Step-by-Step SQL Procedure

U Michigan

January 2023

Charu Shankar

SAS Institute, Toronto

Step-by-Step PROC SQL

Instructor Bio

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 taught at several SAS international user group conferences on SAS programming, SAS Enterprise Guide, PROC SQL, DS2 programming, Viya, Python, tips and tricks, efficiencies.

SAS blog: https://blogs.sas.com/content/author/charushankar/

Agenda

1. Understand the Syntax Order in Which to Submit Queries to PROC SQL
2. Summarize Data Using Boolean Operations
3. Manage Metadata Using DICTIONARY Tables
4. Join Tables Using Inner Join and Reflexive Join Conditions
5. Q & A

1. Understand the Syntax Order in Which to Submit Queries to PROC SQL

1.1 Overview of the SQL Procedure

SQL Procedure

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

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.

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

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 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.

PROC SQL Syntax Order

This demonstration illustrates the syntax order of PROC SQL

s101e01.sas
s101e02.sas
s101e03.sas
s101e04.sas

2: Summarizing Data Using Boolean Operations

2.1 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    Administration Manager
Administration    Secretary I
Administration    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
```

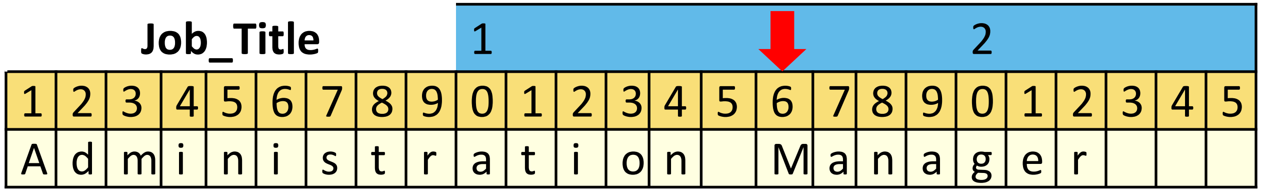
Use the FIND function in a Boolean expression to identify rows that contain *Manager* in the **Job_Title** column.

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					
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>*)

Using Boolean Expressions

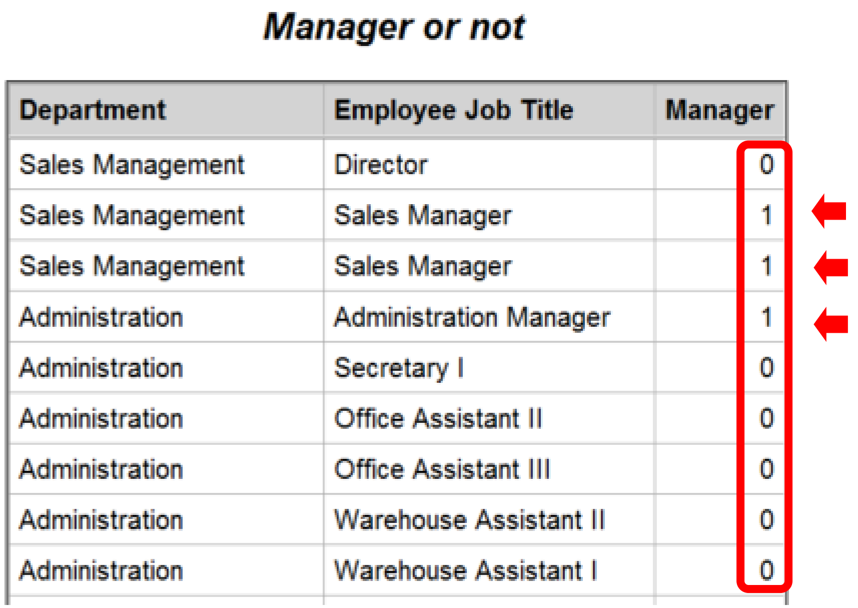
Part 1: Use a Boolean expression to determine whether an employee is a manager.

