

Five Secrets of The SQL Goddess

Charu Shankar, SAS® Institute Inc.

ABSTRACT

PROC SQL is a powerful language that can express many of your queries simply and with clarity. Users who are continuously improving process and looking to stay within PROC SQL to analyze and process data will benefit from this HOW. Come learn to maximize human & computing efficiency elegantly. Readers will learn to join tables dynamically, Create Inline views for joins, Pivot like a dancer with the Boolean, recognize patterns in data and to create running totals. This paper is aimed at all levels of learning.

INTRODUCTION

PROC SQL is the language of databases. After teaching at SAS for over 15 years to thousands of learners, Charu got an awesome compliment when a user ordained her 'The SQL Goddess' as he had found her instruction via user groups, white papers, YouTube videos instrumental in his SQL learning. This paper was thus titled tongue in cheek. Though the intent to secrets and best practices still remains the purpose of this paper.

SQL OVERVIEW

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

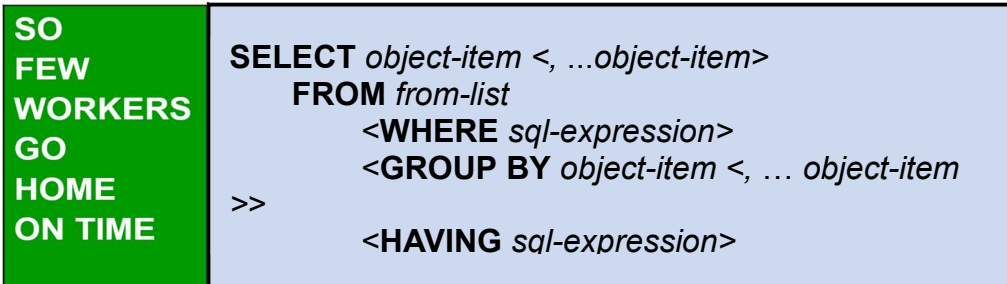


Figure 1: PROC SQL Mnemonic

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.

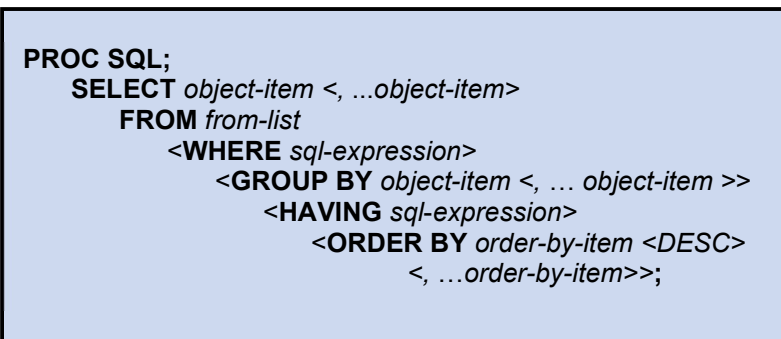


Figure 2: PROC SQL Syntax order

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.

1 JOIN TABLES DYNAMICALLY

One of the questions I get asked a lot in the SQL classroom is this. 'How can I join my tables dynamically with the least bit of sweat'? I hear you dear user/learner. Who wants to go poking around in each table finding out the columns that exist and then the whole manual process of exploring & trying to find other tables with matching columns?

I will certainly share my secret here. And will begin with an admonition.

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 technique I simply must make. I LOVE dictionary tables and cannot imagine life without them. When you see this secret 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

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;
```

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

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.

23

s108d01



FINDING SAME NAMED COLUMNS

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
            uppercase(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.

27

s108d01



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
EISMKN	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.



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 secret #1? 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;
```

Viewing the Output

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_CITYYN	ZIPCODE	char	300
ALIAS_CITYYN	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.

