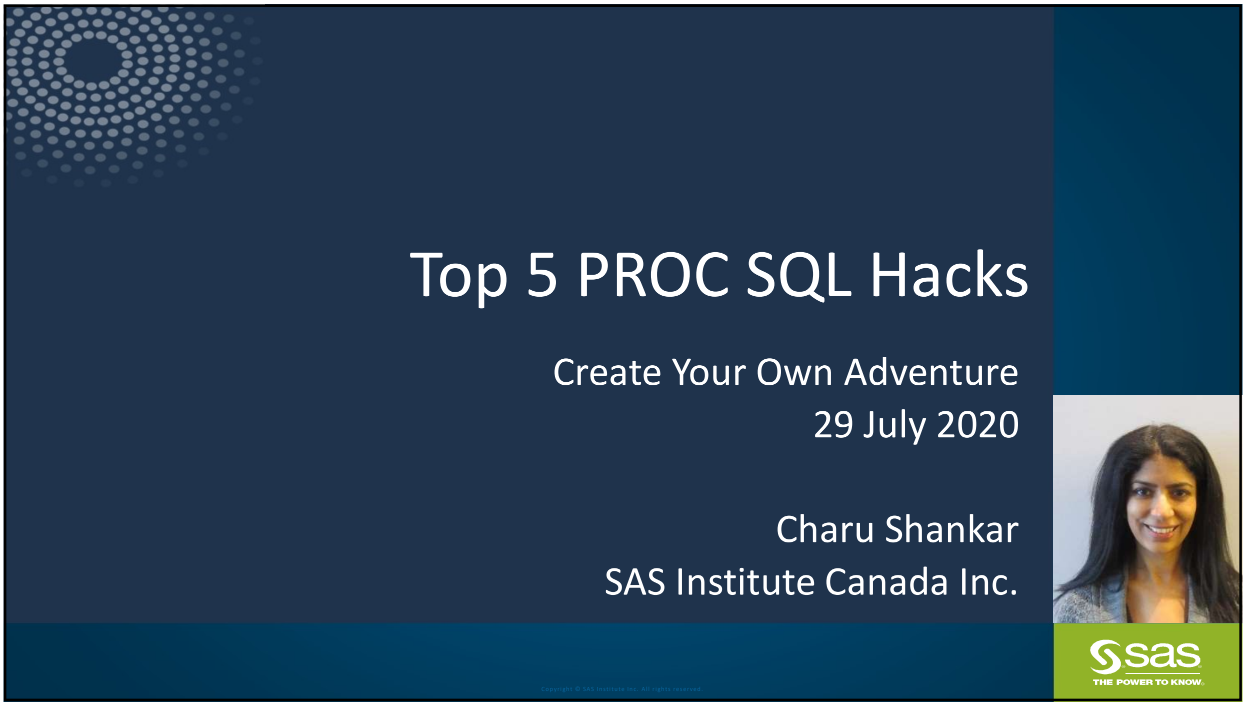
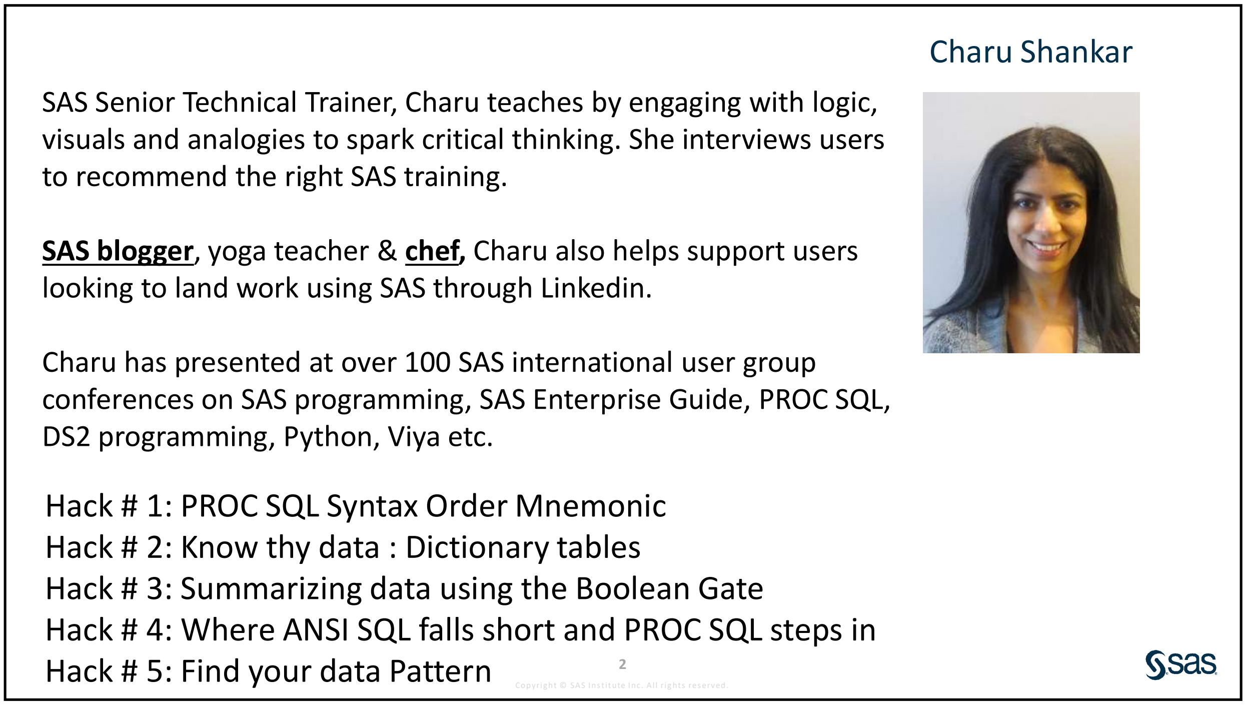


Top 5 PROC SQL Hacks

Create Your Own Adventure
29 July 2020

Charu Shankar
SAS Institute Canada Inc.

THE POWER TO KNOW.



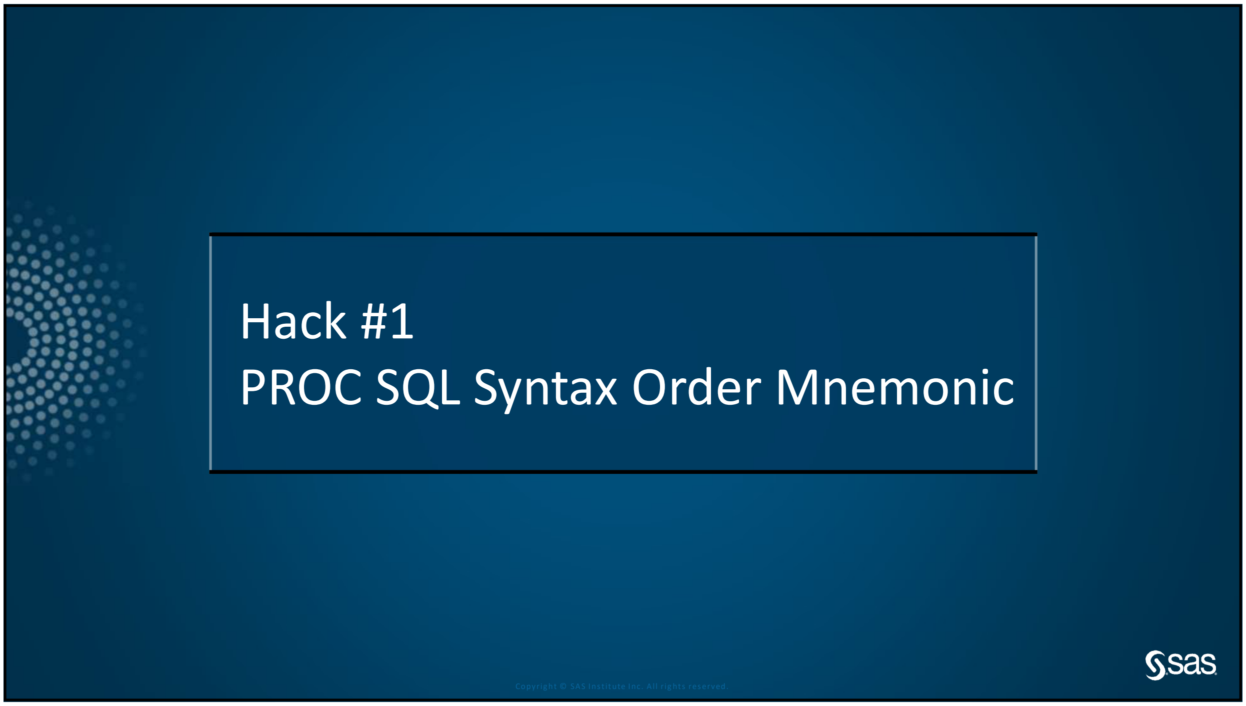
Charu Shankar

SAS Senior Technical Trainer, Charu teaches by engaging with logic, visuals and analogies to spark critical thinking. She interviews users to recommend the right SAS training.

SAS blogger, yoga teacher & **chef,** Charu also helps support users looking to land work using SAS through Linkedin.

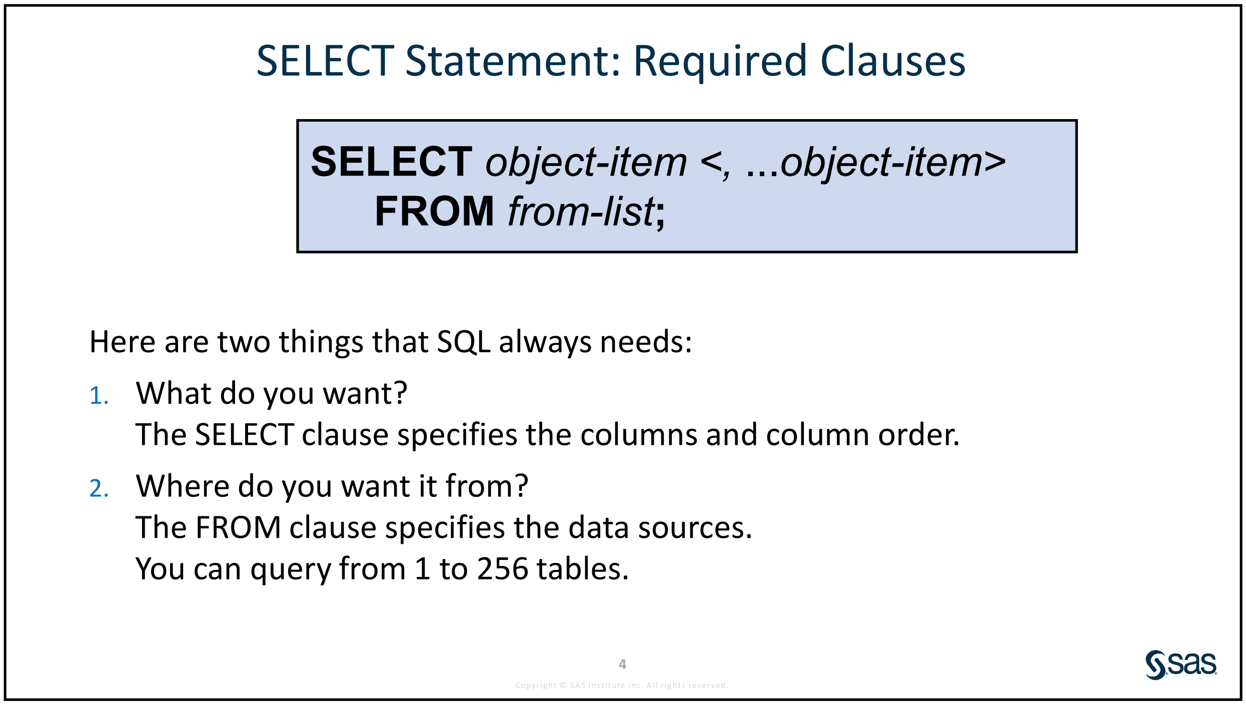
Charu has presented at over 100 SAS international user group conferences on SAS programming, SAS Enterprise Guide, PROC SQL, DS2 programming, Python, Viya etc.

Hack # 1: PROC SQL Syntax Order Mnemonic
Hack # 2: Know thy data : Dictionary tables
Hack # 3: Summarizing data using the Boolean Gate
Hack # 4: Where ANSI SQL falls short and PROC SQL steps in
Hack # 5: Find your data Pattern



Hack #1
PROC SQL Syntax Order Mnemonic

3



SELECT Statement: Required Clauses

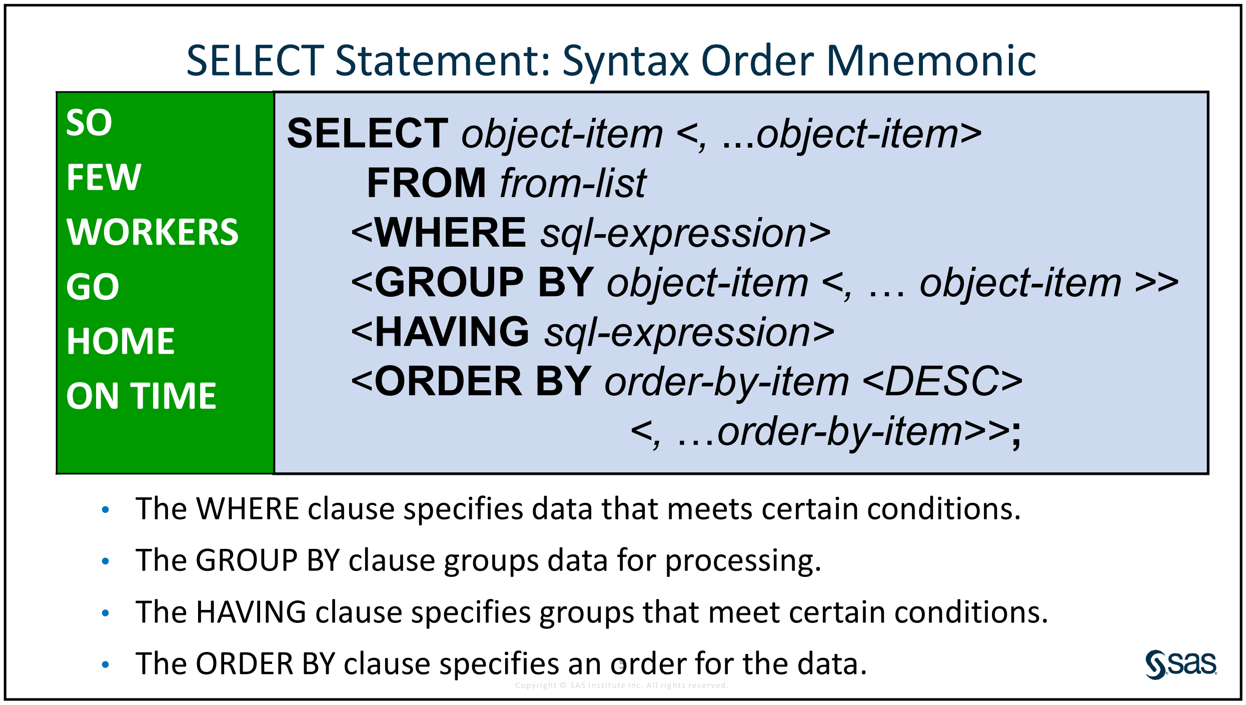
SELECT *object-item* <, ...*object-item*>
    FROM *from-list*;

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.

