

# Confessions Of A Proc SQL Instructor

MSUG, 12 June 2024

Charu Shankar  
SAS Education

Copyright © SAS Institute Inc. All rights reserved.



## Charu Shankar, SAS® Institute

---



With a background in computer systems management. SAS Instructor Charu Shankar engages with logic, visuals, and analogies to spark critical thinking since 2007.

Charu curates and delivers unique content on SAS, SQL, Viya, etc. to support users in the adoption of SAS software.

When not coding, Charu teaches yoga and loves to explore Canadian trails with her husky Miko.



# Agenda



Confession 1: PROC SQL Syntax Order Mnemonic



Confession 2: Know thy data : Dictionary tables



Confession 3: Stack data horizontally



Confession 4: Where ANSI SQL falls short and PROC SQL steps in



Confession 5: Summarizing data using the Boolean Gate



Handy Links

# Confession 1: PROC SQL Syntax Order Mnemonic

## Overview of the SQL Procedure

4

# SQL Procedure

The SQL procedure is initiated with a PROC SQL statement. It is terminated with a QUIT statement.

```
proc sql;  
  select Employee_ID, Employee_Gender,  
         Salary  
  from msug.employee_information;  
quit;
```

```
PROC SQL <option(s)>;  
statement(s);  
QUIT;
```

# SQL Procedure

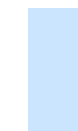
- Multiple statements can be included in a PROC SQL step.
- Each statement defines a process and is executed immediately.

```
PROCSQL <option(s)>;  
statement(s);  
QUIT;
```

# SELECT Statement

A *SELECT statement* is used to query one or more tables. The results of the SELECT statement are written to the default output destination.

```
proc sql;  
select Employee_ID, Employee_Gender, Salary  
  from msug.employee_information  
 where Employee_Gender='F'  
 order by Salary desc;  
quit;
```



# SELECT Statement

A SELECT statement contains smaller building blocks called *clauses*

```
proc sql;  
select Employee_ID, Employee_Gender, Salary  
from msug.employee_information  
where Employee_Gender='F'  
order by Salary desc;  
quit;
```

clauses

**Note:** Although it can contain multiple clauses, each SELECT statement begins with the SELECT keyword and ends with a semicolon



# Viewing the Output

## Partial PROC SQL Output

The SAS System		
Employee ID	Employee Gender	Employee Annual Salary
120260	F	\$207,885
120719	F	\$87,420
120661	F	\$85,495
121144	F	\$83,505
120798	F	\$80,755

# SELECT Statement: Required Clauses

```
SELECT object-item <, ...object-item>  
FROM from-list;
```

Here are two things that SQL always needs:

1. What do you want?  
The SELECT clause specifies the columns and column order.
2. Where do you want it from?  
The FROM clause specifies the data sources.  
You can query from 1 to 256 tables.

# SELECT Statement: Syntax Order

SO  
FEW  
WORKERS  
GO  
HOME  
ON TIME

```
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>>;
```

- 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.

# Discussion

```
proc sql;  
select Employee_ID, Employee_Gender,  
       Salary  
  from msug.employee_information  
 order by Employee_ID  
where Employee_Gender='M';  
quit;
```

Is this code correct?

# Syntax Check with the NOEXEC Option

To explicitly check for syntax errors without submitting the code for execution, include the NOEXEC option in the PROC SQL statement. This option applies to all statements in a PROC SQL step.

**PROC SQL <NOEXEC>;**

```
proc sql noexec;  
select Employee_ID, Employee_Gender, Salary  
       from msug.employee_information  
       where Employee_Gender='M'  
       order by Salary desc;  
quit;
```

# Viewing the Log

## Partial SAS Log

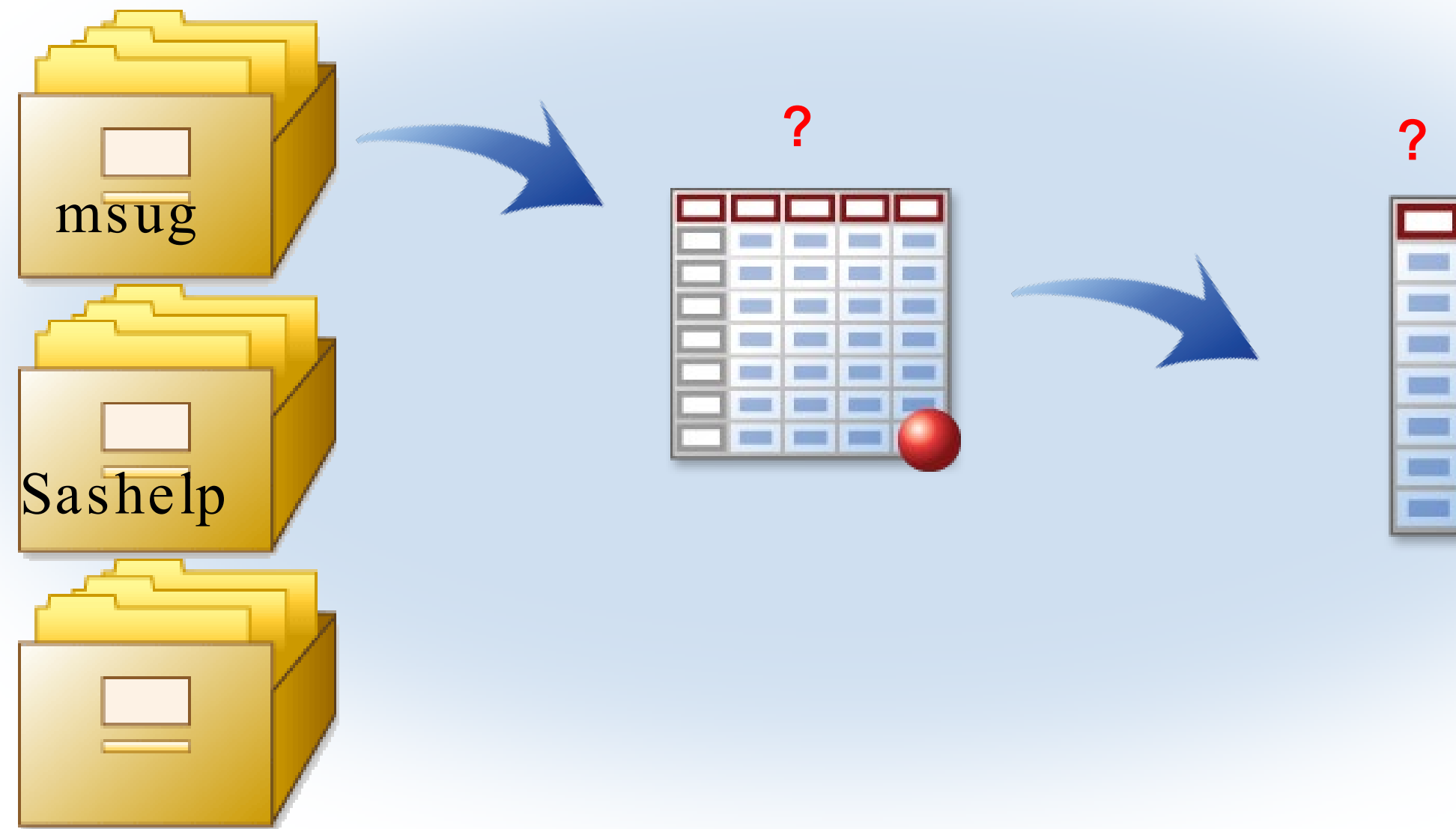
```
proc sql noexec;  
select Employee_ID, Employee_Gender, Salary  
       from msug.employee_information  
       where Employee_Gender='M'  
       order by Salary desc;  
NOTE: Statement not executed due to NOEXEC option.  
quit;
```