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

```

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.

2 CREATE INLINE VIEWS FOR JOINS

Joins are easily one of the most complex actions that SQL performs. I'm all about keeping life simple, and to reduce the complexity of a join, I give you an inline view.

An inline view? Is that a view? No, it's a separate entity. It's a chunk of code that replaces a table on the FROM clause. An in-line view is a query expression(SELECT statement) that sits in a FROM clause. It acts as a virtual table, used in place of a physical table in a query.

Let me show you the business scenario/code and then we will discuss the and benefits of an Inline view.

Business has asked us to find out who makes less than the average for their job titles. We translate that to a coding statement. We want to list all active Sales department employees who have annual salaries less than 95% of the average for everyone with the same job title.

Step 1 Code to calculate job averages

```

title 'Sales Department Average Salary';
title2 'By Job Title';
proc sql;
  select Job_Title, avg(Salary) as Job_Avg
  from orion.employee_payroll as p,
  orion.employee_organization as o
  where p.Employee_ID=o.Employee_ID
  and Employee_Term_Date is missing
  and Department="Sales"
  group by Job_Title;
quit;

```

Sales Department Average Salary By Job Title	
Job_Title	Job_Avg
Sales Rep. I	26575.76
Sales Rep. II	27347.97
Sales Rep. III	29213.62
Sales Rep. IV	31588.5

Figure 3: PROC SQL Output

Code to compare employee' salary to the group's average

```

proc sql;
  select Employee_Name, emp.Job_Title, Salary
  format=comma7., Job_Avg format=comma7.
  from (select Job_Title, avg(Salary) as Job_Avg
  format=comma7.
  from orion.employee_payroll as p,
  orion.employee_organization as o
  where p.Employee_ID=o.Employee_ID
  and Employee_Term_Date is missing
  and Department="Sales"
  group by Job_Title) as job, orion.salesstaff as emp
  where emp.Job_Title=job.Job_Title
  and Salary < Job_Avg*.95 and Emp_Term_Date is missing
  order by Job_Title, Employee_Name;
quit;

```

The inline view is nothing other than step 1 sitting on the FROM clause

**Sales Department Average Salary
By Job Title**

Employee_Name	Employee Job Title	Employee Annual Salary	Job_Avg
Ould, Tulsidas	Sales Rep. I	22,710	26,576
Polky, Asishana	Sales Rep. I	25,110	26,576
Voron, Tachaun	Sales Rep. I	25,125	26,576

Figure 4: PROC SQL Output

There you have it. We were able to combine average values with the detailed rows of data, without having to build a separate table for the average.

Do you see the efficiency of an inline view, saving valuable I/O as we skipped building a table and one step in this process?

3 PIVOT LIKE A DANCER WITH THE BOOLEAN

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 to learn more about this magic.

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.

GROUPING DATA

Let's begin with a simple business scenario to understand grouping first. We have been asked to produce a report that determines the average salary by gender.

Code to calculate average salary

```

proc sql number;
  select Employee_Gender, avg(Salary) as Average
  from SGF2020.employee_information
  where Employee_Term_Date is missing;
quit;

```

Figure 5: Code for average salary by gender

The result is not quite as expected. Instead of receiving 2 rows of data, the output contains 308 rows. This is the number of rows in the SGF2020.employee_information table. Also, the Average is not an average for each gender, rather the average for the entire table.

Viewing the Output

PROC SQL Output

Is this average salary by gender

Row	Employee Gender	Average
1	M	40478.92
2	M	40478.92
3	M	40478.92
4	F	40478.92
5	F	40478.92
6	M	40478.92
7	F	40478.92
8	F	40478.92
9	F	40478.92
10	M	40478.92
11	M	40478.92
12	F	40478.92

sas

Figure 6: Unexpected Output for average salary by gender

THE GROUP BY CLAUSE

You can use the GROUP BY clause to do the following:

- classify the data into groups based on the values of one or more columns
- calculate statistics for each unique value of the grouping columns

Code to calculate average salary by gender

```
Title "Average Salary by Gender";
proc sql;
  select Employee_Gender as Gender, avg(Salary) as Average
  from SGF2020.employee_information
  where Employee_Term_Date is missing
  group by Employee_Gender;
quit;
```

The results are more satisfactory this time, with 2 rows of data.

Viewing the Output

PROC SQL Output

Average Salary by Gender	
Employee Gender	Average
F	37002.88
M	43334.26

sas

Figure 7: Correct Output for average salary by gender

Let's move on to the next level of complexity. We have been tasked to produce a report showing the count of employees in departments that have at least 25 people. Display the results in descending order by count.

A first step would be to count the number of employees for each department.

