

# SQL and Macro 101

University of Iowa SAS Users Group  
October 3, 2025

Rebecca Callaway  
SAS Education

Copyright © SAS Institute Inc. All rights reserved.



# SQL and Macro 101

Rebecca Callaway, SAS® Institute

With a background in Mathematics and Statistics, SAS Instructor Rebecca Callaway engages with logic, visuals, and analogies to spark critical thinking since 2000.

Rebecca teaches classes on SAS programming, SQL, SAS Visual Analytics, SAS Viya, etc. to support users in the adoption of SAS software.

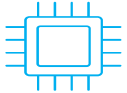
When not working, Rebecca enjoys spending time outdoors enjoying the lovely San Diego weather and hanging out with her husband Ken and their cat Zigmo.



# Agenda



Nuts & Bolts - PROC SQL Overview



Specifying Columns



Specifying Rows



Summarizing Data



Joining Tables



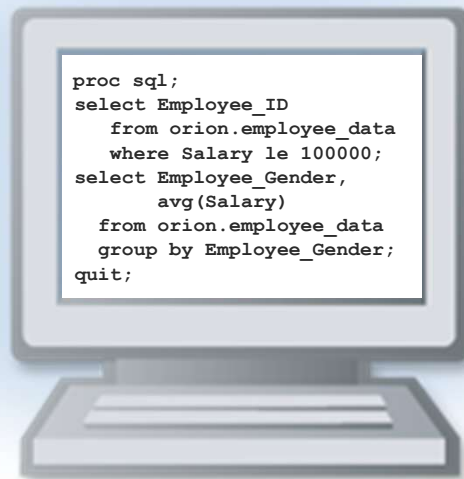
Introduction to Macro

# Nuts & Bolts - PROC SQL Overview

# Structured Query Language

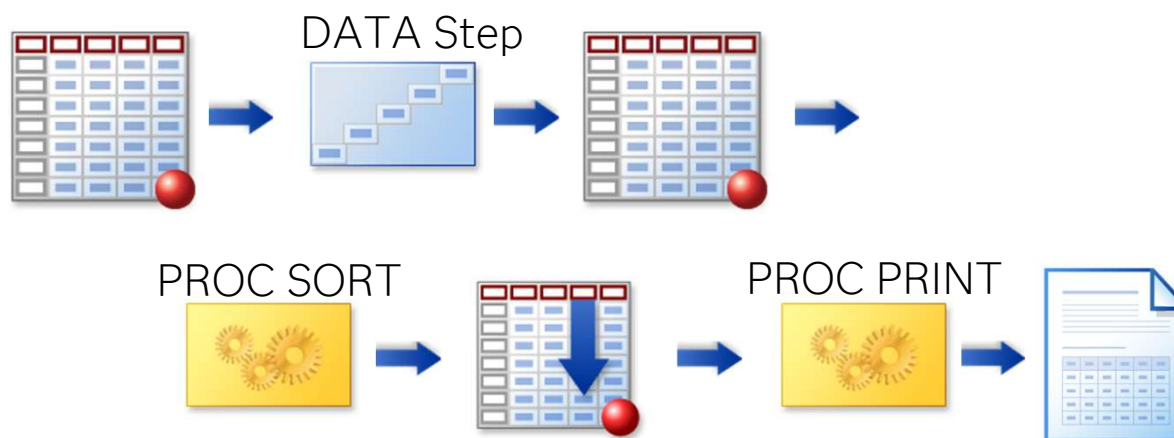
*Structured Query Language* (SQL) is a standardized language originally designed as a relational database query tool.

SQL is currently used in many software products to retrieve and update data.



# SQL Procedure versus Traditional SAS

The SQL procedure can sometimes reproduce the results of multiple DATA and procedure steps with a single query.



# Objectives

- Identify key syntax of the SQL procedure.
- List key features of the SQL procedure.
- List key features of the SELECT statement.
- List SQL procedure statements.

# SELECT Statement

A SELECT statement contains smaller building blocks called *clauses*.

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

clauses

 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;
```

- The SELECT clause specifies the columns and column order.
- The FROM clause specifies the data sources.
- You can query from 1 to 256 tables.

## SELECT Statement Syntax

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



The specified order of the above clauses within the SELECT statement is required.



# Specifying Columns

# Objectives

- Explore unfamiliar data.
- Display columns directly from a table.
- Display columns calculated from other columns in a query.

# Querying All Columns in a Table

To print all of a table's columns in the order in which they were stored, specify an asterisk in a SELECT clause.