# Confession 2. Know Thy Data:

Dictionary Tables & Views

# Business Scenario

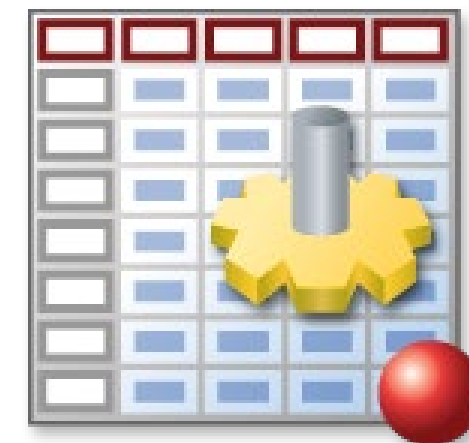
You have inherited many different data tables and want to become familiar with their content.





# DICTIONARY Tables: Overview

- *DICTIONARY tables* are Read-Only metadata views that contain session metadata, such as information about SAS libraries, data sets, and external files in use or available in the current SAS session.
- DICTIONARY tables are
  - created at SAS session initialization
  - updated automatically by SAS
  - limited to Read-Only access.
- You can query DICTIONARY tables with PROC SQL.



# Querying Metadata about SAS Libraries

There can be more than 30 DICTIONARY tables. We will focus 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

# Exploring DICTIONARY Tables

You can use a DESCRIBE statement to explore the structure of DICTIONARY tables:

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

Partial 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', ...);
```

# Querying Dictionary Information

Display information about the tables in the SASHELP library.

```
title 'Tables in the SASHELP Library';  
proc sql;  
select memname 'Table Name',  
       nobs,nvar,crdate  
  from dictionary.tables  
 where libname='SASHELP';  
quit;
```

Library names are  
stored in uppercase  
in DICTIONARY tables.

# Viewing the Output

## Partial PROC SQL Output

### *Tables in the SASHELP Library*

Table Name	Number of Physical Observations	Number of Variables	Date Created
AACOMP	2020	4	25JUN15:01:05:47
AARFM	61	4	25JUN15:01:07:08
ADSMMSG	426	6	25JUN15:01:09:46
AFMSG	1090	6	25JUN15:01:06:18
AIR	144	2	25JUN15:01:12:52
APPLIANC	156	25	25JUN15:01:12:54
ASSCMGR	402	19	25JUN15:01:19:20
AUTHLIB	4	7	25JUN15:01:24:40

# Querying Dictionary Information

Display information about the columns in **sashelp.cars**

```
title 'Columns in the sashelp.cars Table';  
proc sql;  
select Name, Type, Length  
       from dictionary.columns  
       where libname='SASHELP'  
              and memname='CARS';  
quit;
```

Table names (*memnames*)  
are also stored in uppercase  
in DICTIONARY tables.

# Viewing the Output

## PROC SQL Output

### Columns in the sashelp.cars Table

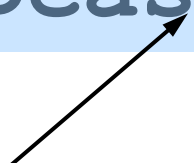
Column Name	Column Type	Column Length
Make	char	13
Model	char	40
Type	char	8
Origin	char	6
DriveTrain	char	5
MSRP	num	8
Invoice	num	8
EngineSize	num	8
Cylinders	num	8
Horsepower	num	8
MPG_City	num	8
MPG_Highway	num	8
Weight	num	8
Wheelbase	num	8
Length	num	8

Column names are stored in mixed case

# Using Dictionary Information

Which tables contain an ID column?

```
title 'Tables Containing an ID Column';  
proc sql;  
select memname 'Table Names', name  
      from dictionary.columns  
      where libname='SASHELP' and  
            upcase(name) contains 'ID';  
quit;
```



Because different tables might use different cases for same-named columns, you can use the UPCASE function for comparisons. However, this significantly degrades the performance of the query.



# Viewing the Output

## Tables Containing an ID Column

Table Names	Column Name
ADSMMSG	MSGID
AFMSG	MSGID
ASSCMGR	ID
BURROWS	ID
CLNMSG	MSGID
COLUMN	TABLEID
COLUMN	ID
DEMOGRAPHICS	ID
DFTDICT	ID
DYNATTR	SOURCEID
DYNATTR	ID
EISMKCN	ID

All ID column names are stored in uniform uppercase, so the UPCASE function is not needed the next time that a query such as this is executed.

# Finding Common Column Names Dynamically

All of the previous 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.

Use the following code

```
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;
```

# Viewing the Output

## Common columns in SASHELP

Column Name	Member Name	Column Type	Column Length
ACTUAL	PRDSAL2	num	8
ACTUAL	PRDSAL3	num	8
ACTUAL	PRDSALE	num	8
ALIAS_CITY	ZIPCODE	char	300
ALIAS_CITY	ZIPMIL	char	300
ALIAS_CITYN	ZIPCODE	char	300
ALIAS_CITYN	ZIPMIL	char	300
AMOUNT	BUY	num	8
AMOUNT	NVST1	num	8
AMOUNT	NVST2	num	8
AMOUNT	NVST3	num	8
AMOUNT	NVST4	num	8
AMOUNT	NVST5	num	8
AMOUNT	RENT	num	8
AMOUNT	ROCKPIT	num	8

Joins are easier because the structure of each table does not have to be examined before determining common columns. Let SAS bring common columns dynamically by looking up DICTONARY tables.

# Using DICTIONARY Tables in Other SAS Code

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**

The following code executes successfully:

```
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?

```
options fulltimer;  
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



# An Efficiency Question: PROC SQL or PRINT?

What do these statistics mean?

Statistic	Description
Real Time	The amount of real time (clock time) spent to process the SAS job. Real time is also referred to as <i>elapsed time</i> .
User CPU Time	The CPU time that is spent in the user program.
System CPU Time	CPU time is spent to perform operating system tasks (system overhead tasks) that support the execution of your SAS code.
Memory	The amount of memory required to run a step.
OS Memory	the largest amount of operating system memory that is available to SAS during the step.
Timestamp	The date and time that a step was executed.
Step Count	Count of DATA steps or procedures that run in a SAS program.
Switch Count	A count of task switches within a step—that is, within a DATA step or procedure—in a SAS program. A task switch occurs when a step requests service from another process. Another task switch occurs when the step resumes. The number reported is for the last step that runs.

# An Efficiency Question: PROC SQL or PRINT?

Can I use PROC PRINT instead?