```
Title 'Employee count by department';
proc sql;
  select Department, count(*) as Count
  from SGF2020.employee_information
  group by Department;
quit;
```

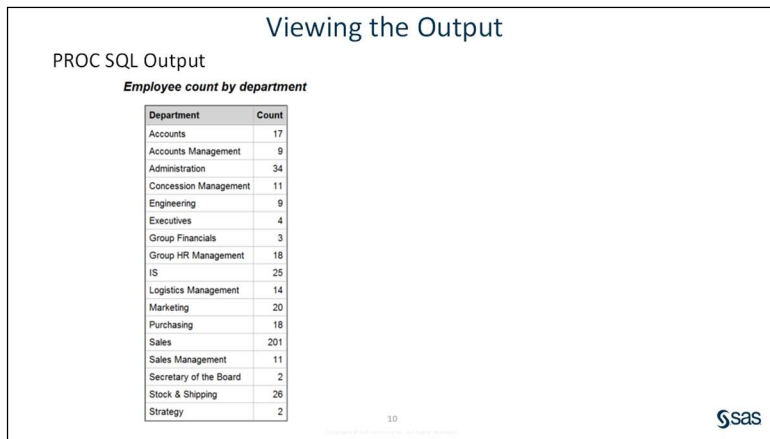


Figure 8: Employee Counts by Department

In the next step, we will control the result to include only the departments that have at least 25 people, with the departments in decreasing order. To do this, we will utilize the Having Clause. The *HAVING clause* subsets groups based on the expression value.

```
Title 'Employee counts by department in departments with at least 25
employees';
proc sql;
  select Department, count(*) as Count
  from SGF2020.employee_information
  group by Department
  having Count ge 25
  order by Count desc;
quit;
```

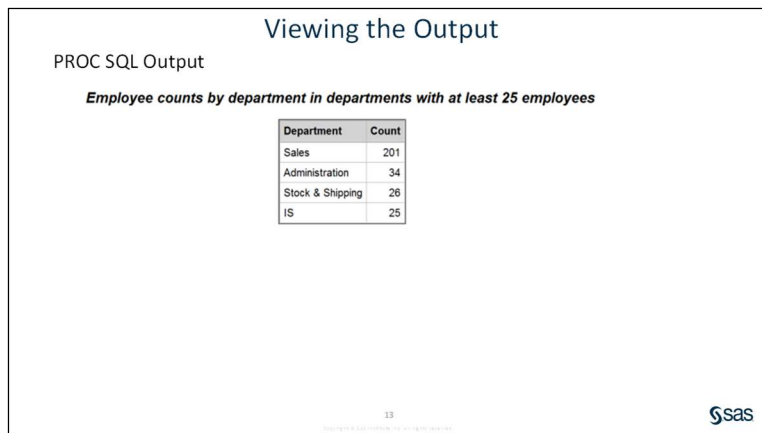


Figure 9: Employee Counts by Department with at least 25 Employees

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

Business Scenario 3

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%




Figure 10: Business Scenario for total number of managers and employees

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 the FIND Function

```
title 'Manager or not';
proc sql;
  select Department, Job_Title,
         (find(Job_Title,"manager","i")>0) "Manager"
  from SGF2020.employee_information;
quit;
```

Code to write a Boolean expression

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

```
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 SGF2020.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	8	68.3%
Secretary of the Board	0	2	0.0%
Stock & Shipping	5	21	23.8%
Strategy	0	2	0.0%




Figure 12: Output using Boolean Operations

This was just one way to use the Boolean. The expressions can be as complex as necessary.

4 RECOGNIZE PATTERNS IN YOUR DATA

Pattern Recognition is a frequent ask of SAS programmers. My secret is one that doesn't involve any statistical functions. Rather a free, easy to use function available in your SAS software.

USING PERL TO MATCH A PATTERN

Perl was designed specifically for text processing. Perl is a very high-level language. That means that the code is quite dense. A Perl program might be around 30% to 70% as long as the corresponding program in C.

PERL IN SAS

Perl regular expressions were added to SAS in Version 9. SAS regular expressions (similar to Perl regular expressions but using a different syntax to indicate text patterns) have been around since version 6.12, but many SAS users are unfamiliar with either SAS or Perl regular expressions.

Because SAS already has such a powerful set of string functions, you might wonder why you need regular expressions. Many of the string processing tasks can be performed either with the traditional character functions or regular expressions. However, regular expressions can sometimes provide a much more compact solution to a complicated string manipulation task.