4

4

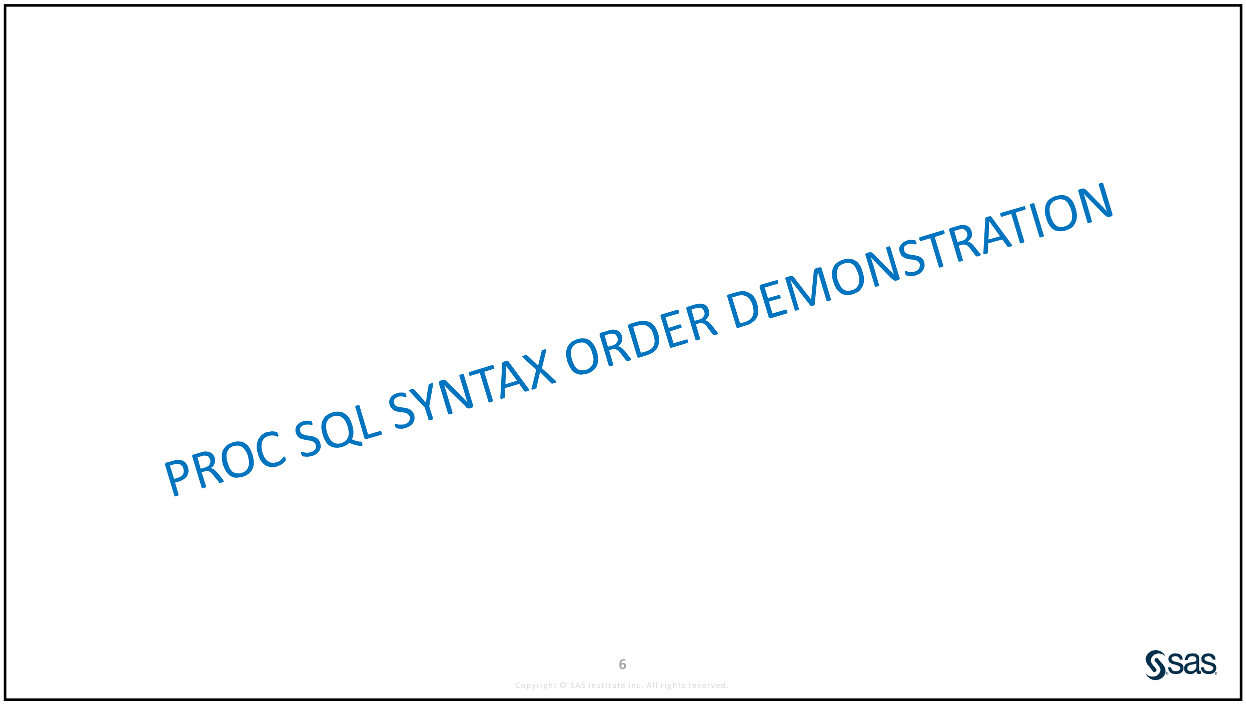


SELECT Statement: Syntax Order Mnemonic

SO	**SELECT** *object-item <, ...object-item>*
FEW	**FROM** *from-list*
WORKERS	<**WHERE** *sql-expression*>
GO	<**GROUP BY** *object-item <, … object-item >*>
HOME	<**HAVING** *sql-expression*>
ON TIME	<**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.

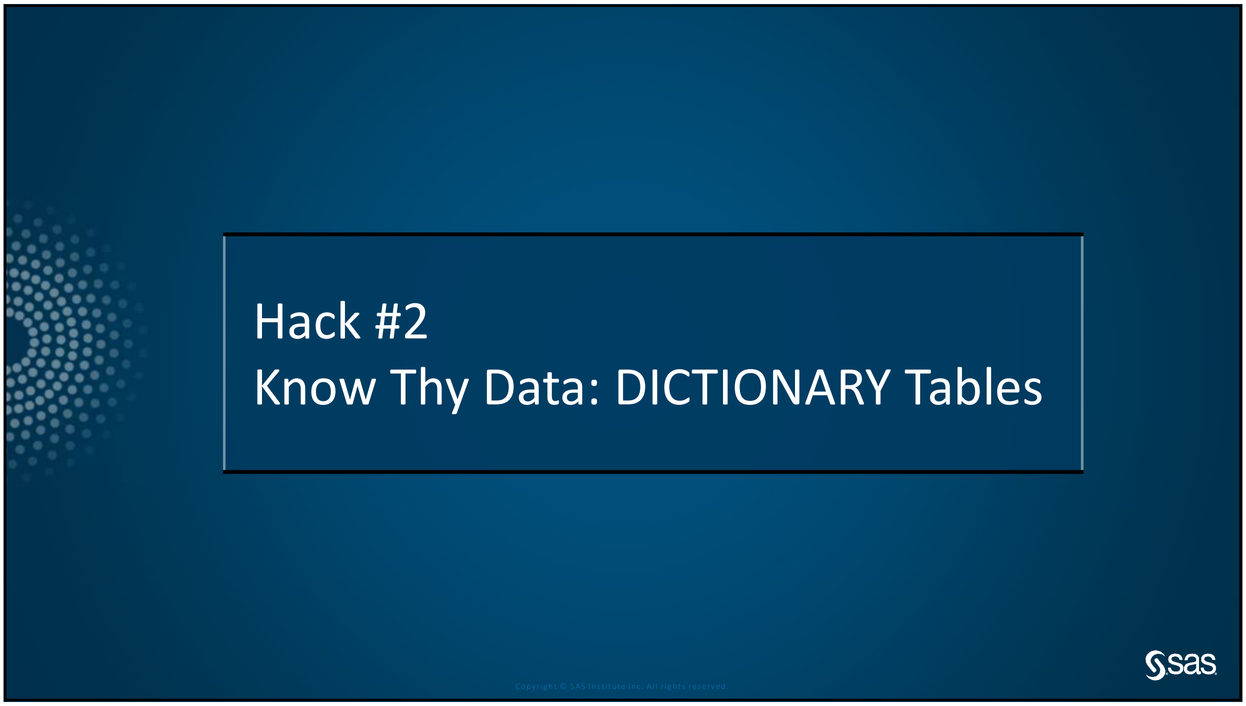
5



PROC SQL SYNTAX ORDER DEMONSTRATION

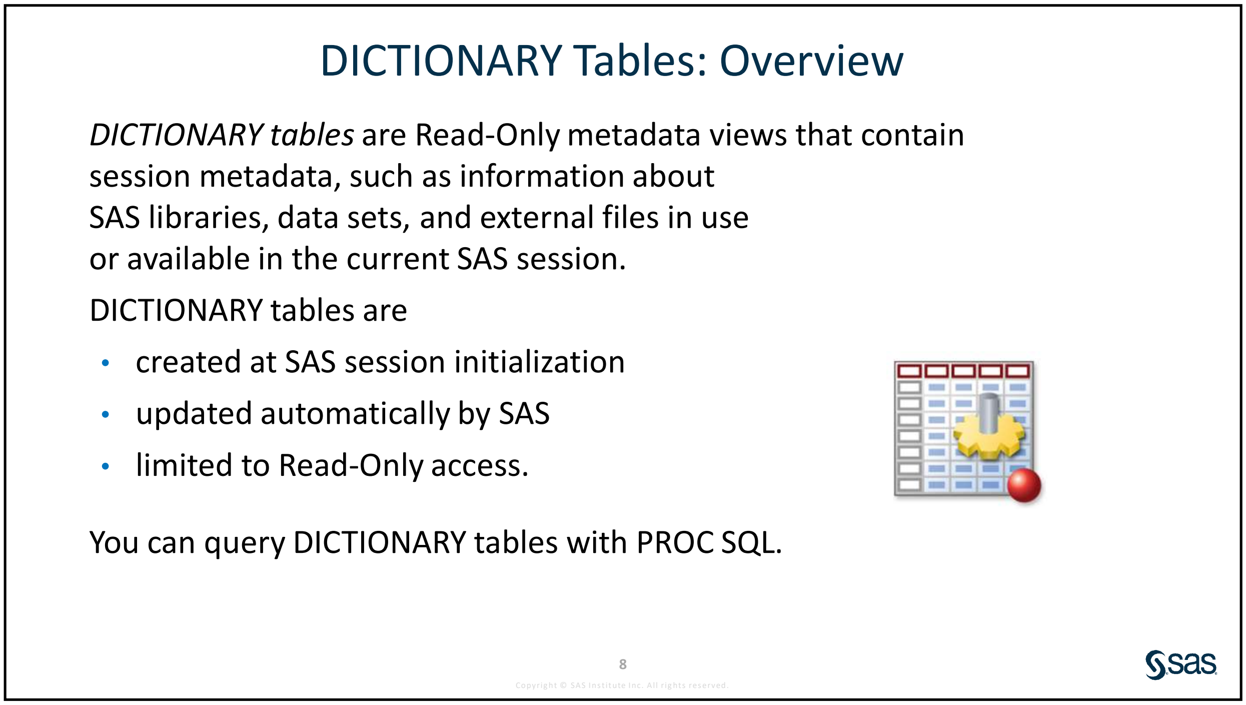
6

6



Hack #2
Know Thy Data: DICTIONARY Tables

7



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.

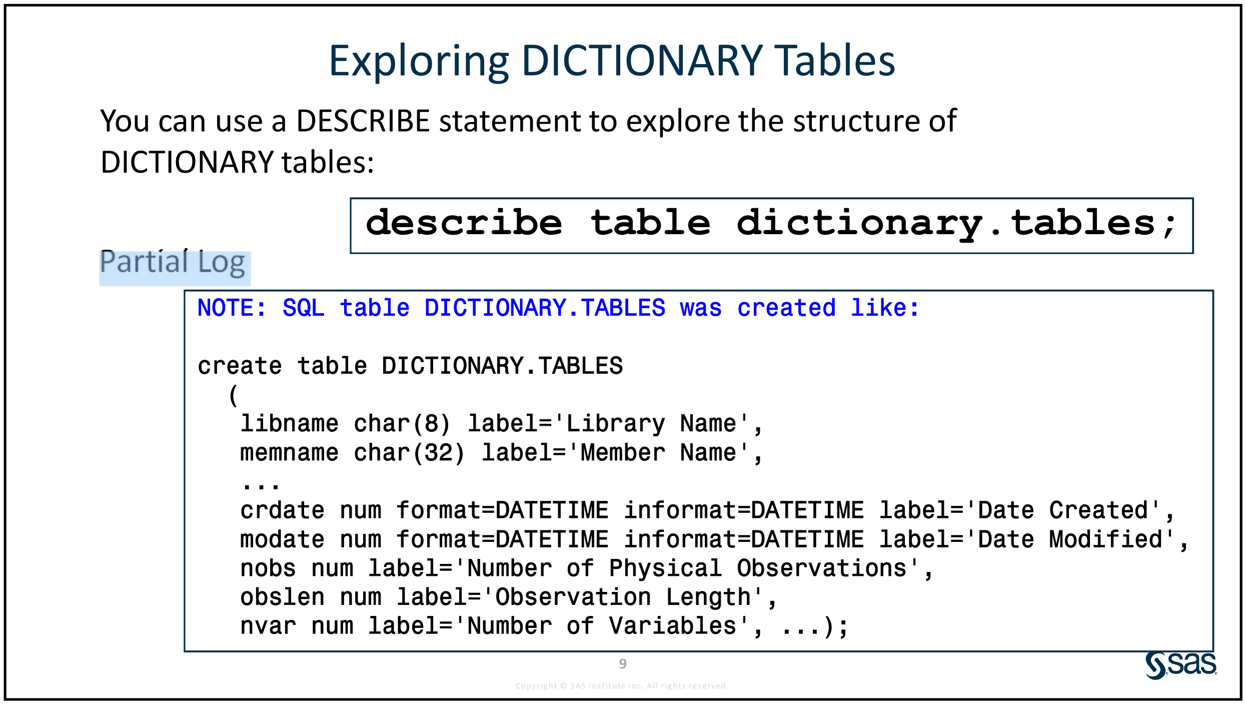
DICTIONARY tables are

- created at SAS session initialization
- updated automatically by SAS
- limited to Read-Only access.

You can query DICTIONARY tables with PROC SQL.

8

8



Exploring DICTIONARY Tables

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

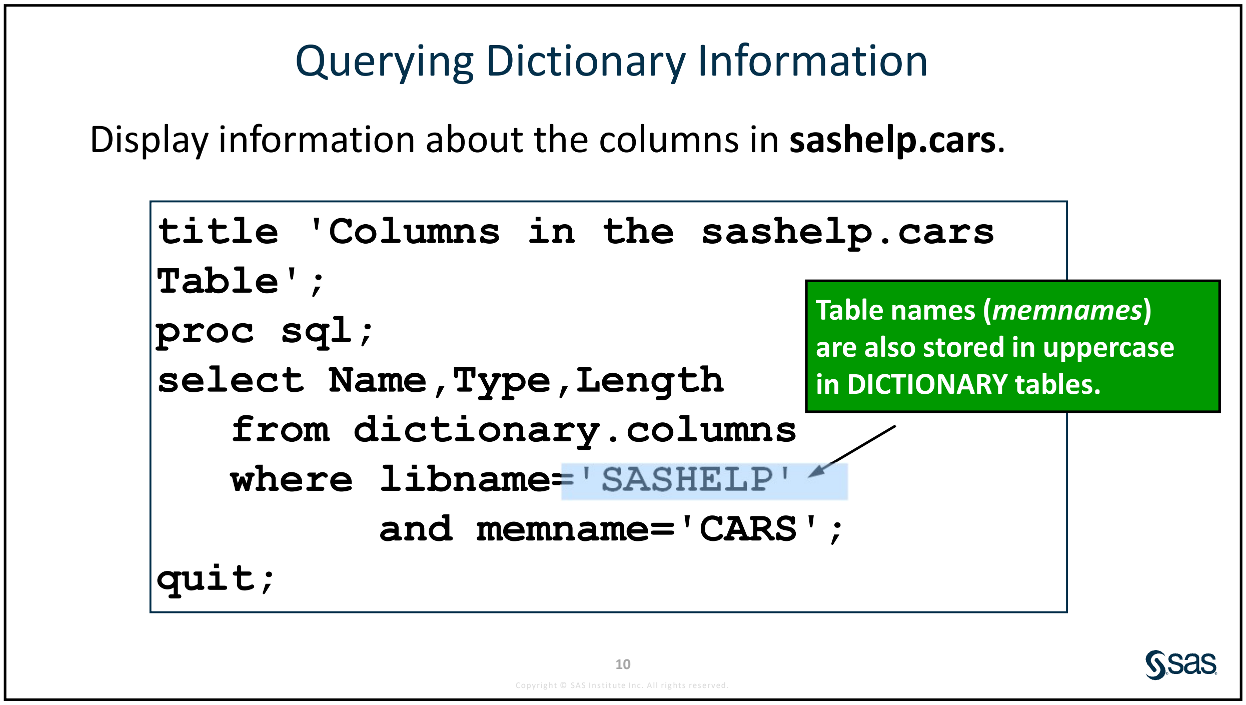
```
describe table dictionary.tables;
```