```
options fulltimer;  
proc print data=sashelp.vcolumn;  
    var libname memname name type length;  
    where upcase(name) contains 'ID' and  
libname='SASHELP' and type='num';  
run;
```

NOTE: There were 34 observations read from the data set SASHELP.VCOLUMN.  
WHERE UPCASE(name) contains 'ID' and (libname='SASHELP') and  
(type='num');

NOTE: PROCEDURE PRINT used (Total process time):

real time	2.19 seconds
user cpu time	0.92 seconds
system cpu time	1.18 seconds
memory	6738.81k
OS Memory	25440.00k



# Confession 3. Stack Data Horizontally

Subqueries: Best Practices, Dangers of Correlated



# Subqueries

A subquery

- returns values to be used in the outer query's WHERE or HAVING clause

```
SELECT ...>  
FROM ...  
<WHERE ...>  
  
<GROUP BY ...>  
<HAVING ...>  
  
<ORDER BY ...>;
```

```
... (select Employee_ID  
from msug.staff  
where ... ) ...
```

# Subqueries: Noncorrelated

There are two types of subqueries:

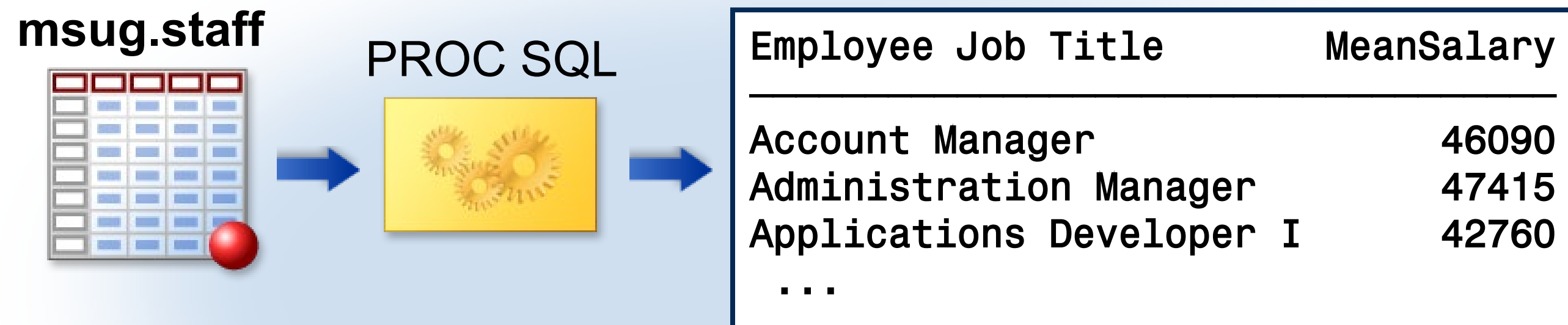
- A *noncorrelated subquery* is a self-contained query. It executes independently of the outer query.

```
proc sql;  
select Job_Title, avg(Salary) as MeanSalary  
from msug.staff  
group by Job_Title  
having avg(Salary) >  
      (select avg(Salary)  
       from msug.staff);  
quit;
```

This query is a standalone query.

# Business Scenario

HR and Payroll managers requested a report that displays **Job\_Title** for job groups with an average salary greater than the average salary of the company as a whole.



# Step 1

Calculate the company's average salary.

```
proc sql;  
select avg(Salary) as CompanyMeanSalary  
  from msug.staff;  
quit;
```

Company MeanSalary
38041.51

## Step 2

Determine the job titles whose average salary exceeds the company's average salary.

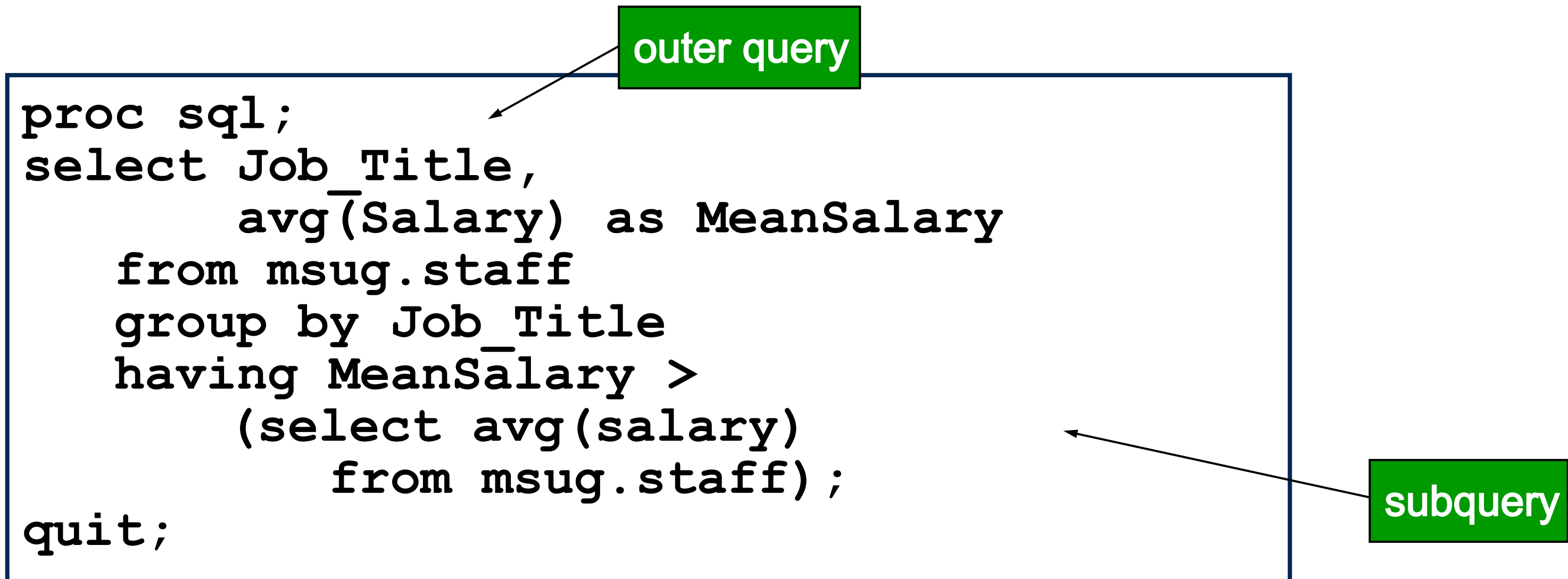
```
proc sql;  
select Job_Title,  
       avg(Salary) as MeanSalary  
from msug.staff  
group by Job_Title  
having MeanSalary>38041.51;  
quit;
```

Partial PROC SQL Output

Employee Job Title	MeanSalary
Account Manager	46090
Administration Manager	47415
Applications Developer I	42760

## Step 3

- Write the program as a single step using a subquery.
- A *subquery* is a query that resides within an outer query.



**Note:** *The subquery must be resolved before the outer query can be resolved*

# Business Scenario

Each month, the CEO sends a birthday card to each employee having a birthday in that month. Create a report listing the names and addresses of employees with February birthdays.



# Noncorrelated Subqueries

The **msug.Employee\_Addresses** table contains names and addresses. Birth dates are found in the **msug.Employee\_Payroll** table.

Write a query to grab the February born staff.