MATCHING A PATTERN

Since the backslash, forward slash, parentheses and several other characters have special meaning in a regular expression, you may wonder, how do you search a string for a \ character or a left or right parenthesis? You do this by preceding any of these special characters with a backslash character (in Perl jargon called an escape character). So, to match a \ in a string, you code two backslashes like this: \\ . To match an open parenthesis, you use \(.

/ delimiters

\(matches an open parenthesis

\D matches a non-digit

\d matches a digit

\s matches a space

{n,m} Matches the previous subexpression n or more times, but no more than m

\) matches a closed parenthesis

Here's the Business Problem

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

Code to find a pattern using PERL

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

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

Figure 13: Output using PERL Regular Expression

5 CREATE RUNNING TOTALS

There comes a time in life when you simply must create a running total, and you don't want to leave the comfort of your PROC SQL environment. You know the heavy duty data step can do this simply, but you just want to do it in PROC SQL.

My next secret may just satisfy that yearning.

Here's the business scenario:

Calculate running totals for shoe sales

Running total for shoe sales

Region	Product	Subsidiary	Total Sales	Running_total
Africa	Boot	Addis Ababa	\$29,761	\$29,761
Africa	Men's Casual	Addis Ababa	\$67,242	\$97,003
Africa	Men's Dress	Addis Ababa	\$76,793	\$173,796
Africa	Sandal	Addis Ababa	\$62,819	\$236,615
Africa	Slipper	Addis Ababa	\$68,641	\$305,256
Africa	Sport Shoe	Addis Ababa	\$1,690	\$306,946
Africa	Women's Casual	Addis Ababa	\$51,541	\$358,487
Africa	Women's Dress	Addis Ababa	\$108,942	\$467,429
Africa	Boot	Algiers	\$21,297	\$488,726
Africa	Men's Casual	Algiers	\$63,206	\$551,932

Figure 14: Expected Output

Code to create running total First attempt

```
data shoes;
  set sashelp.shoes;
  /* Give an index for each row in the shoes dataset*/
  uniq=_n_;
run;
proc sql;
  select region, product, sales,
         (select sum(a.sales) from shoes as a
          where a.uniq <= b.uniq) as Running_total
  from shoes as b;
quit;
```

Here we first passed through the data step to create a unique row number. Remember SQL works at the table level and doesn't really have an intelligent way to capture row number. There is of course the undocumented monotonic function, but we would rather stick with the known and hence the data step first.

Then in Proc sql, calculate the running total using a subquery on the Select.

If, however a unique column already exists in your dataset, then you can do it all in one step as below.

Code to create running total Second attempt

```
proc sql;
  select a.order_date, a.quantity,
         (select SUM(b.quantity) from running.order_fact as b
          where b.order_date <= a.order_date) as Running_total
  from running.order_fact as a
  order by a.order_date;
quit;
```

CONCLUSION

This paper attempted to showcase the best strengths of PROC SQL and lay out these strengths secret-by-secret. The author has used her teaching and consulting experiences to highlight those tips that are very unique to PROC SQL.

ACKNOWLEDGEMENTS

The author is grateful to the many SAS users that have entered her life. Charu is grateful to the Western Users of SAS Software 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.

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.

REFERENCES

SAS 9.4 Proc sql user's guide

<https://go.documentation.sas.com/?docsetId=sqlproc&docsetTarget=titlepage.htm&docsetVersion=9.4&locale=en>

Logical Query Processing Order

A Database Professional's Best Friend, Shankar, Charu

<https://blogs.sas.com/content/sastraining/2013/02/04/a-database-professionals-best-friend-2/>

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/>

PROC SQL Dictionary Tables

"Know thy data: Techniques for Data Exploration", Shankar, Charu

<https://www.pharmasug.org/proceedings/2018/BB/PharmaSUG-2018-BB11.pdf>

"Working with Subquery in the SQL procedure" Zhang, Lei. Yi, Danbo

<https://www.lexjansen.com/nesug/nesug98/dbas/p005.pdf>

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/>

The Power of SAS SQL YouTube Video

SAS Instructor Demos From SAS Global Forum 2021

<https://youtu.be/BOrk-qY4xjk>

SAS Tutorial | Step-by-Step PROC SQL

SAS Users Youtube Channel

<https://youtu.be/1xyHE8qI9Hk>