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', ...);
```

9

9



Querying Dictionary Information

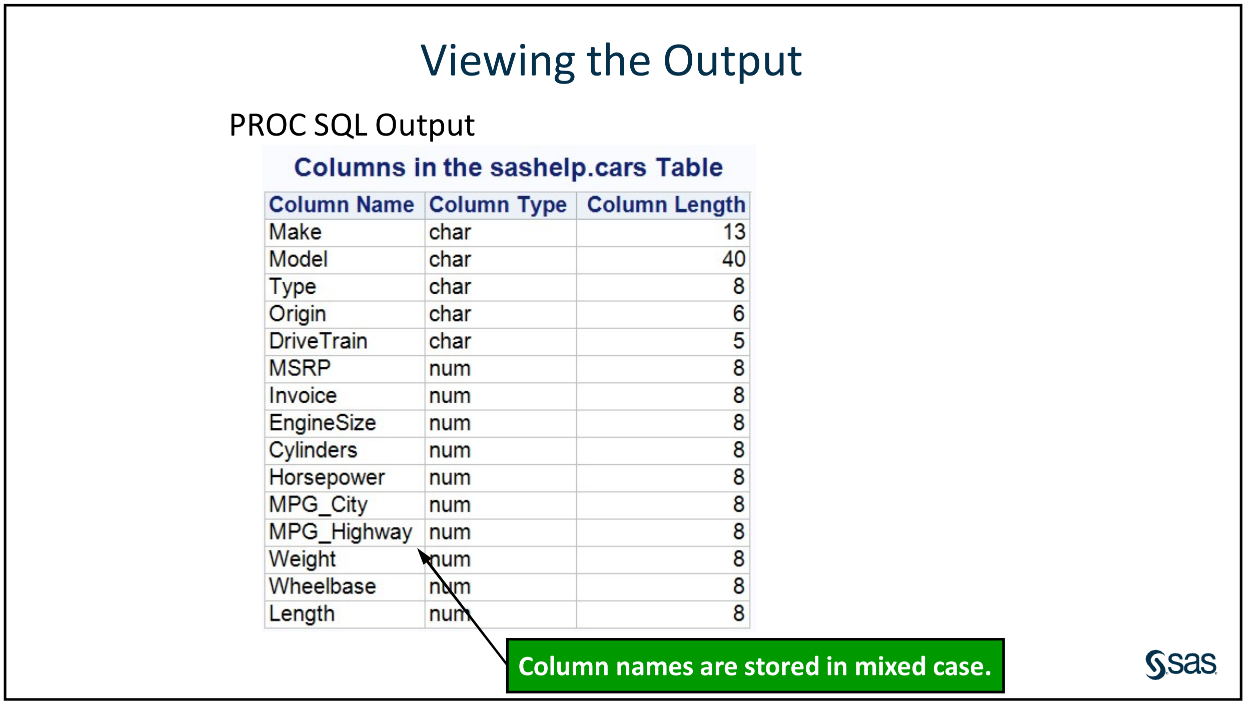
Display information about the columns in **sashelp.cars**.

```
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.

10

10



Viewing the Output

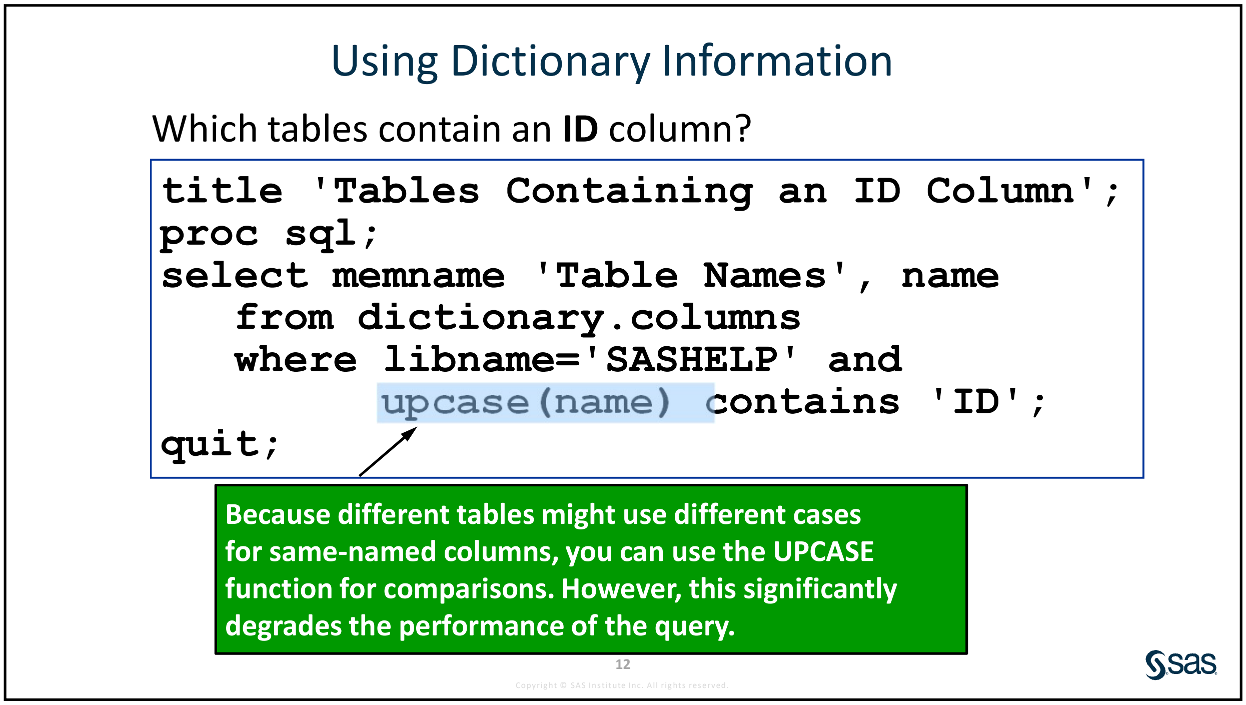
PROC SQL Output

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.

11



Using Dictionary Information

Which tables contain an **ID** column?

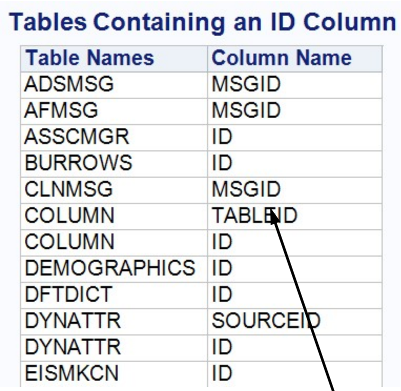
```
title 'Tables Containing an ID Column';
proc sql;
select memname 'Table Names', name
   from dictionary.columns
   where libname='SASHELP' and
         upcase(name) contains 'ID';
quit;
```

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.

12

12

Viewing the Output



Tables Containing an ID Column

Table Names	Column Name
ADSMSG	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.

13

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.

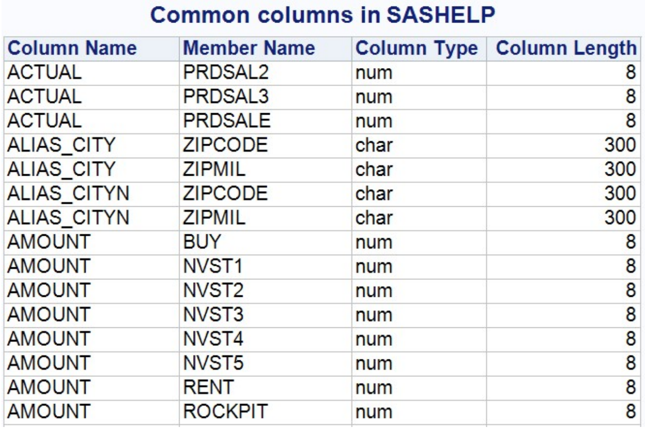
Use the following code

```
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;
```

14

14

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

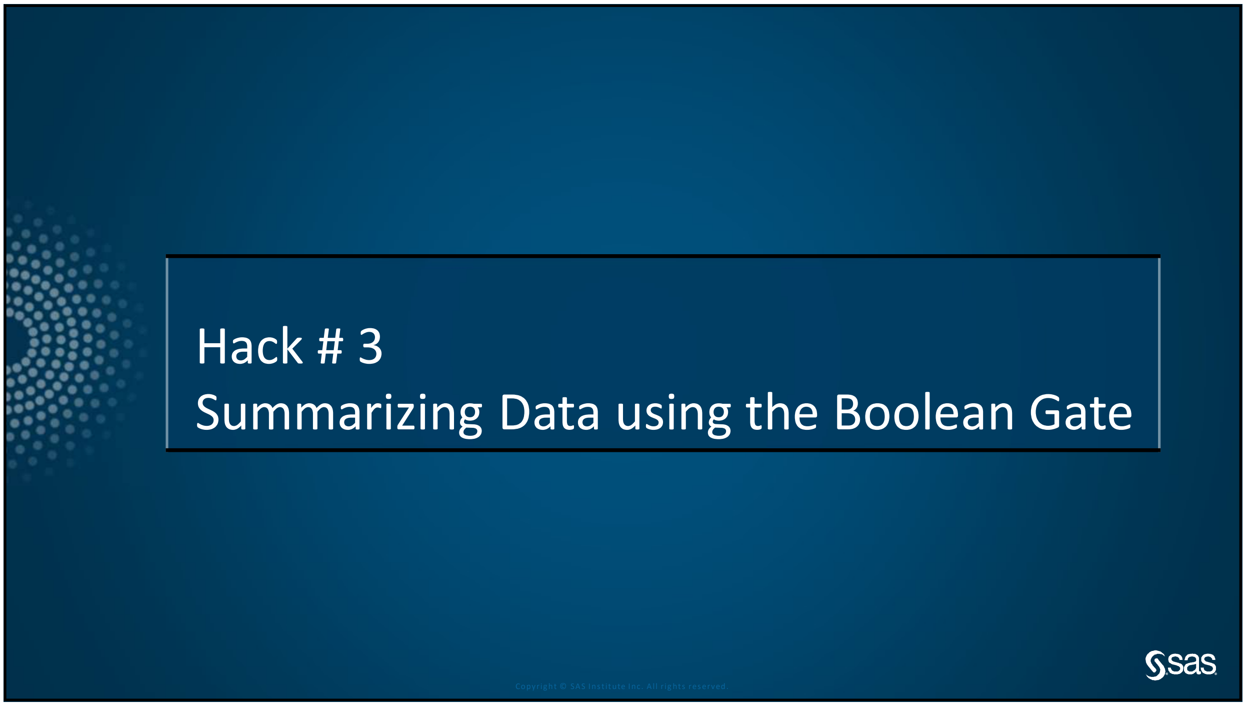
Joins are easier because the structure of each table does not have to be examined before determining common columns. Let SAS bring common columns dynamically by looking up DICTIONARY tables.

15

DICTIONARY TABLES DEMONSTRATION

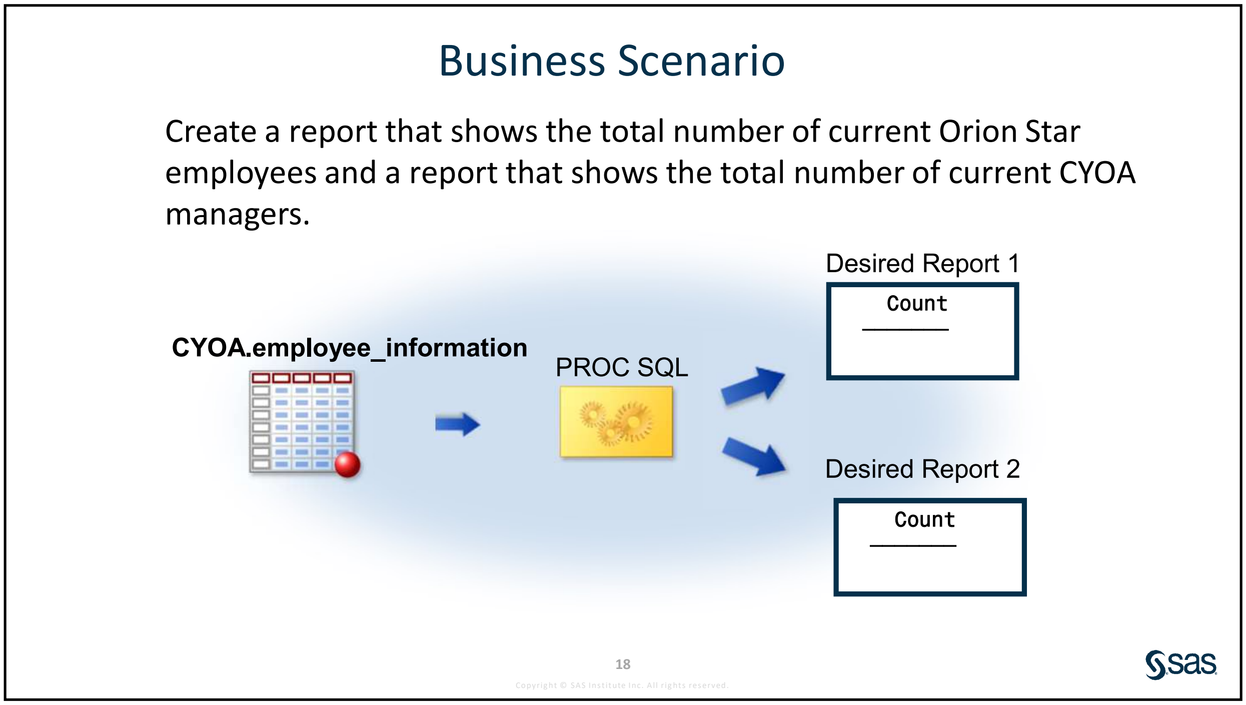
16

16



Hack # 3
Summarizing Data using the Boolean Gate

17



Business Scenario

Create a report that shows the total number of current Orion Star employees and a report that shows the total number of current CYOA managers.