```
proc sql;  
select Employee_ID  
       from msug.Employee_Payroll  
       where month(Birth_Date)=2;  
quit;
```



# Noncorrelated Subqueries

Pass the query with the February borns as a subquery.

```
proc sql;  
    select Employee_ID,  
           Employee_Name, City,  
           Country  
    from msug.Employee_Addresses  
   where Employee_ID in  
          (select Employee_ID  
           from msug.Employee_Payroll  
          where month(Birth_Date)=2)  
    order by 1;  
quit;
```

# Noncorrelated Subqueries: How Do They Work?

```
proc sql;
  select Employee_ID,
         Employee_Name, City,
         Country
  from msug.Employee_Addresses
  where Employee_ID in
        (select Employee_ID
         from msug.Employee_Payroll
         where month(Birth_Date)=2)
  order by 1;
quit;
```

Partial  
msug.Employee\_Payroll

Employee_ID	Birth_Date
...	...
120106	23DEC1948
120107	21JAN1953
120108	23FEB1988
120109	15DEC1990
120110	20NOV1953
120111	23JUL1953
120112	17FEB1973
120113	10MAY1948
...	...

**Step 1: Evaluate the inner query and build a virtual table that satisfies the WHERE criteria.**

# Noncorrelated Subqueries: How Do They Work?

```
proc sql;  
  select Employee_ID,  
         Employee_Name, City,  
         Country  
  from msug.Employee_Addresses  
 where Employee_ID in  
        (120108,120112,120114,120157,  
         120159, 120170,...)  
 order by 1;  
quit;
```

**Partial**  
msug.Employee\_Payroll

Employee_ID	Birth_Date
...	...
120106	23DEC1948
120107	21JAN1953
<b>120108</b>	<b>23FEB1988</b>
120109	15DEC1990
120110	20NOV1953
120111	23JUL1953
<b>120112</b>	<b>17FEB1973</b>
120113	10MAY1948
...	...

Values returned by the inner query

# Noncorrelated Subqueries: How Do They Work?

```
proc sql;  
  select Employee_ID,  
         Employee_Name, City,  
         Country  
  from msug.Employee_Addresses  
 where Employee_ID in  
        (120108,120112,120114,120157,  
         120159, 120170,...)  
 order by 1;  
quit;
```

## Partial

msug.Employee\_Payroll

Employee_ID	Birth_Date
...	...
120106	23DEC1948
120107	21JAN1953
120108	23FEB1988
120109	15DEC1990
120110	20NOV1953
120111	23JUL1953
120112	17FEB1973
120113	10MAY1948
...	...

**Step 2: Pass the values to the outer query for use in the WHERE clause.**

# Noncorrelated Subqueries: Output

The SAS System

Employee_ID	Employee_Name	City	Country
120108	Gromek, Gladys	Melbourne	AU
120112	Glattback, Ellis	Melbourne	AU
120114	Buddery, Jeannette	Sydney	AU
120157	Karavdic, Leonid	Sydney	AU
120159	Phoumirath, Lynelle	Sydney	AU
120170	Kingston, Alban	Sydney	AU

Do these look familiar?  
They are the employee IDs  
returned by the inner query.

# Correlated Subqueries

- Correlated subqueries
  - cannot be evaluated independently
  - require values to be passed to the inner query from the outer query
  - are evaluated for each row in the outer query.

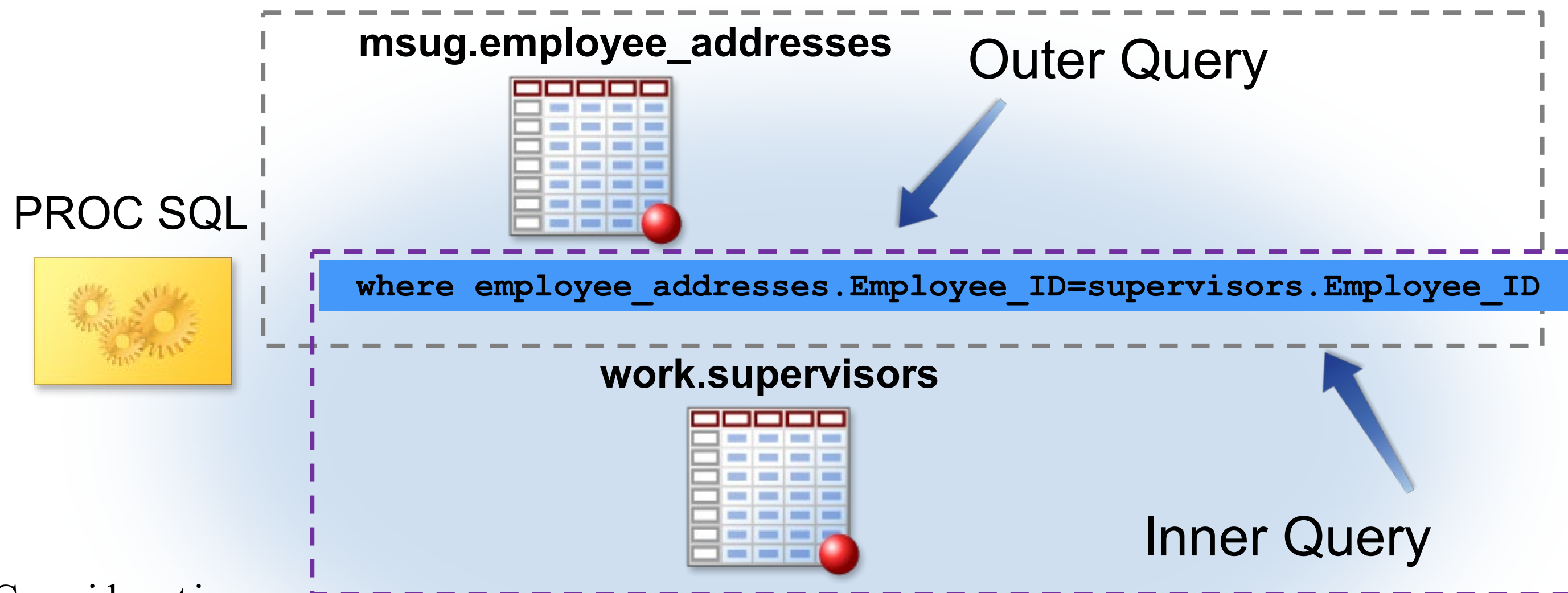
# Subqueries: Correlated

A *correlated subquery* requires a value or values to be passed to it by the outer (main) query before it can be successfully resolved.

