whose support and permission, this paper would not be possible.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

CONTACT INFORMATION

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

Charu Shankar
SAS Institute Canada, Inc.
Charu.shankar@sas.com
https://blogs.sas.com/content/author/charushankar/

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

Copyright © 2016, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.