CYOA.employee_information

PROC SQL

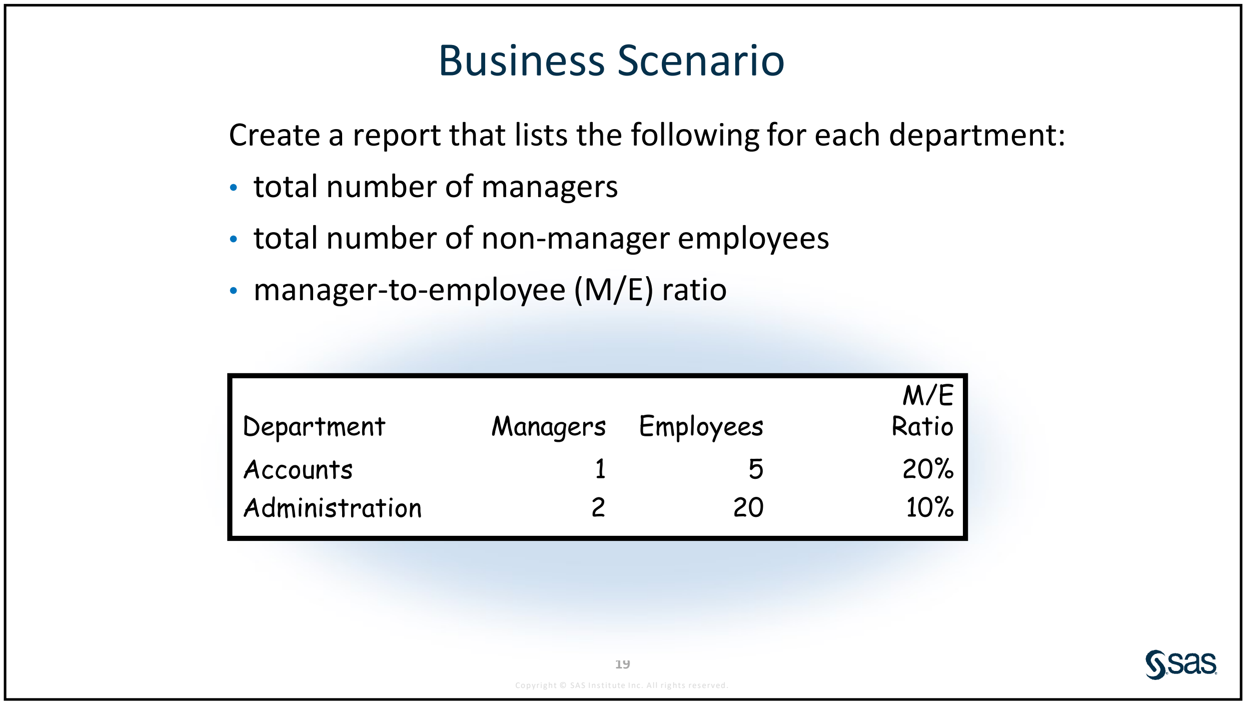
Desired Report 1

Count

Desired Report 2

Count

18



Business Scenario

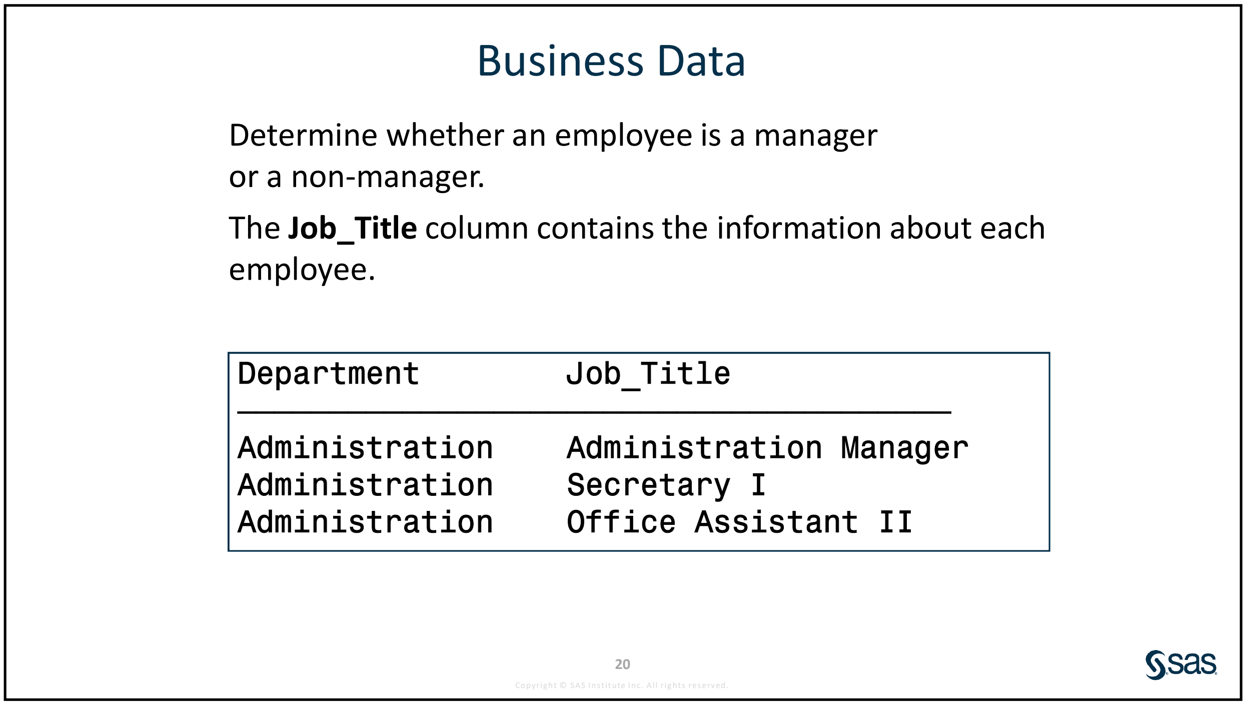
Create a report that lists the following for each department:

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

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

19

19



Business Data

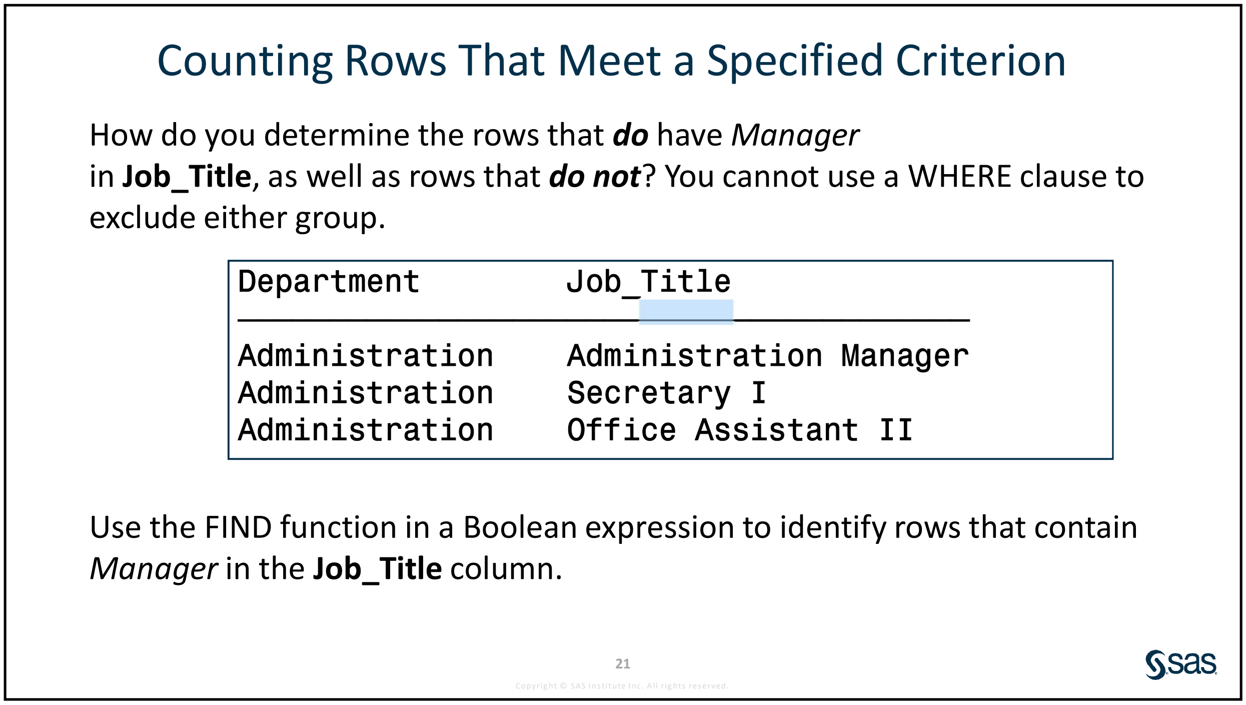
Determine whether an employee is a manager
or a non-manager.

The **Job_Title** column contains the information about each employee.

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

20

20



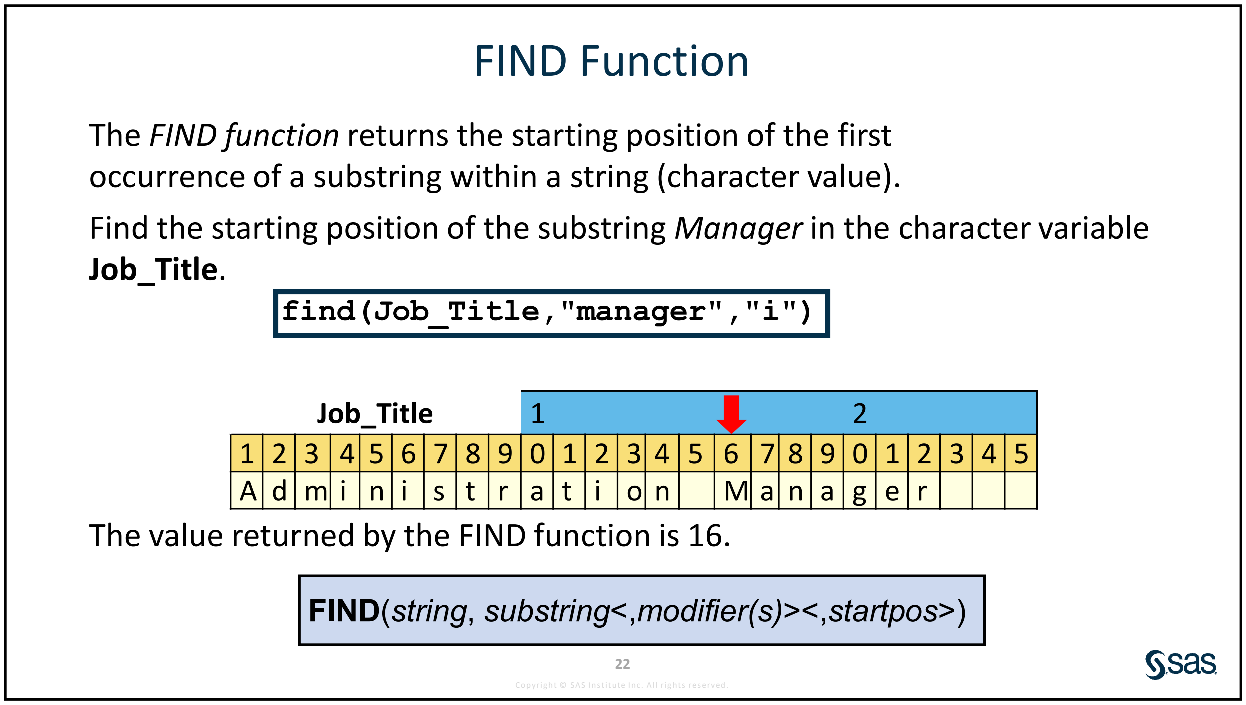
Counting Rows That Meet a Specified Criterion

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.

21



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")
```

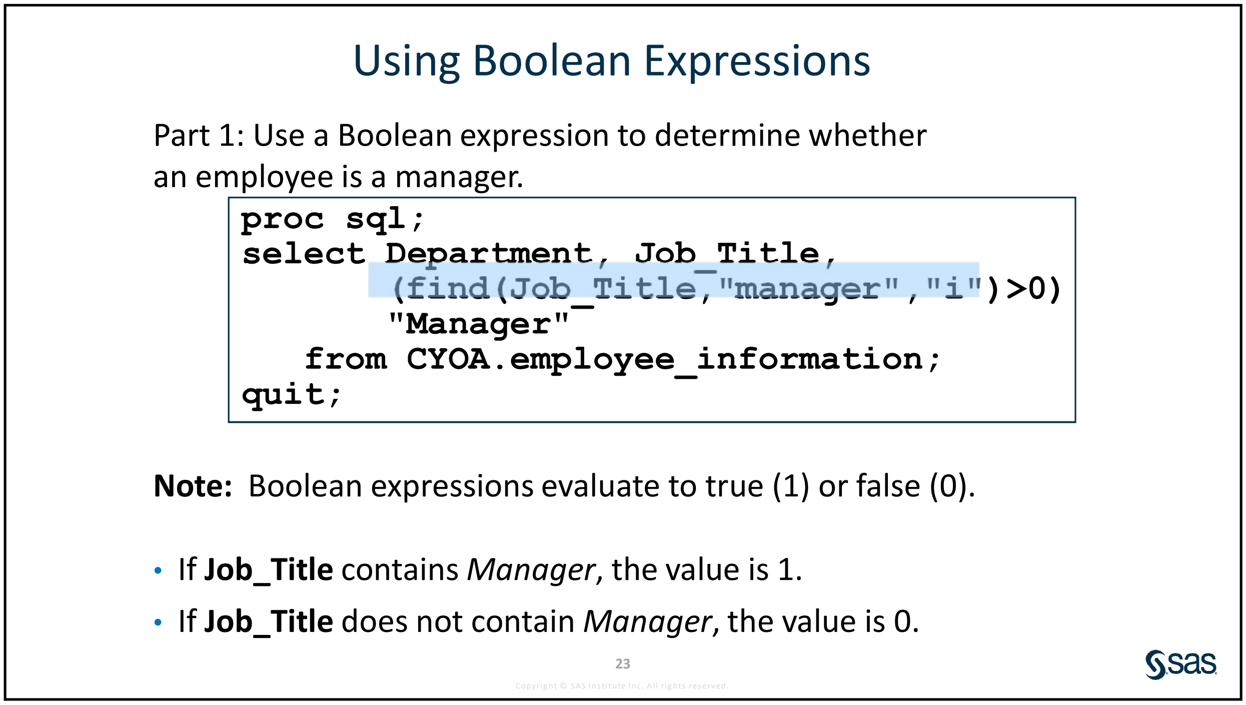
Job_Title

									1										2					
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)*><,*startpos*>)

22



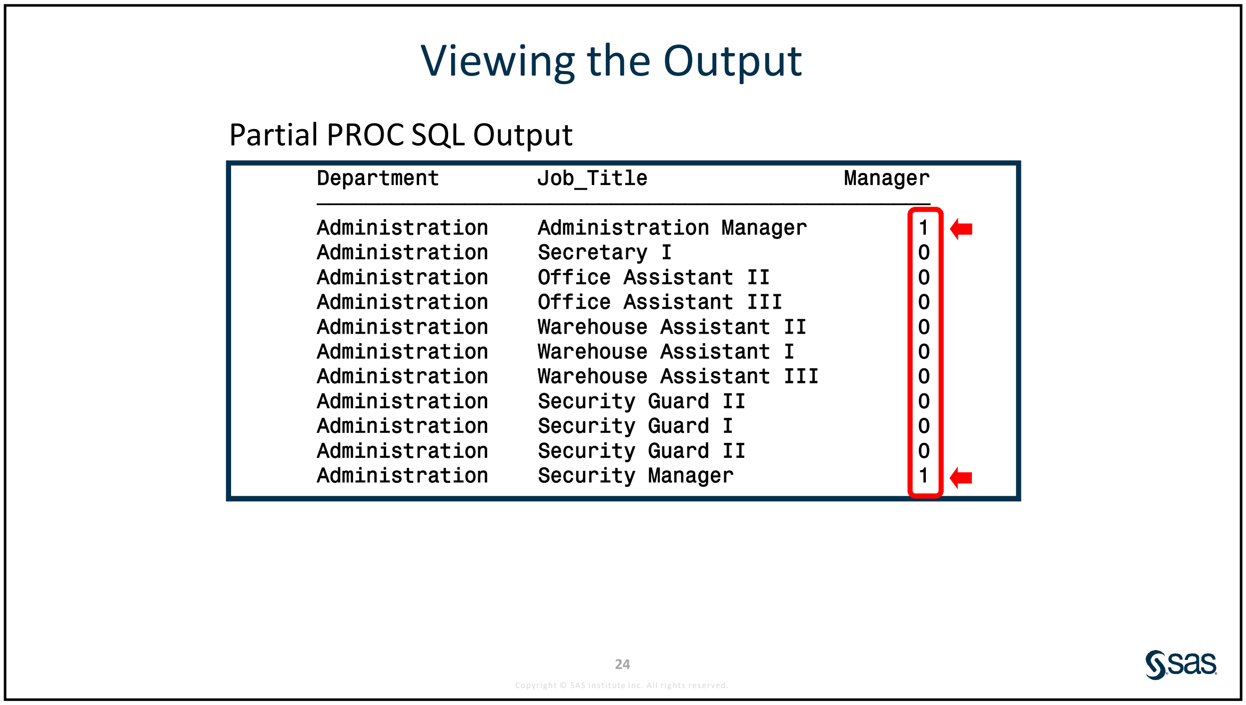
Using Boolean Expressions

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 CYOA.employee_information;
quit;
```