```
proc sql;  
select Employee_ID, avg(Salary) as MeanSalary  
  from msug.employee_addresses  
 where 'AU'=  
       (select Country  
        from work.supervisors  
        where employee_addresses.Employee_ID=  
              supervisors.Employee_ID) ;  
quit;
```

# Business Scenario

Use a correlated subquery to create a report listing the employee identifier and name for all managers in Australia.



Considerations:

- You have a temporary table, **Supervisors**, containing **Employee\_ID** and **Country** for all managers.
- The table **msug.Employee\_Addresses** contains **Employee\_Name** for all employees



# Correlated Subqueries

In a correlated subquery, the outer query provides information so that the subquery resolves successfully.

```
proc sql;  
  select Employee_ID,  
         Employee_name  
  from msug.Employee_Addresses  
 where 'AU'=  
       (select Country  
        from Work.Supervisors  
 where Employee_Addresses.Employee_ID=  
        Supervisors.Employee_ID) ;  
quit;
```

This query is not stand-alone.  
It needs additional information  
from the main query.

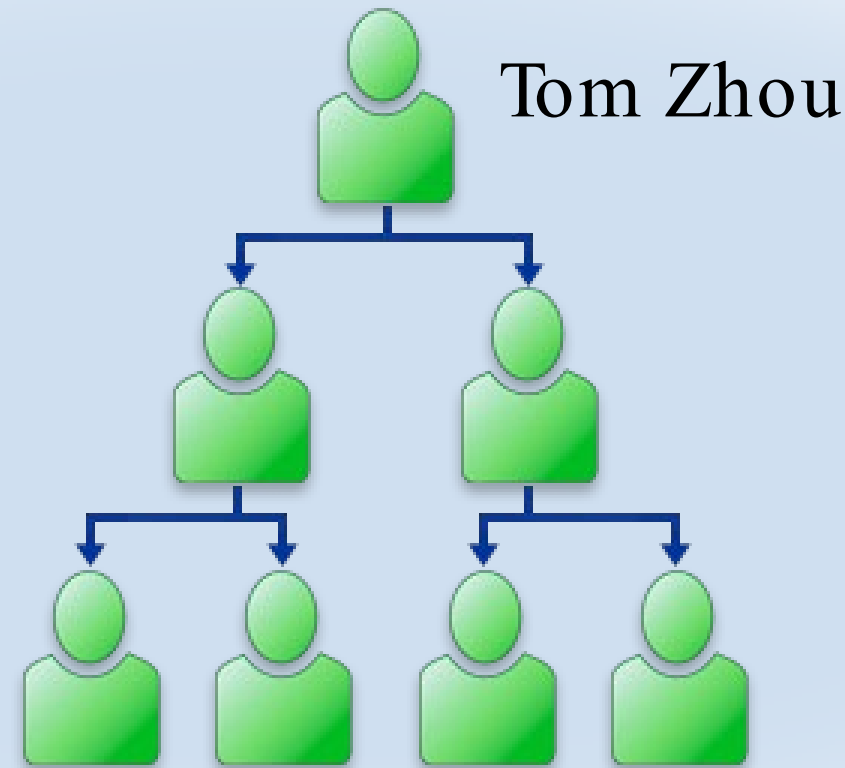
You must qualify each column with a table name.

# Confession 4. Where ANSI falls short and PROC SQL steps in

Making a view portable

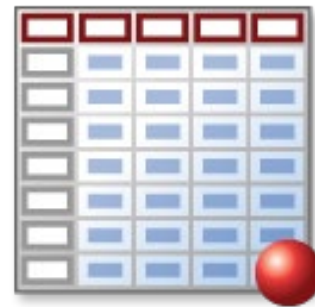
# Business Scenario

Tom Zhou is a sales manager who needs access to personnel information for his staff

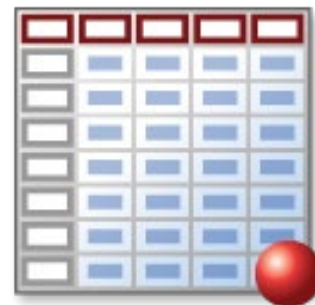


# Business Data

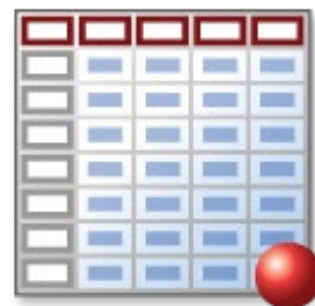
The data that Tom needs is name, job title, salary, and years of service. Data is contained in three tables.



**msug.employee\_addresses**



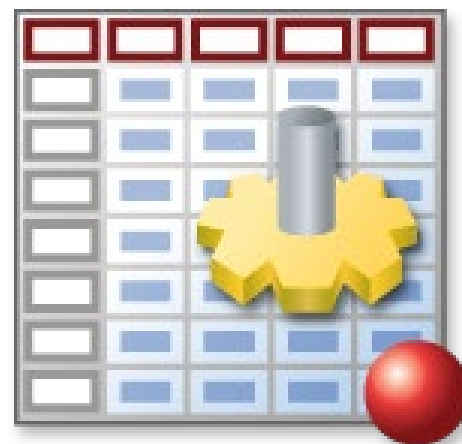
**msug.employee\_payroll**



**msug.employee\_organization**

# Considerations

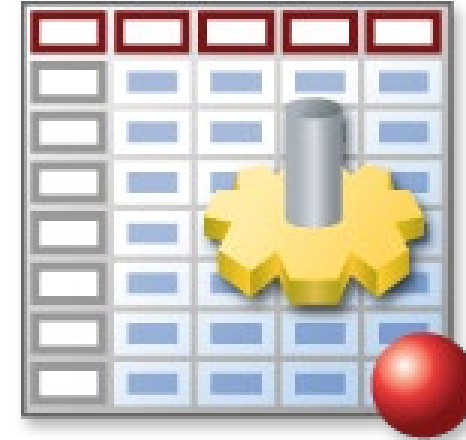
- What is the best way to help Tom, given the following requirements:
  - He should not be allowed access to personnel data for any employee that is not his direct report.
  - He can write simple PROC SQL queries and use basic SAS procedures, but cannot write complex joins.
- A PROC SQL view accessing data for Tom Zhou's direct reports can provide the information that Tom needs in a secure manner



# What Is a PROC SQL View?

## *A PROC SQL view*

- is a stored query
- contains no actual data
- can be derived from one or more tables, PROC SQL views, DATA step views, or SAS/ACCESS views
- extracts underlying data each time it is used and accesses the most current data
- can be referenced in SAS programs in the same way as a data table
- cannot have the same name as a data table stored in the same SAS library.



# Creating a PROC SQL View

To create a PROC SQL view, use the CREATE VIEW statement.