```
title 'Manager or not';
proc sql;
select Department, Job_Title,
   (find(Job_Title,"manager","i")>0) "Manager"
   from Umich.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



Manager or not

Department	Employee Job Title	Manager
Sales Management	Director	0
Sales Management	Sales Manager	1
Sales Management	Sales Manager	1
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

Using Boolean Expressions

Part 2: Calculate the statistics requested.

```
title "Manager-to-Employee Ratios";
proc sql;
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 Umich.employee_information
   group by Department;
quit;
```

s102e05

Summarizing Data Using Boolean Operations

s102e01.sas
s102e02.sas
s102e03.sas
s102e04.sas
s102e05.sas

Section 3: Manage Metadata Using DICTIONARY Tables

3.1 DICTIONARY Tables and Views

Manage Metadata Using DICTIONARY Tables

s103e01.sas
s103e02.sas
s103e03.sas
s103e04.sas
s103e05.sas

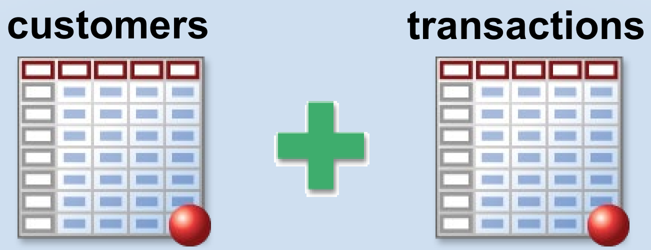
Section 4: Join Tables Using Inner Join and Reflexive Join Conditions

4.1 SQL Joins: Inner Joins

4.2 SQL Joins: Reflexive (self) Joins

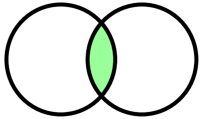
Combining Data from Multiple Tables

SQL uses *joins* to combine tables horizontally. Requesting a join involves matching data from one row in one table with a corresponding row in a second table. Matching is typically performed on one or more columns in the two tables.

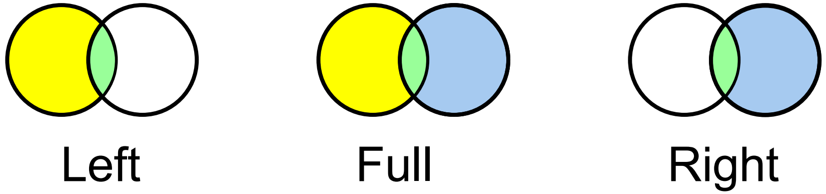


Types of Joins

PROC SQL supports two types of joins: *Inner joins* return only matching rows.



Outer joins return all matching rows, plus nonmatching rows from one or both tables.



Cartesian Product

A query that lists multiple tables in the FROM clause without a WHERE clause produces all possible combinations of rows from all tables. This result is called a *Cartesian product*.

```
title 'Combining data from multiple tables';
proc sql;
select *
   from Umich.customers, Umich.transactions;
quit;
```

SELECT …
FROM *table-name*, *table-name*
<, …,*table-name* >**;**

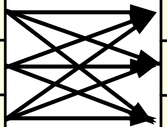
To understand how SQL processes a join, it is helpful to understand the concept of the Cartesian product.

s104e01

Nonmatching Data in the Cartesian Product

customers

ID	Name
101	Smith
104	Jones
102	Blank



transactions

ID	Action	Amount
102	Purchase	$100
103	Return	$52
105	Return	$212

Result Set

```
 ID   Name     ID   Action        Amount
________________________________________
101   Smith   102   Purchase        $100
101   Smith   103   Return           $52
101   Smith   105   Return          $212
104   Jones   102   Purchase        $100
104   Jones   103   Return           $52
104   Jones   105   Return          $212
102   Blank   102   Purchase        $100
102   Blank   103   Return           $52
102   Blank   105   Return          $212
```