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

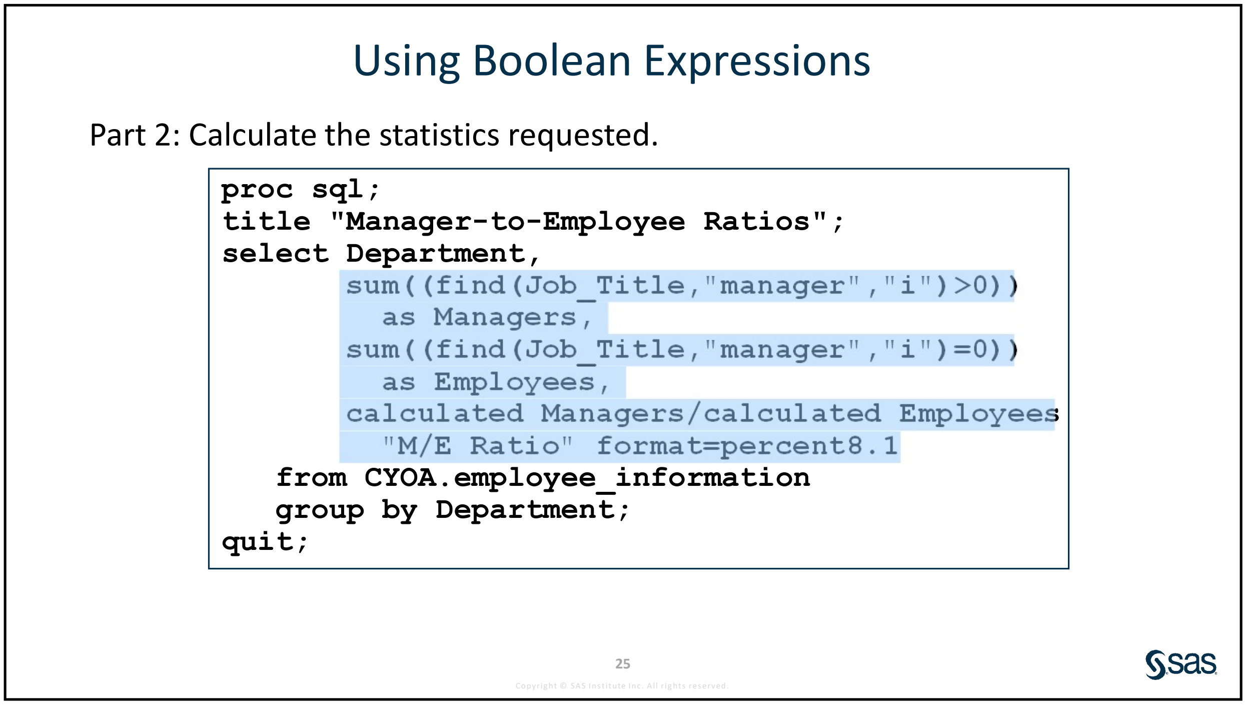
- If **Job_Title** contains *Manager*, the value is 1.
- If **Job_Title** does not contain *Manager*, the value is 0.



Viewing the Output

Partial PROC SQL Output

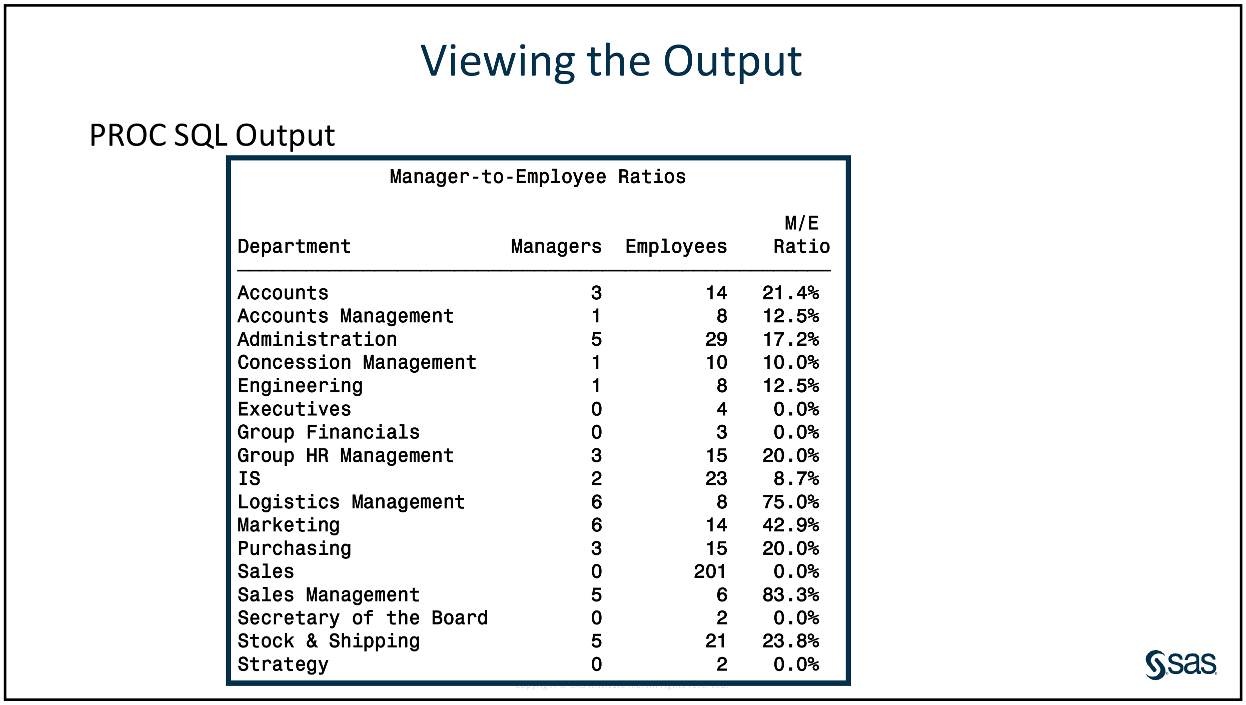
```
Department          Job_Title                  Manager
------------------------------------------------------
Administration      Administration Manager           1
Administration      Secretary I                      0
Administration      Office Assistant II              0
Administration      Office 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.

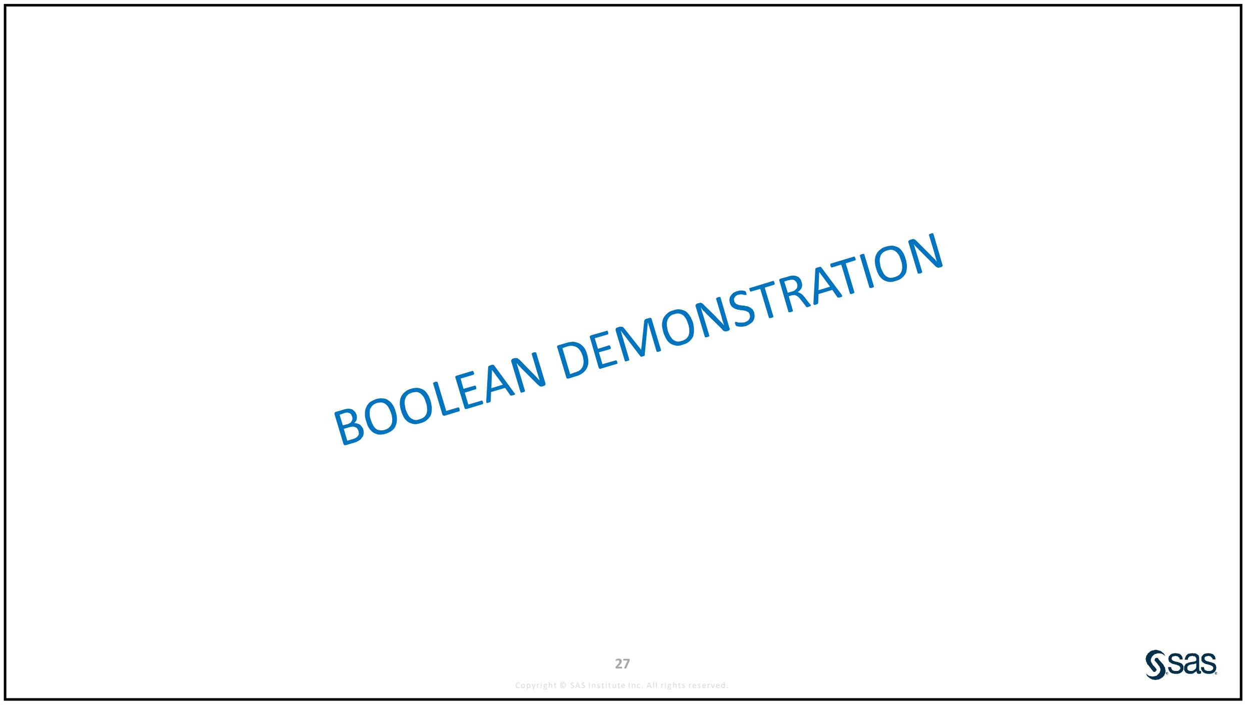
```
proc sql;
title "Manager-to-Employee Ratios";
select Department,
       sum((find(Job_Title,"manager","i")>0))
         as Managers,
       sum((find(Job_Title,"manager","i")=0))
         as Employees,
       calculated Managers/calculated Employees
         "M/E Ratio" format=percent8.1
   from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



Viewing the Output

PROC SQL Output

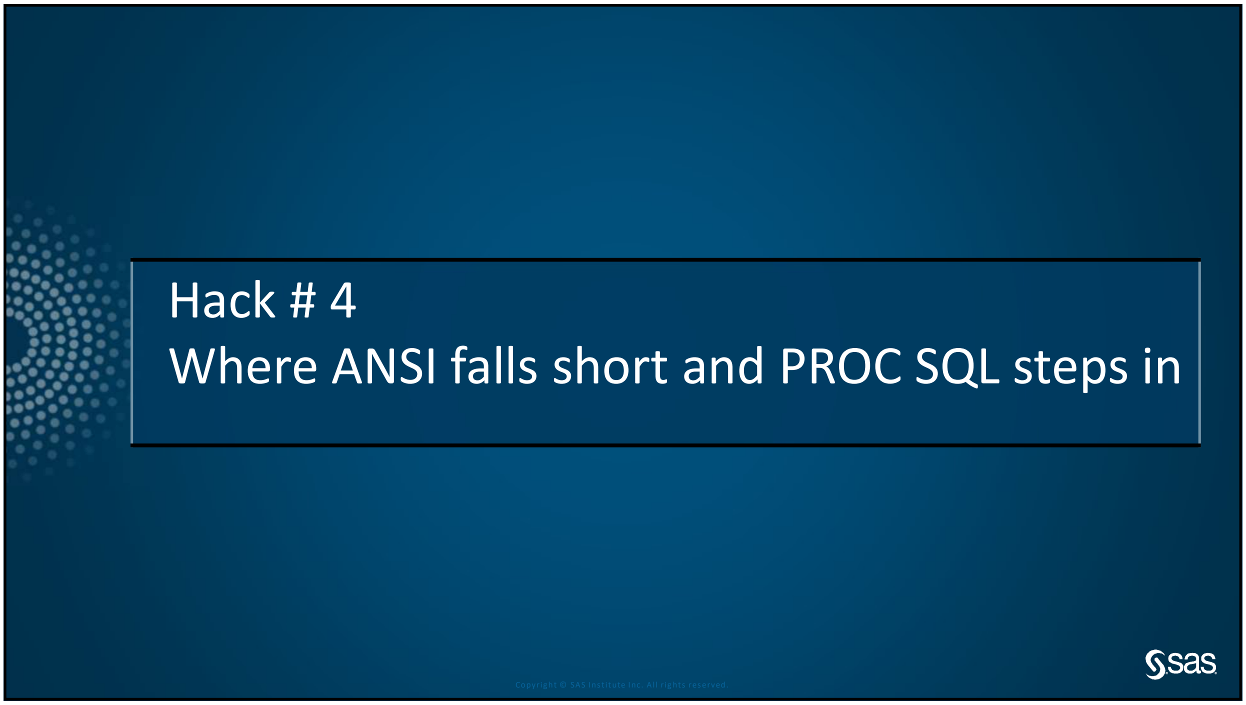
```
                    Manager-to-Employee Ratios

                                                            M/E
Department                   Managers  Employees          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%
```