```
CREATE VIEW view-name AS  
SELECT ...;
```

```
proc sql;  
create view msug.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;
```

# View the Log

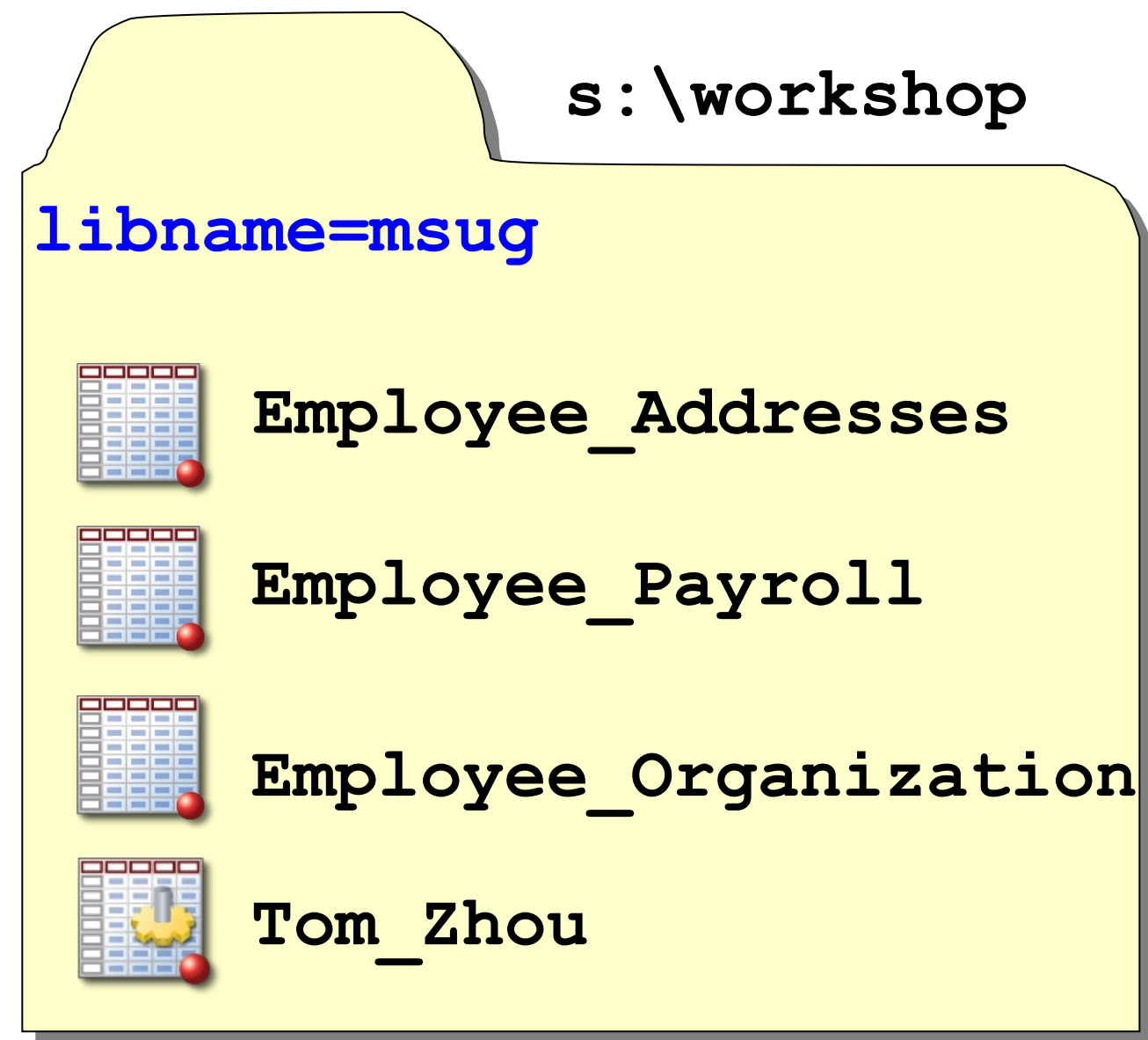
## Partial SAS Log

```
5      proc sql;
46     create view msug.tom_zhou as
47       select Employee_Name as Name format=$25.0,
48              Job_Title as Title format=$15.0,
49              Salary 'Annual Salary' format=comma10.2,
50              int((today()-Employee_Hire_Date)/365.25)
51              as YOS 'Years of Service'
52       from employee_addresses as a,
53            employee_payroll as p,
54            employee_organization as o
55       where a.Employee_ID=p.Employee_ID and
56            o.Employee_ID=p.Employee_ID and
57            Manager_ID=120102;
NOTE:  SQL view msug.TOM_ZHOU has been defined.
```



# Location of a PROC SQL View

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



# Location of the Source Tables: ANSI

In PROC SQL, the default libref for the table (or tables) in the FROM clause is the libref of the library that contains the view. When the view and data source are in the same location, you specify a one-level name for the table (or tables) in the FROM clause.