Nonmatching IDs

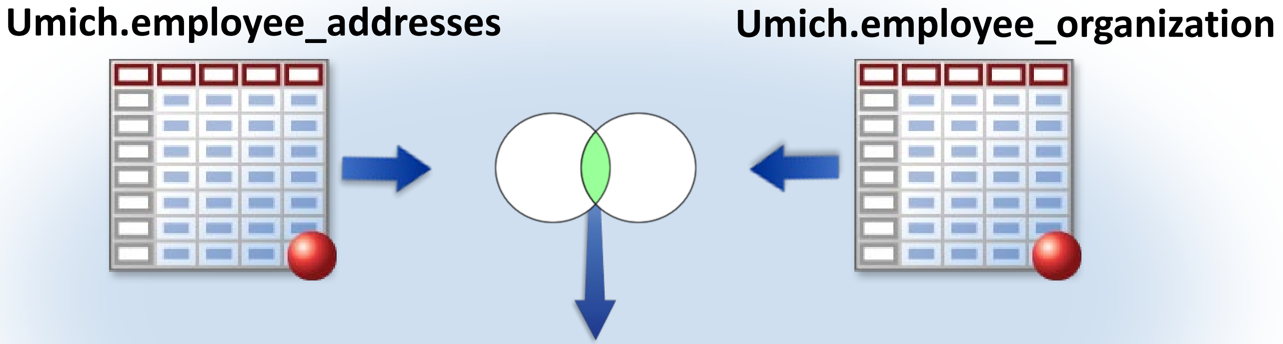
9 rows

The Cartesian product is rarely the desired result of a query.

Section 4: Join Tables Using Inner Join and Reflexive Join Conditions

Business Scenario

The chief sales officer wants to have a report with the name of all sales employees and the name of each employee’s direct manager.



```
Employee ID  Employee Name      Manager ID   Manager Name    Country
_____________________________________________________________________
     120140  Minas, Michael         120103   Dawes, Wilson   AU
     120141  Liebman, Amanda        120103   Dawes, Wilson   AU
     120142  Eastley, Vincent       120103   Dawes, Wilson   AU
     120143  Sloey, Phu             120103   Dawes, Wilson   AU
     120144  Barbis, Viney          120103   Dawes, Wilson   AU
```

Business Data

To return the employee name and the manager name, you need to read the **addresses** table twice.

1. Return the employee's ID and name.

addresses

EMP_ID	EMP_NAME
100	John
101	Sue

organization

EMP_ID	MGR_ID
100	101
101	57

EMP_ID	EMP_Name	MGR_ID	MGR_Name
100	John		

...

Business Data

To return the employee name and the manager name, you need to read the **addresses** table twice.

1. Return the employee's ID and name.
2. Determine the ID of the employee's manager.

addresses

EMP_ID	EMP_NAME
100	John
101	Sue

organization

EMP_ID	MGR_ID
➡ 100	101
101	57

EMP_ID	EMP_Name	MGR_ID	MGR_Name
100	John	101	

...

Business Data

To return the employee name and the manager name, you need to read the **addresses** table twice.

1. Return the employee's ID and name.
2. Determine the ID of the employee's manager.
3. Return the manager's name.

addresses

EMP_ID	EMP_NAME
100	John
➡ 101	Sue

organization

EMP_ID	MGR_ID
100	101
101	57

EMP_ID	EMP_Name	MGR_ID	MGR_Name
100	John	101	Sue

Required Table Aliases

In order to read from the same table twice, it must be listed in the FROM clause twice. Here, a different table alias is required to distinguish the different uses.

```
from Umich.employee_addresses as e,
     Umich.employee_addresses as m,
```

FROM *table-name-1* <**AS**> *alias-1*,
table-name-1 <**AS**> *alias-2*

Thanks for Attending
Questions?

Email	Charu.Shankar@sas.com
Linkedin	www.linkedin.com/in/charushankar/
Twitter	charuyogacan
Blog	blogs.sas.com/content/sastraining/author/charushankar