BOOLEAN DEMONSTRATION

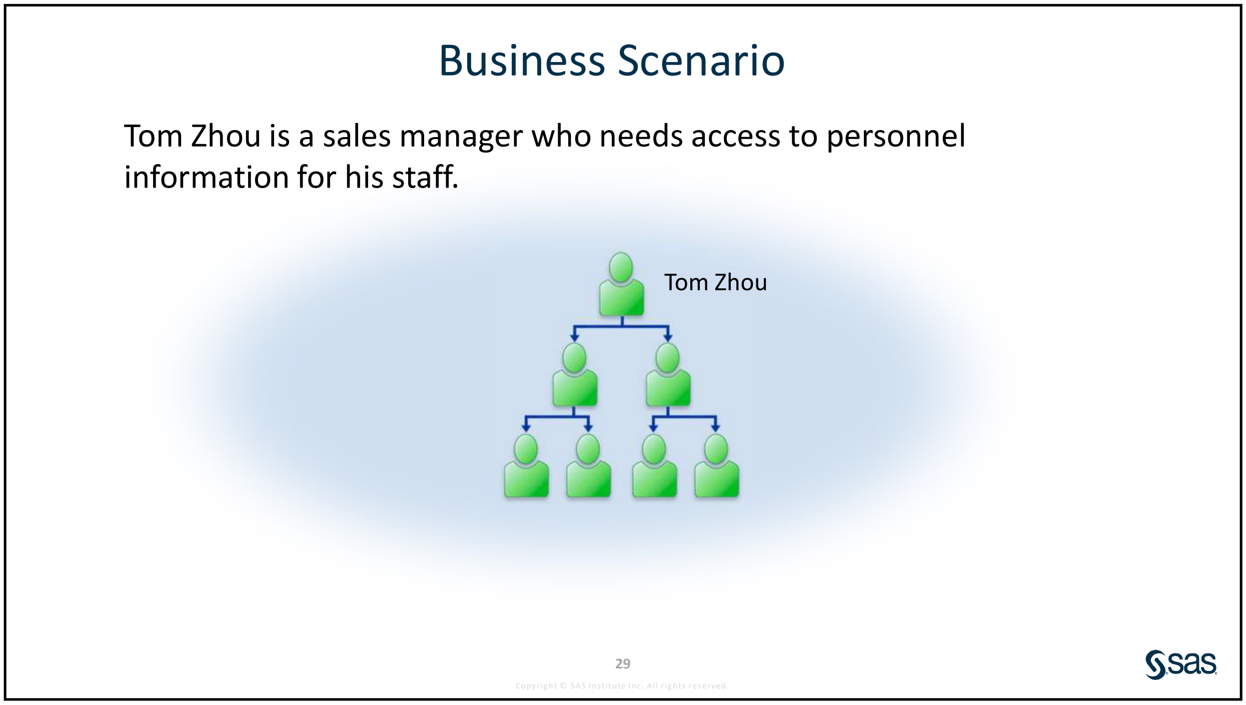
27

27



Hack # 4
Where ANSI falls short and PROC SQL steps in

28

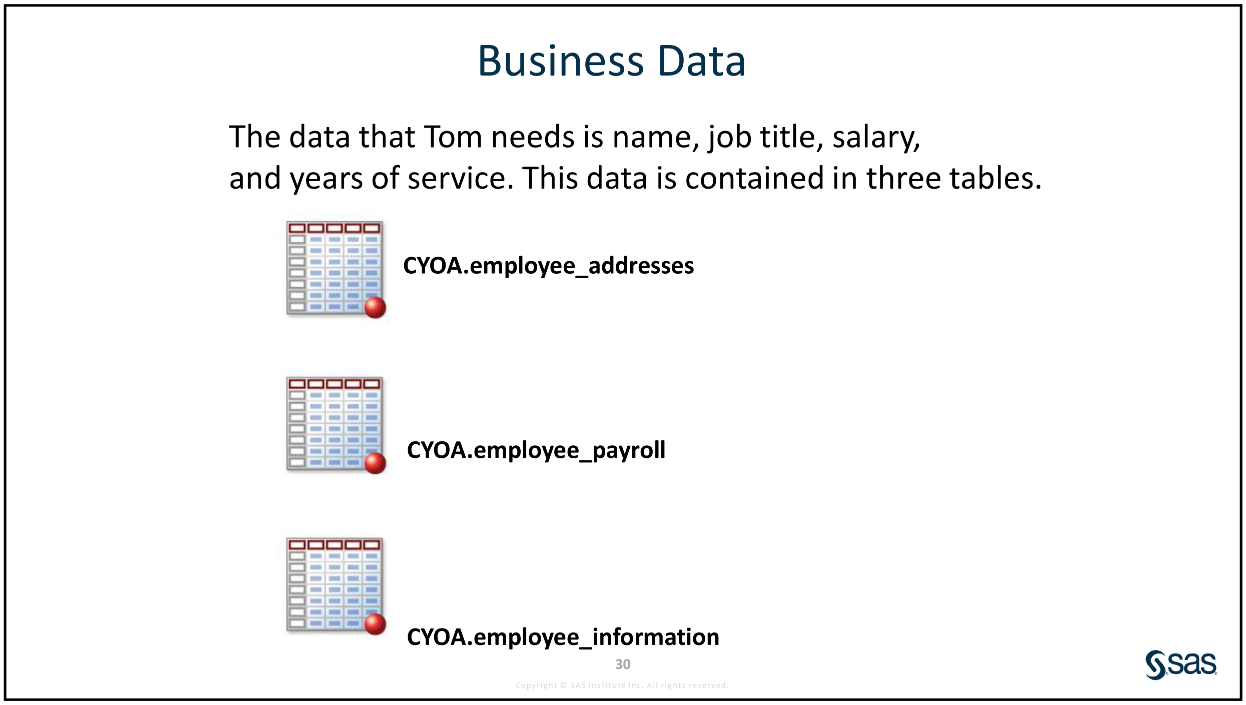


Business Scenario

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

29

29



Business Data

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

CYOA.employee_addresses

CYOA.employee_payroll

CYOA.employee_information

30

30

What Is a PROC SQL View?

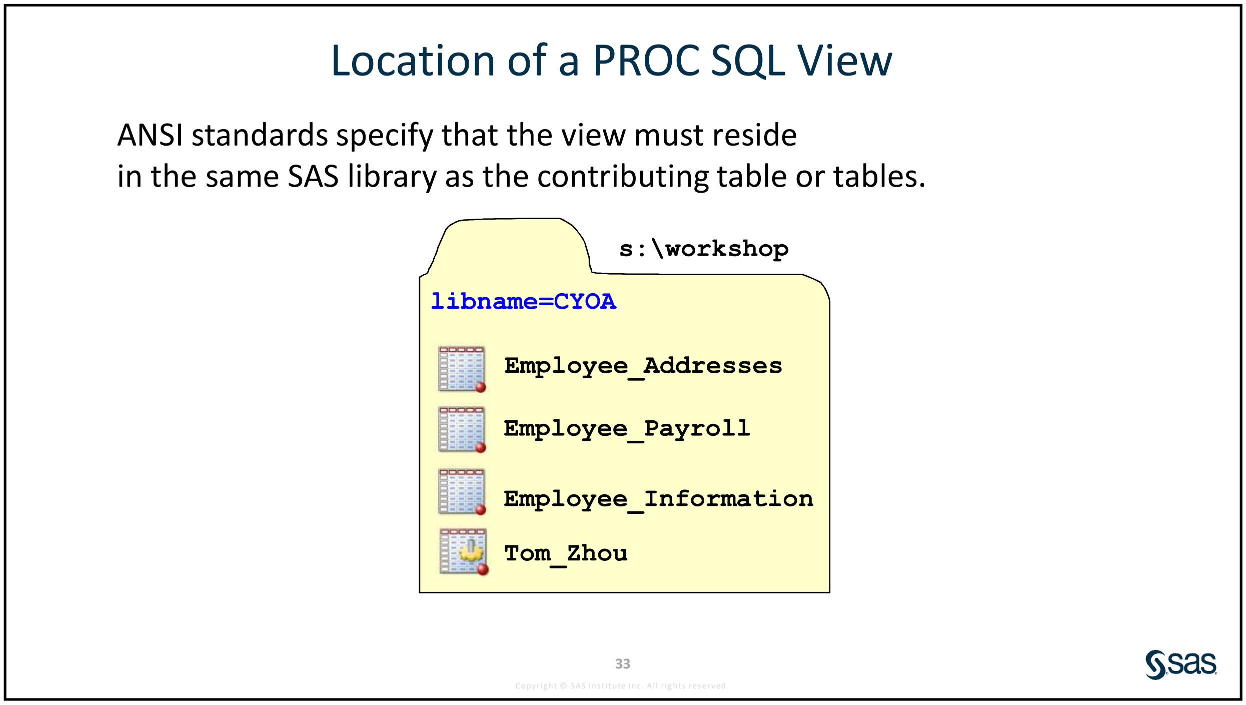
A *PROC SQL view*

- 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

Partial SAS Log

```
5      proc sql;
46   create view CYOA.tom_zhou as
47    select Employee_Name as Name format=$25.0,
48         Job_Title as Title format=$15.0,
49       p.Salary 'Annual Salary' format=comma10.2,
50               int((today()- p.Employee_Hire_Date)/365.25)
51                       as YOS 'Years of Service'
52                from employee_addresses as a,
53                     employee_payroll as p,
54                     employee_information 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 CYOA.TOM_ZHOU has been defined.
```



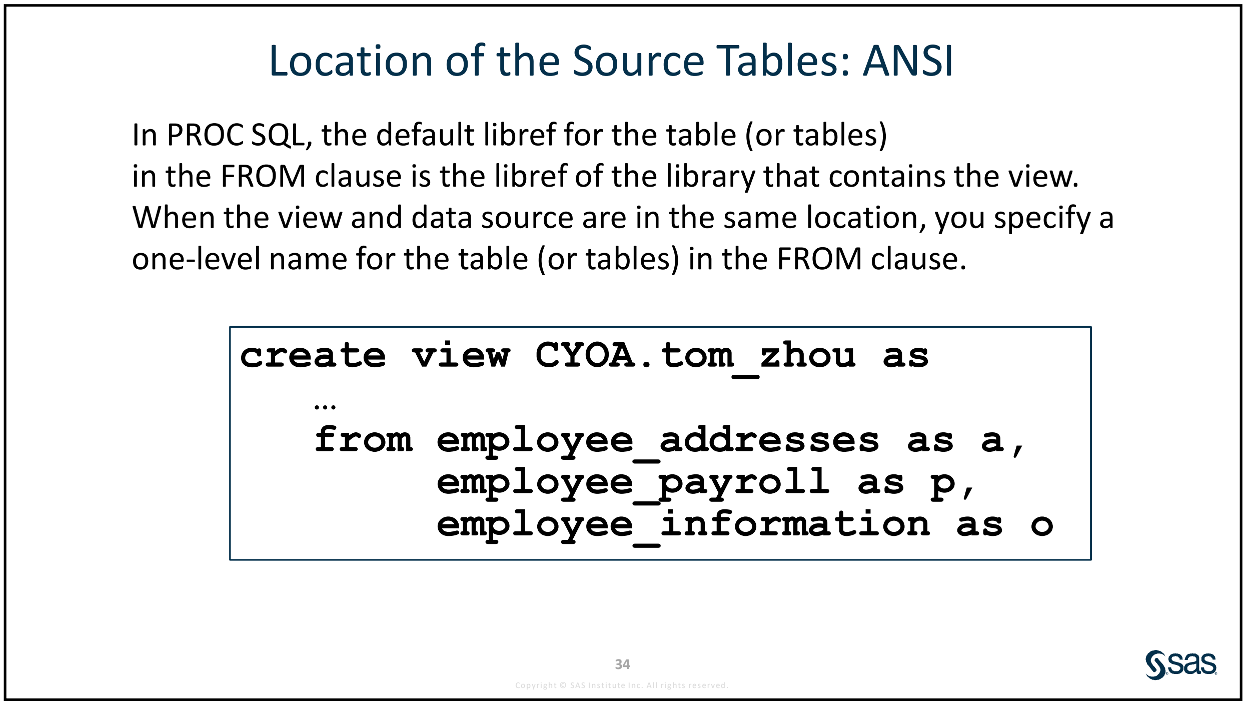
Location of a PROC SQL View

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

s:\workshop
libname=CYOA
Employee_Addresses
Employee_Payroll
Employee_Information
Tom_Zhou

33

33



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 CYOA.tom_zhou as
   …
   from employee_addresses as a,
        employee_payroll as p,
        employee_information as o
```

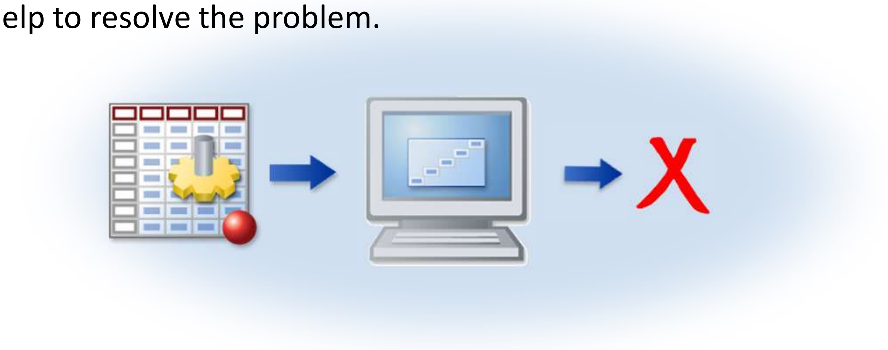
34

34

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.

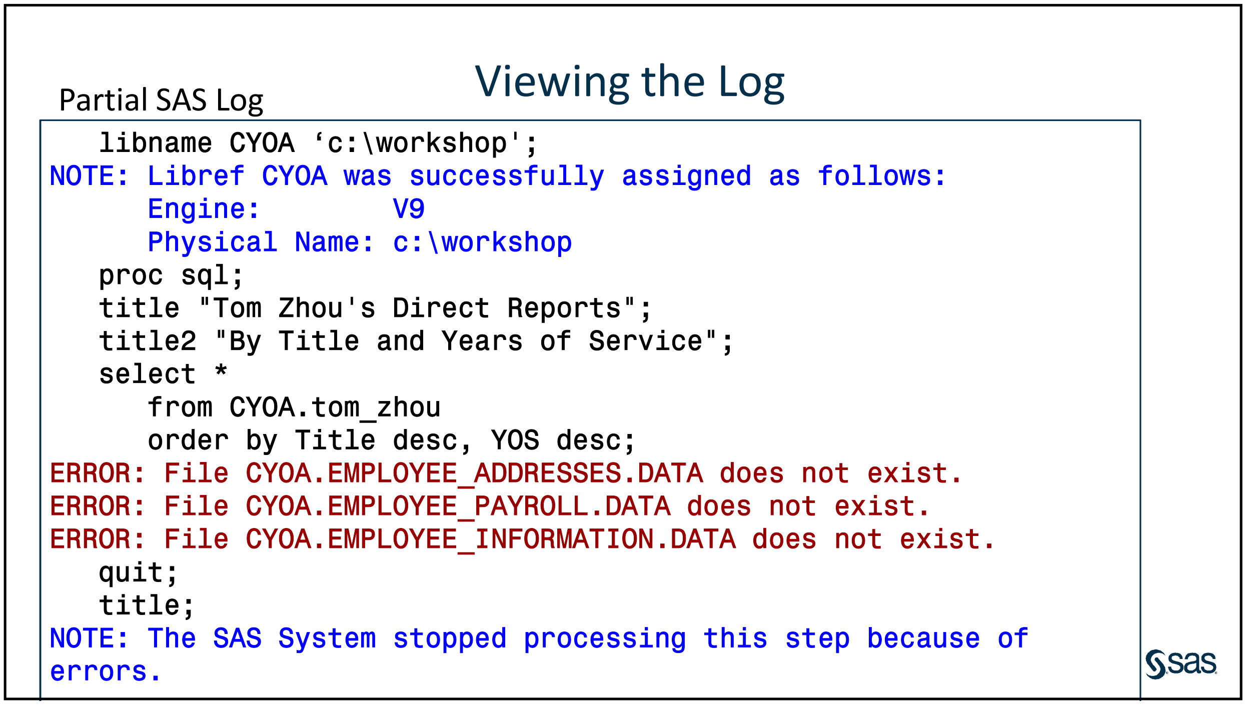


Exploring the Problem

Tom submitted the following:

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

s107d09

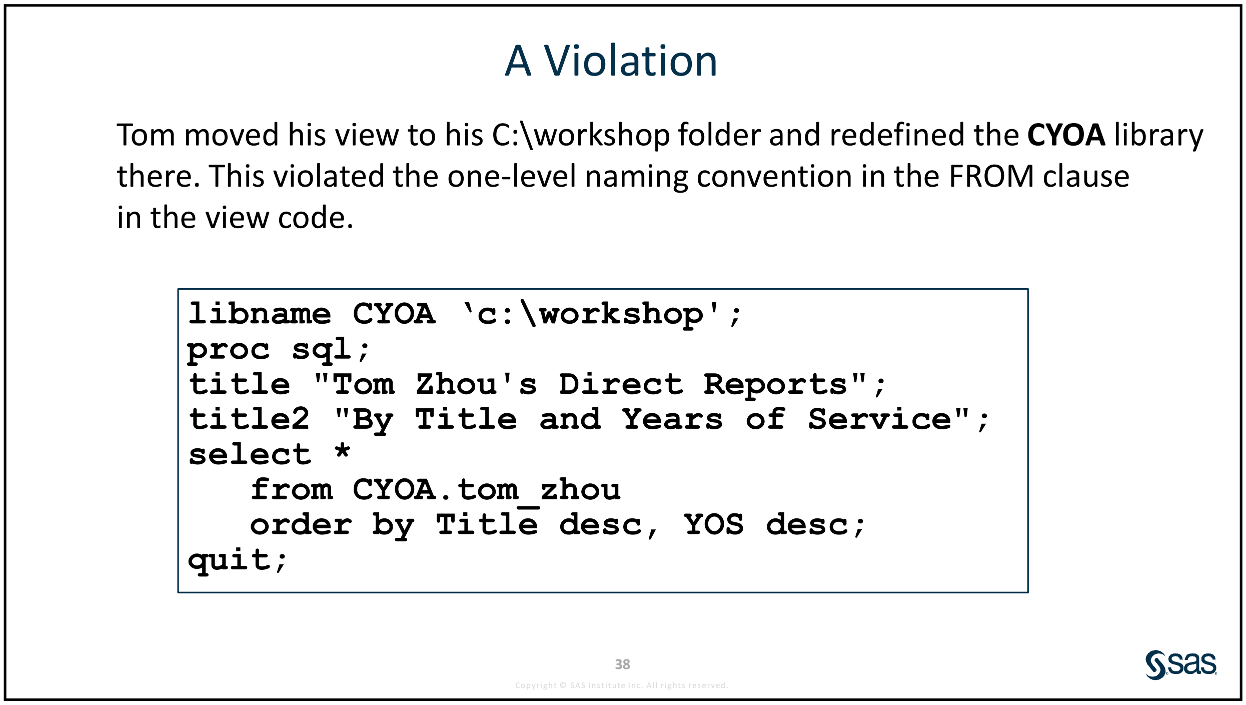


Viewing the Log

Partial SAS Log