```
create view msug.tom_zhou as
...
from employee_addresses as a,
     employee_payroll as p,
     employee_organization as o
```

# Using a View

Tom can use the view to produce simple reports.

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

Partial PROC SQL Output (executed 5June2024)

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	46
Hofmeister, Fong	Sales Rep. IV	32,040.00	41
Phoumirath, Lynelle	Sales Rep. IV	30,765.00	33
Platts, Alexei	Sales Rep. IV	32,490.00	22
Kletschkus, Monica	Sales Rep. IV	30,890.00	13
Comber, Edwin	Sales Rep. III	28,345.00	46
Hayawardhana, Caterina	Sales Rep. III	30,490.00	46
Kaiser, Fancine	Sales Rep. III	28,525.00	41
Roebuck, Alvin	Sales Rep. III	30,070.00	34
Blind, David	Sales Rep. III	28,685.00	33

# Business Scenario

- You created a PROC SQL view to provide Tom Zhou access to personnel data for his direct reports.
- Tom copied his view to a folder on his hard drive.  
Now Tom reports that the view does not work anymore, and he asked for your help to resolve the problem.



# Exploring the Problem

Tom submitted the following:

```
libname msug 'c:\temp';  
proc sql;  
title "Tom Zhou's Direct Reports";  
title2 "By Title and Years of Service";  
select *  
    from msug.tom_zhou  
    order by Title desc, YOS desc;  
quit;  
title;
```

# Viewing the Log

## Partial SAS Log

```
libname msug 'c:\workshop';
```

```
NOTE: Libref msug 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 msug.tom_zhou
```

```
    order by Title desc, YOS desc;
```

```
ERROR: File msug.EMPLOYEE_ADDRESSES.DATA does not exist.
```

```
ERROR: File msug.EMPLOYEE_PAYROLL.DATA does not exist.
```

```
ERROR: File msug.EMPLOYEE_ORGANIZATION.DATA does not exist.
```

```
quit;
```

```
title;
```

```
NOTE: The SAS System stopped processing this step because of errors.
```

# A Violation

Tom moved his view to his C:\workshop folder and redefined the **msug** library there. This violated the one-level naming convention in the FROM clause in the view code.

```
libname msug 'c:\workshop';  
proc sql;  
title "Tom Zhou's Direct Reports";  
title2 "By Title and Years of Service";  
select *  
    from msug.tom_zhou  
    order by Title desc, YOS desc;  
quit;
```

# Making a View Portable

```
CREATE VIEW view AS SELECT...  
<USING LIBNAME-clause<, ...LIBNAME-clause>>;
```

```
create view msug.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 msug.employee_addresses as a,  
       msug.employee_payroll as p,  
       msug.employee_organization as o  
 where a.Employee_ID=p.Employee_ID and  
       o.Employee_ID=p.Employee_ID and  
       Manager_ID=120102  
 using libname msug "s:\workshop";
```

two-level data  
set names

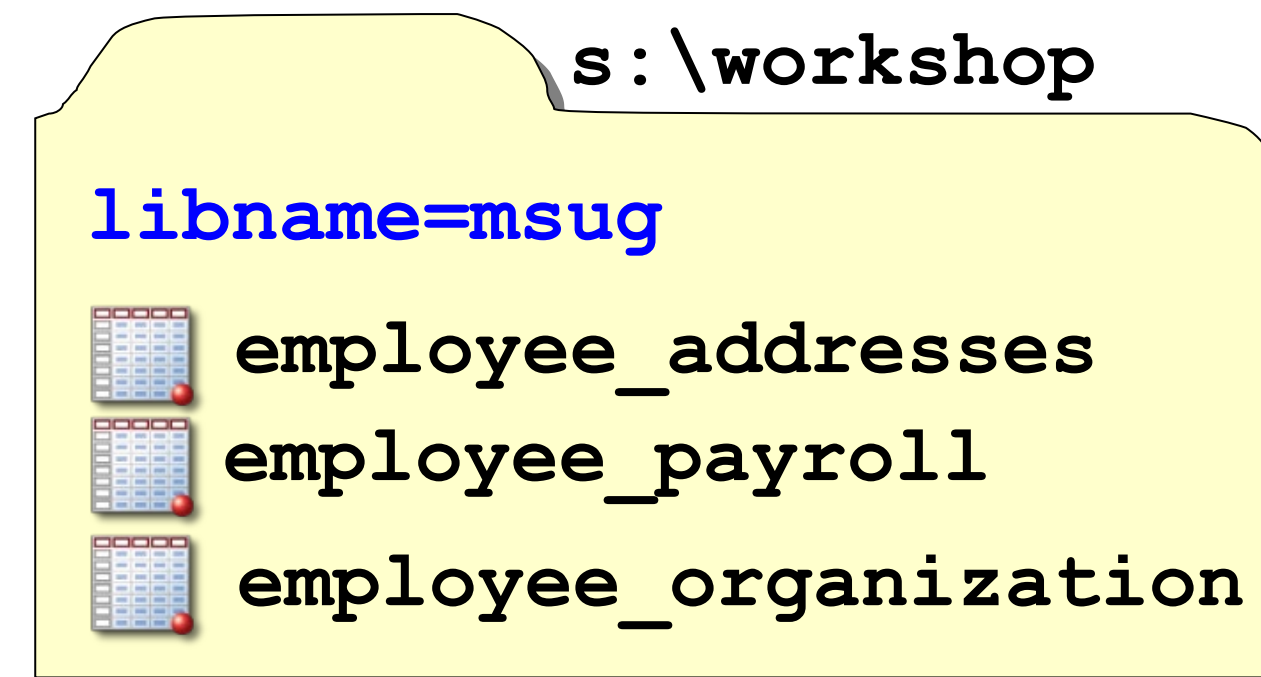
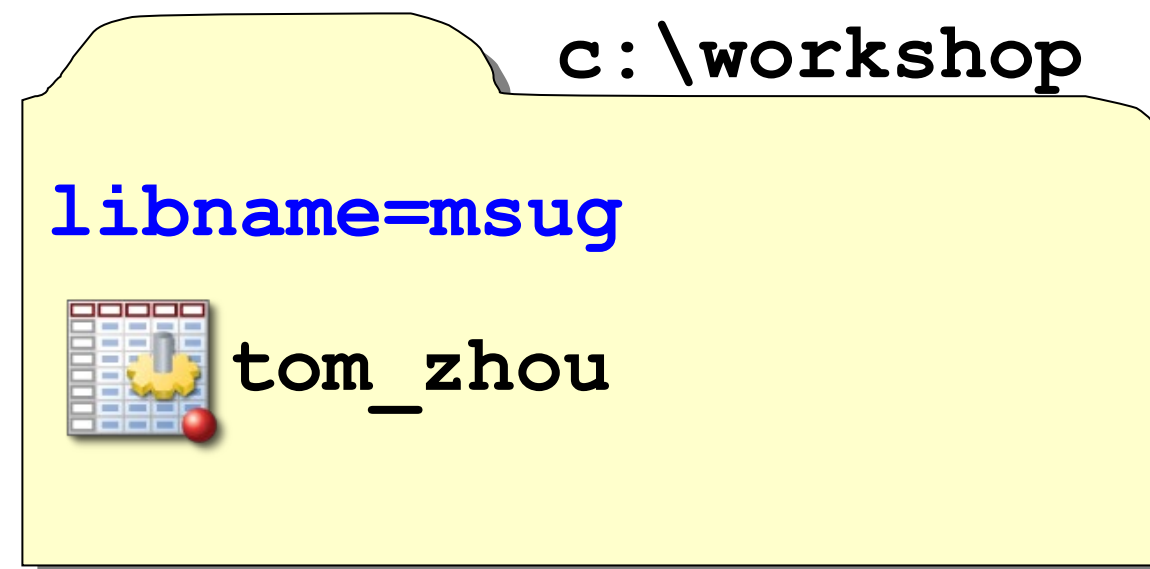
A USING clause names the  
location of the tables.



# Two-Level Table Names in Permanent Views