```
   libname CYOA ‘c:\workshop';
NOTE: Libref CYOA 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 CYOA.tom_zhou
      order by Title desc, YOS desc;
ERROR: File CYOA.EMPLOYEE_ADDRESSES.DATA does not exist.
ERROR: File CYOA.EMPLOYEE_PAYROLL.DATA does not exist.
ERROR: File CYOA.EMPLOYEE_INFORMATION.DATA does not exist.
   quit;
   title;
NOTE: The SAS System stopped processing this step because of
errors.
```

37



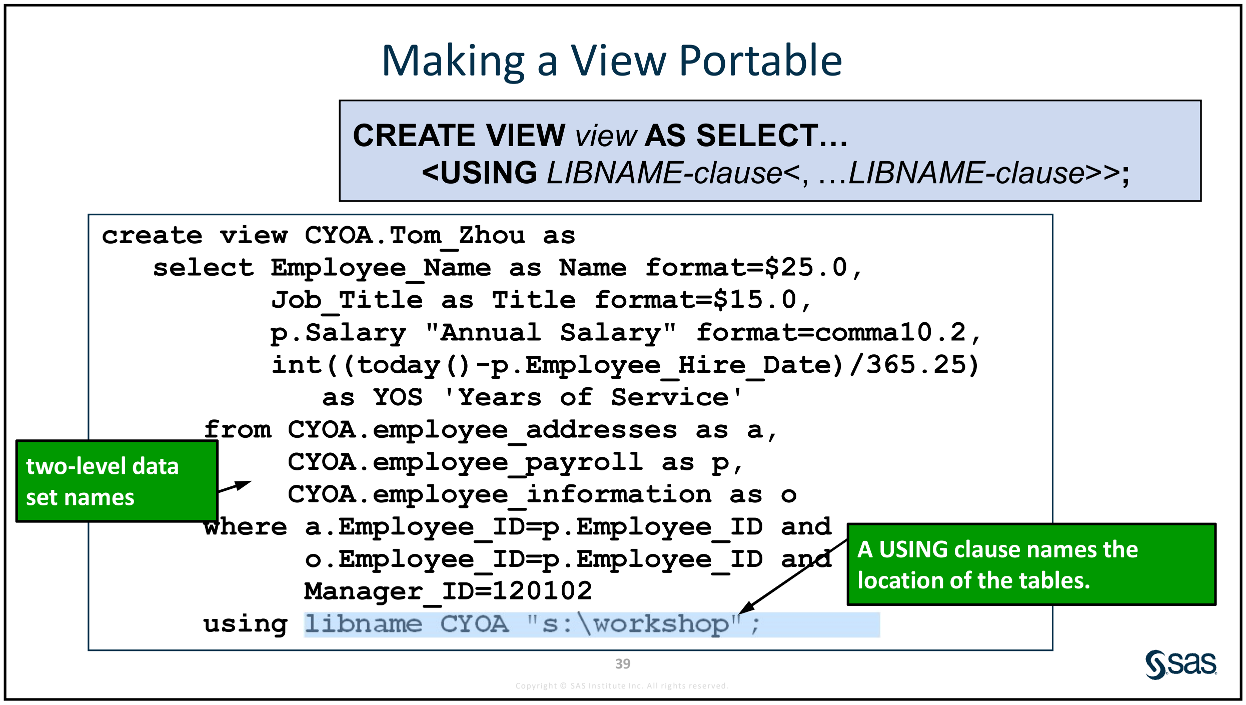
A Violation

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

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

38

38



Making a View Portable

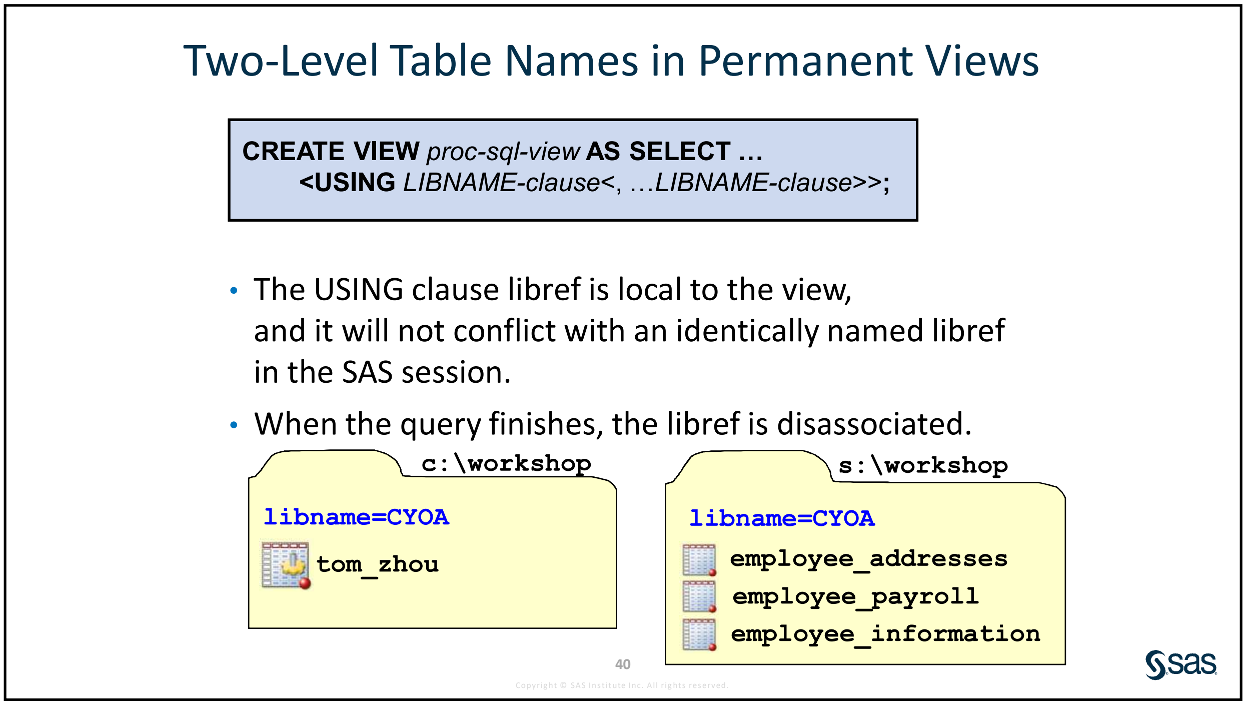
CREATE VIEW *view* **AS SELECT…**
<USING *LIBNAME-clause*<, …*LIBNAME-clause*>>;

```
create view CYOA.Tom_Zhou as
   select Employee_Name as Name format=$25.0,
          Job_Title as Title format=$15.0,
          p.Salary "Annual Salary" format=comma10.2,
          int((today()-p.Employee_Hire_Date)/365.25)
             as YOS 'Years of Service'
      from CYOA.employee_addresses as a,
           CYOA.employee_payroll as p,
           CYOA.employee_information as o
      where a.Employee_ID=p.Employee_ID and
            o.Employee_ID=p.Employee_ID and
            Manager_ID=120102
      using libname CYOA "s:\workshop";
```

two-level data set names

A USING clause names the location of the tables.

39



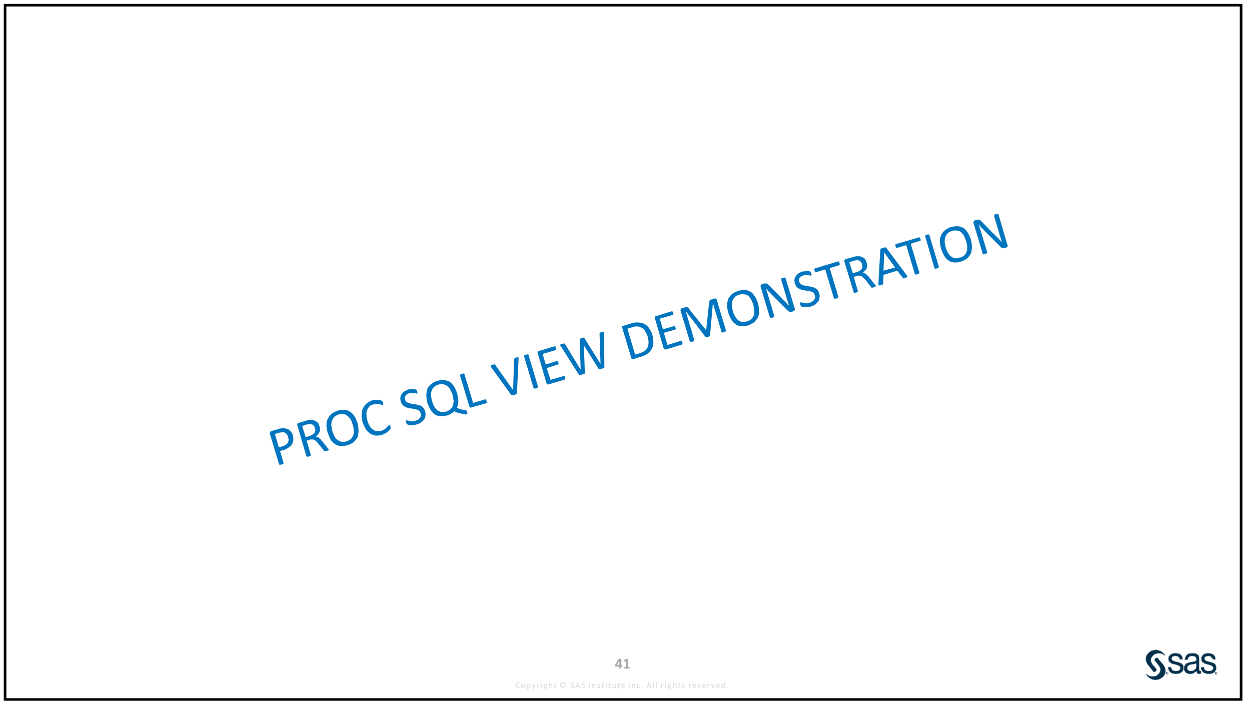
Two-Level Table Names in Permanent Views

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.

c:\workshop
libname=CYOA
tom_zhou
s:\workshop
libname=CYOA
employee_addresses
employee_payroll
employee_information

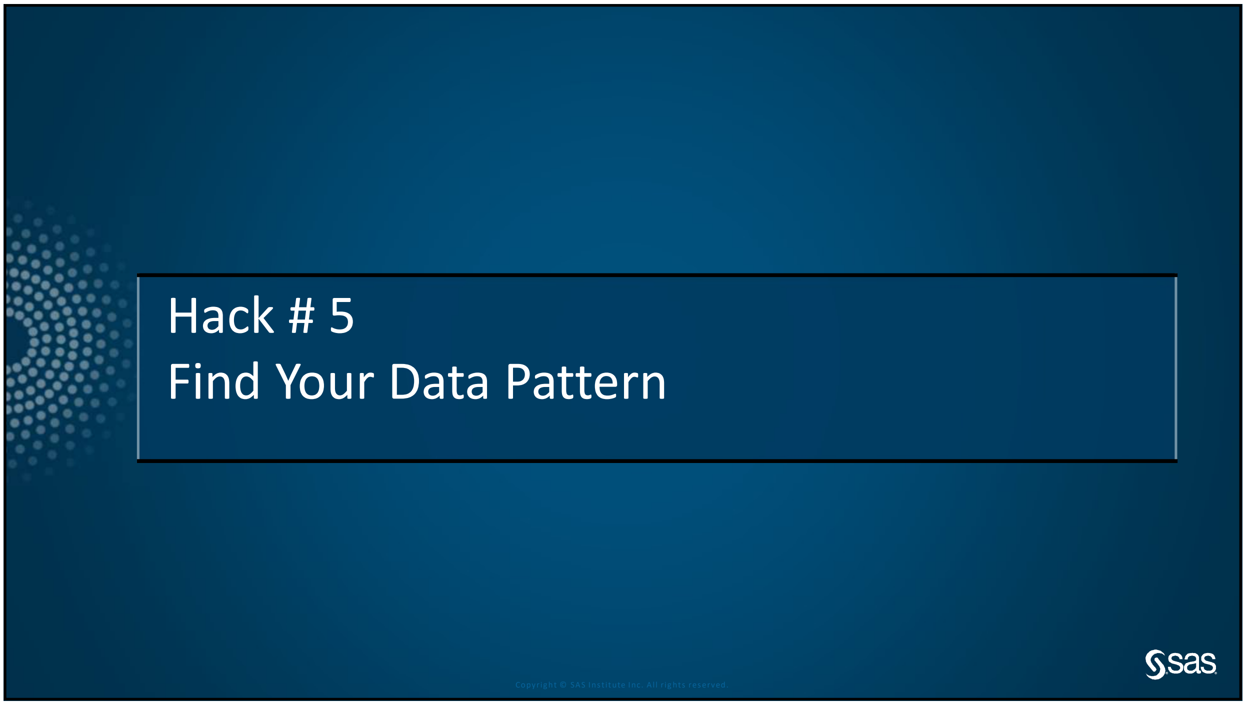
40



PROC SQL VIEW DEMONSTRATION

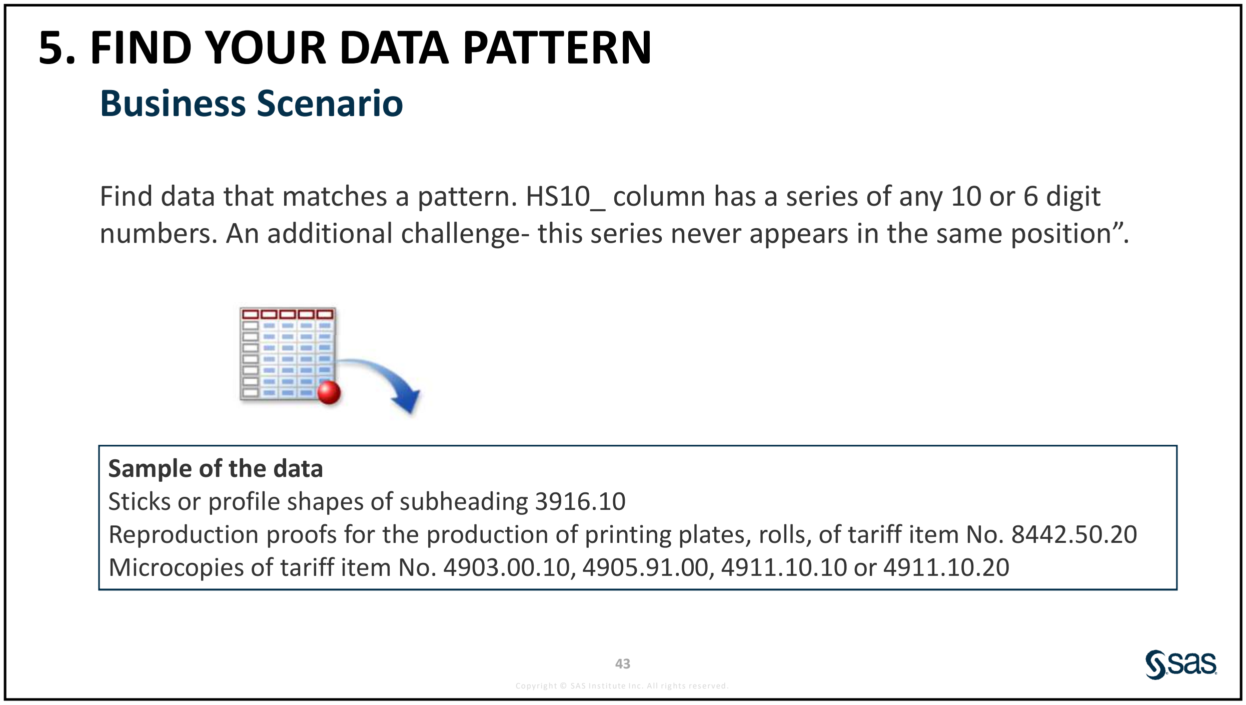
41

41



Hack # 5
Find Your Data Pattern

42

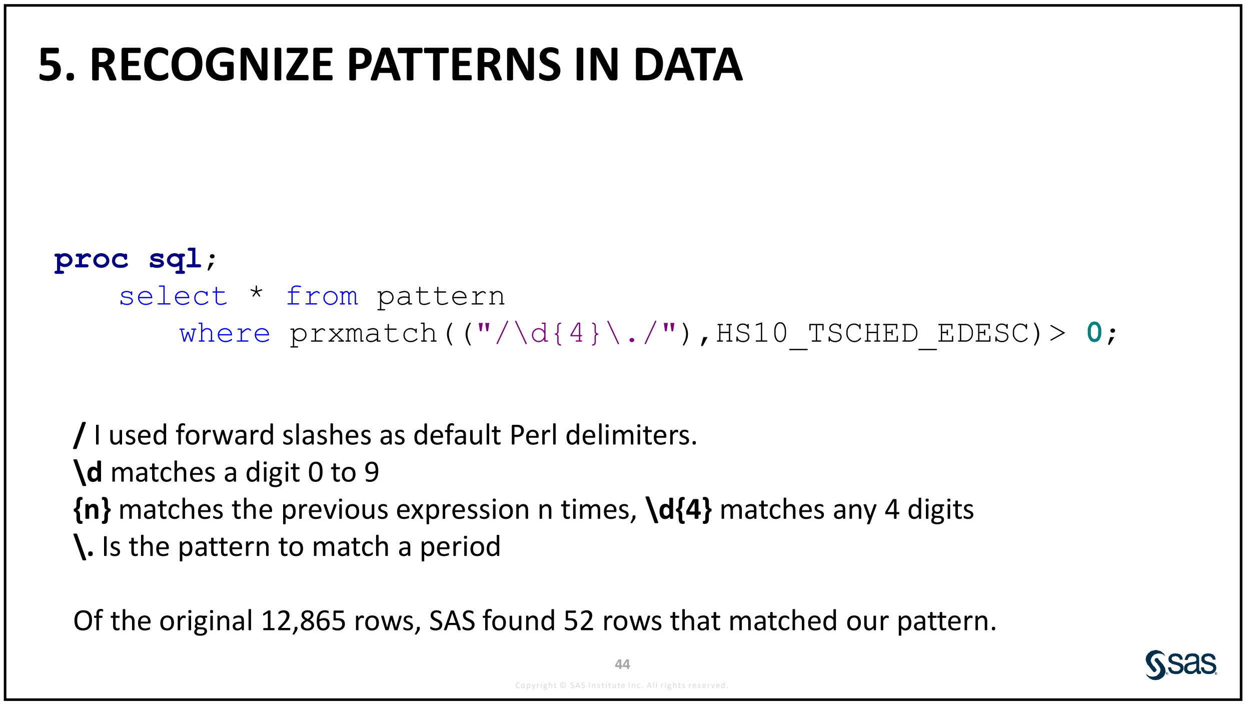


5. FIND YOUR DATA PATTERN

Business Scenario

Find data that matches a pattern. HS10_ column has a series of any 10 or 6 digit numbers. An additional challenge- this series never appears in the same position”.

Sample of the data
Sticks or profile shapes of subheading 3916.10
Reproduction proofs for the production of printing plates, rolls, of tariff item No. 8442.50.20
Microcopies of tariff item No. 4903.00.10, 4905.91.00, 4911.10.10 or 4911.10.20



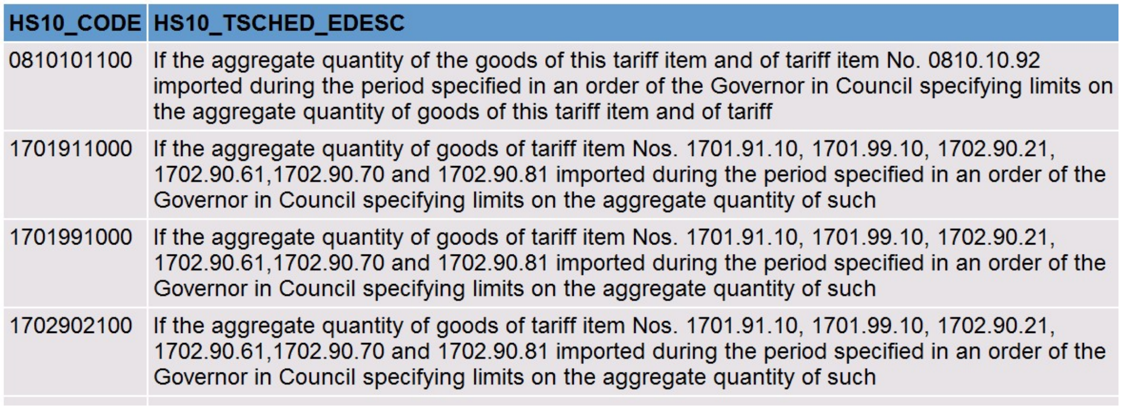
5. RECOGNIZE PATTERNS IN DATA

```
proc sql;
    select * from pattern
       where prxmatch(("/\d{4}\./"),HS10_TSCHED_EDESC)> 0;
```

/ I used forward slashes as default Perl delimiters.
\d matches a digit 0 to 9
{n} matches the previous expression n times, **\d{4}** matches any 4 digits
\. Is the pattern to match a period

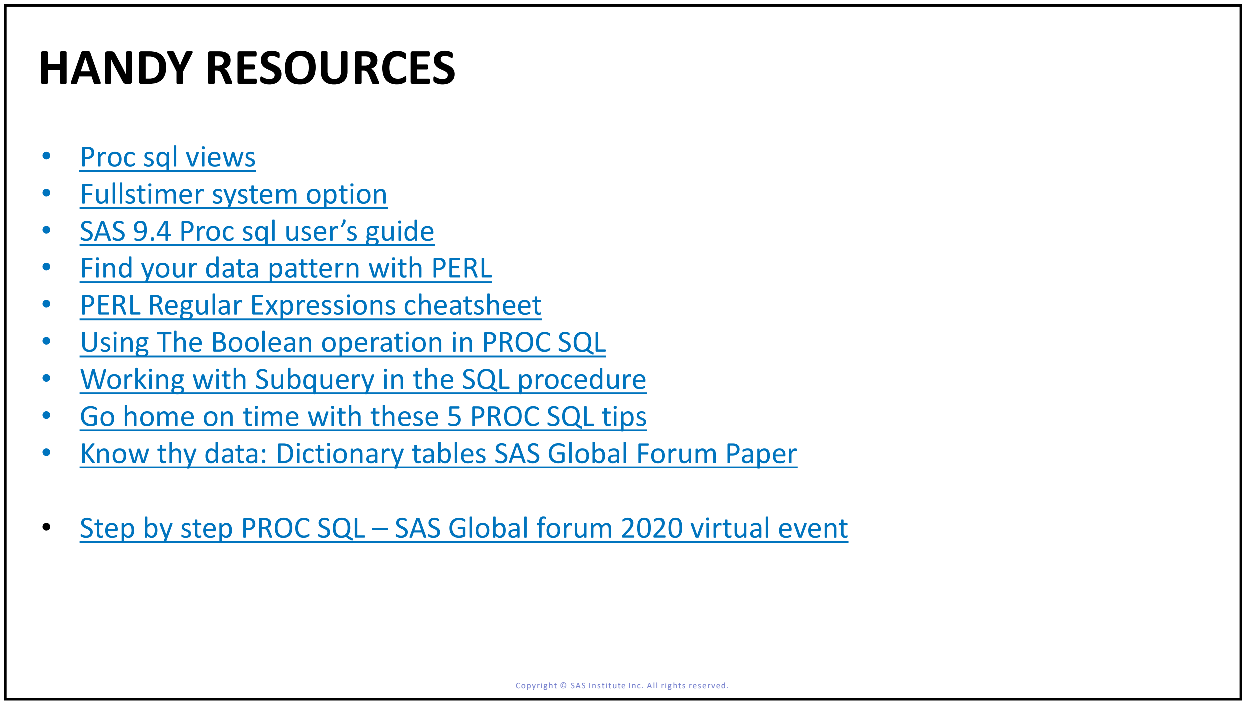
Of the original 12,865 rows, SAS found 52 rows that matched our pattern.

5. RECOGNIZE PATTERNS IN DATA



HS10_CODE	HS10_TSCHED_EDESC
0810101100	If the aggregate quantity of the goods of this tariff item and of tariff item No. 0810.10.92 imported during the period specified in an order of the Governor in Council specifying limits on the aggregate quantity of goods of this tariff item and of tariff
1701911000	If the aggregate quantity of goods of tariff item Nos. 1701.91.10, 1701.99.10, 1702.90.21, 1702.90.61,1702.90.70 and 1702.90.81 imported during the period specified in an order of the Governor in Council specifying limits on the aggregate quantity of such
1701991000	If the aggregate quantity of goods of tariff item Nos. 1701.91.10, 1701.99.10, 1702.90.21, 1702.90.61,1702.90.70 and 1702.90.81 imported during the period specified in an order of the Governor in Council specifying limits on the aggregate quantity of such
1702902100	If the aggregate quantity of goods of tariff item Nos. 1701.91.10, 1701.99.10, 1702.90.21, 1702.90.61,1702.90.70 and 1702.90.81 imported during the period specified in an order of the Governor in Council specifying limits on the aggregate quantity of such

FINDING PATTERNS DEMONSTRATION

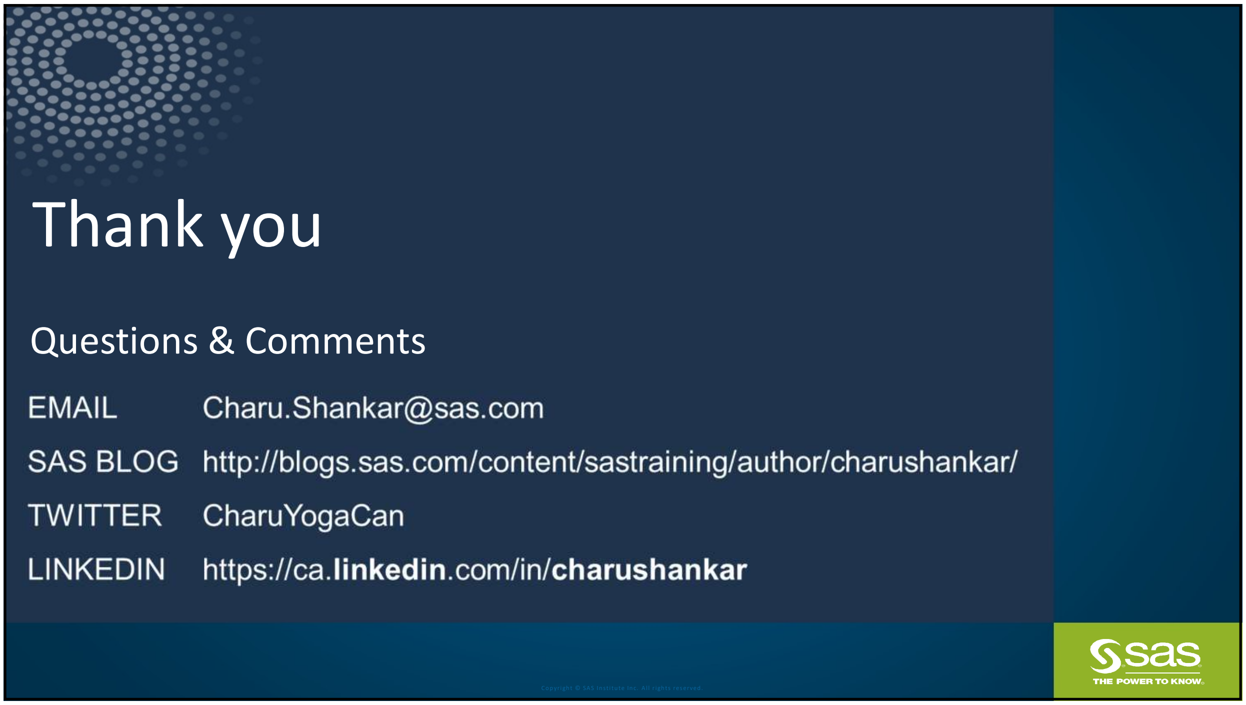


HANDY RESOURCES

- Proc sql views
- Fullstimer system option
- SAS 9.4 Proc sql user's guide
- Find your data pattern with PERL
- PERL Regular Expressions cheatsheet
- Using The Boolean operation in PROC SQL
- Working with Subquery in the SQL procedure
- Go home on time with these 5 PROC SQL tips
- Know thy data: Dictionary tables SAS Global Forum Paper

- Step by step PROC SQL – SAS Global forum 2020 virtual event

47



Thank you

Questions & Comments

EMAIL Charu.Shankar@sas.com

SAS BLOG http://blogs.sas.com/content/sastraining/author/charushankar/

TWITTER CharuYogaCan

LINKEDIN https://ca.**linkedin**.com/in/**charushankar**

48