```
CREATE VIEW proc-sql-view AS SELECT ...  
  <USING LIBNAME-clause<, ...LIBNAME-clause>>;
```

- The USING clause libref is local to the view, and it will not conflict with an identically named libref in the SAS session.
- When the query finishes, the libref is disassociated.



# Views: Advantages

Advantages	Disadvantages
<ul style="list-style-type: none"><li>• avoid storing copies of large tables.</li><li>• avoid a frequent refresh of table copies. When the underlying data changes, a view surfaces the most current data.</li><li>• pull together data from multiple database tables and multiple libraries or databases.</li><li>• simplify complex queries.</li><li>• prevent other users from inadvertently altering the query code.</li></ul>	<ul style="list-style-type: none"><li>• Because views access the most current data in changing tables, the results might be different each time you access the view.</li><li>• Views can require significant resources each time they execute. With a view, you save disk storage space at the cost of extra CPU and memory usage.</li><li>• When accessing the same data several times in a program, use a table instead of a view. This ensures consistent results from one step to the next and can significantly reduce the resources that are required.</li></ul>

# Confession 5. Summarizing Data using the Boolean Gate

## Summarizing Data

# Business Scenario

Create a report that lists the following for each department:

- total number of managers
- total number of non-manager employees
- manager-to-employee (M/ E) ratio

Below is a rough sketch of the desired report.

Department	Managers	Employees	M/E Ratio
Accounts	1	5	20%
Administration	2	20	10%

# Business Data

Determine whether an employee is a manager  
or a non-manager

The **Job\_Title** column contains the information about each  
employee.

Department	Job_Title
Administration Manager	Administration
Administration Administration	Secretary I Office Assistant II

# Counting Rows That Meet a Specified Criterion

How do you determine the rows that *do* have *Manager* in **Job\_Title**, as well as rows that *do not*? You cannot use a WHERE clause to exclude either group.

Department	Job_Title
Administration	Administration Manager
Administration	Secretary I
Administration	Office Assistant II

Use the FIND function in a Boolean expression to identify rows that contain *Manager* in the **Job\_Title** column.

# FIND Function

The *FIND* function returns the starting position of the first occurrence of a substring within a string (character value).

Find the starting position of the substring *Manager* in the character variable **Job\_Title**.

```
find(Job_Title, "manager", "i")
```

Job_Title										1 2 3 4 5 6 7 8 9 0 1 2 3 4 5														
1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5
A	d	m	i	n	i	s	t	r	a	t	i	o	n		M	a	n	a	g	e	r			

The value returned by the FIND function is 16.

```
FIND(string, substring<,modifier(s)><, msugtpos>)
```

# Using Boolean Expressions

- Part 1: Use a Boolean expression to determine whether an employee is a manager

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

**Note:** Boolean expressions evaluate to true (1) or false (0).

- If **Job\_Title** contains *Manager*, the value is 1.
- If **Job\_Title** does not contain *Manager*, the value is 0.



# Viewing the Output

## Partial PROC SQL Output

Department	Job_Title	Manager
Administration	Administration Manager	1
Administration	Secretary I	0
Administration	Office Assistant II	0
Administration	Office Assistant III	0
Administration	Warehouse Assistant II	0
Administration	Warehouse Assistant I	0
Administration	Warehouse Assistant III	0
Administration	Security Guard II	0
Administration	Security Guard I	0
Administration	Security Guard II	0
Administration	Security Manager	1

# Using Boolean Expressions

Part 2: Calculate the statistics requested.

```
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 msug.employee_information  
group by Department;  
quit;
```

# Viewing the Output

## PROC SQL Output

Manager-to-Employee Ratios			
Department	Managers	Employees	M/E Ratio
Accounts	3	14	21.4%
Accounts Management	1	8	12.5%
Administration	5	29	17.2%
Concession Management	1	10	10.0%
Engineering	1	8	12.5%
Executives	0	4	0.0%
Group Financials	0	3	0.0%
Group HR Management	3	15	20.0%
IS	2	23	8.7%
Logistics Management	6	8	75.0%
Marketing	6	14	42.9%
Purchasing	3	15	20.0%
Sales	0	201	0.0%
Sales Management	5	6	83.3%
Secretary of the Board	0	2	0.0%
Stock & Shipping	5	21	23.8%
Strategy	0	2	0.0%

# Handy Links

[PROC SQL INTO Clause](#)

[SAS 9.4 PROC SQL User's Guide](#)

[The Power Of SAS SQL – SAS YouTube Video](#)

[SAS Tutorial | Step-By-Step PROC SQL – SAS YouTube Video](#)

[Working With Subquery In The SQL Procedure. Zhang, Lei. Yi, Danbo](#)

[Boolean In SQL. #”1 Best Programming Tip For 20 12” Shankar, Charu](#)

[Proc Sql Syntax Order: Go Home On Time With These 5 PROC SQL Tips. Shankar, Charu](#)

[“Shankar, Charu. “Know Thy Data: Techniques For Data Exploration” Pharmasug 20 18,](#)

[Ask the Expert Webinar - Why Choose Between SAS Data Step & PROC SQL When You Can Have Both](#)

# Charu Shankar

EMAIL [Charu.shankar@sas.com](mailto:Charu.shankar@sas.com)  
BLOG <https://blogs.sas.com/content/author/charushankar/>  
TWITTER [CharuYogaCan](#)  
LINKEDIN <https://www.linkedin.com/in/charushankar/>

✓ Enjoy this session, Let us know in the [evaluation](#)