```
proc sql;  
select *  
  from orion.employee_information;  
quit;
```

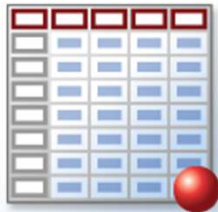
Partial PROC SQL Output

| The SAS System |            |           |                  |                        |                     |                    |                           |                      |
|----------------|------------|-----------|------------------|------------------------|---------------------|--------------------|---------------------------|----------------------|
| Employee ID    | Start Date | End Date  | Department       | Employee Annual Salary | Employee Birth Date | Employee Hire Date | Employee Termination Date | Manager for Employee |
| 120101         | 01JUL2007  | 31DEC9999 | Sales Management | \$163,040              | 18AUG1980           | 01JUL2007          | .                         | 120261               |
| Director       |            |           |                  |                        |                     |                    |                           |                      |

# Business Scenario

Produce a report that contains selected information for all Orion Star employees.

orion.employee\_information



PROC SQL



| Employee_ID | Employee_<br>Gender | Salary |
|-------------|---------------------|--------|
| 120101      | M                   | 163040 |
| 120102      | M                   | 108255 |
| 120103      | M                   | 87975  |
| 120104      | F                   | 46230  |
| 120105      | F                   | 27110  |

# Querying Specific Columns in a Table

List the columns that you want and the order to display them in the SELECT clause.

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

# Viewing the Output

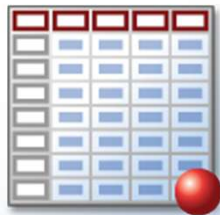
Partial PROC SQL Output

| The SAS System |                 |                        |
|----------------|-----------------|------------------------|
| Employee ID    | Employee Gender | Employee Annual Salary |
| 120101         | M               | \$163,040              |
| 120102         | M               | \$108,255              |
| 120103         | M               | \$87,975               |
| 120104         | F               | \$46,230               |
| 120105         | F               | \$27,110               |
| 120106         | M               | \$26,960               |
| 120107         | F               | \$30,475               |

# Business Scenario

Modify the previous report by creating a new column, **Bonus**, which contains an amount equal to 10% of the employee's salary.

orion.employee\_information



PROC SQL



| Employee_ID | Salary | Bonus   |
|-------------|--------|---------|
| 120101      | 163040 | 16304   |
| 120102      | 108255 | 10825.5 |
| 120103      | 87975  | 8797.5  |
| 120104      | 46230  | 4623    |
| 120105      | 27110  | 2711    |

# Calculated Columns

Name the new column using the AS keyword.

```
proc sql;  
select Employee_ID, Salary,  
       Salary*.10 as Bonus  
       from orion.employee_information;  
quit;
```

Partial PROC SQL Output

| The SAS System |                              |         |
|----------------|------------------------------|---------|
| Employee ID    | Employee<br>Annual<br>Salary | Bonus   |
| 120101         | \$163,040                    | 16304   |
| 120102         | \$108,255                    | 10825.5 |
| 120103         | \$87,975                     | 8797.5  |
| ...            |                              |         |



# Specifying Rows

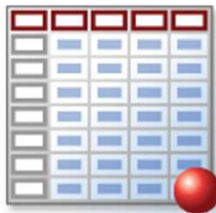
# Objectives

- Select a subset of rows in a query.

# Business Scenario

Management requested a list of employees whose salaries exceed \$112,000.

orion.employee\_information



PROC SQL



| Employee ID | Employee Job Title      | Employee Annual Salary |
|-------------|-------------------------|------------------------|
| 120101      | Director                | \$163,040              |
| 120259      | Chief Executive Officer | \$433,800              |
| 120260      | Chief Marketing Officer | \$207,885              |
| 120261      | Chief Sales Officer     | \$243,190              |
| 120262      | Chief Financial Officer | \$268,455              |

# Subsetting with the WHERE Clause

Use a WHERE clause to specify a condition that the data must satisfy before being selected.

```
proc sql;  
select Employee_ID, Job_Title, Salary  
       from orion.employee_information  
       where Salary > 112000;  
quit;
```

**WHERE *sql-expression***

# Viewing the Output

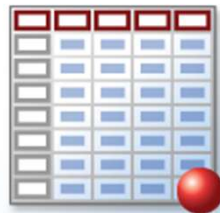
## PROC SQL Output

| The SAS System |                         |  | Employee<br>Annual<br>Salary |
|----------------|-------------------------|--|------------------------------|
| Employee ID    | Employee Job Title      |  |                              |
| 120101         | Director                |  | \$163,040                    |
| 120259         | Chief Executive Officer |  | \$433,800                    |
| 120260         | Chief Marketing Officer |  | \$207,885                    |
| 120261         | Chief Sales Officer     |  | \$243,190                    |
| 120262         | Chief Financial Officer |  | \$268,455                    |
| 120659         | Director                |  | \$161,290                    |
| 121141         | Vice President          |  | \$194,885                    |
| 121142         | Director                |  | \$156,065                    |

# Business Scenario

Management requested a report that includes only those employees who receive bonuses less than \$3000.

orion.employee\_information



PROC SQL



| Employee_ID | Employee_<br>Gender | Salary | Bonus  |
|-------------|---------------------|--------|--------|
| 120105      | F                   | 27110  | 2711   |
| 120106      | M                   | 26960  | 2696   |
| 120108      | F                   | 27660  | 2766   |
| 120109      | F                   | 26495  | 2649.5 |
| 120110      | M                   | 28615  | 2861.5 |

# Subsetting with Calculated Values

First attempt:

```
proc sql;  
select Employee_ID, Employee_Gender,  
       Salary, Salary*.10 as Bonus  
from orion.employee_information  
where Bonus<3000;  
quit;
```

A *WHERE* clause is evaluated before the *SELECT* clause. Therefore, columns used in the *WHERE* clause must exist in the table.

Partial SAS Log

```
ERROR: The following columns were not found in the contributing  
tables: Bonus.
```

# Subsetting with Calculated Values

An alternate method is to use the CALCULATED keyword in the WHERE clause.

```
proc sql;  
select Employee_ID, Employee_Gender,  
       Salary, Salary*.10 as Bonus  
  from orion.employee_information  
 where calculated Bonus<3000;  
quit;
```



SAS enhancement

# Viewing the Output

Partial PROC SQL Output

| The SAS System |                 |                        |        |
|----------------|-----------------|------------------------|--------|
| Employee ID    | Employee Gender | Employee Annual Salary | Bonus  |
| 120105         | F               | \$27,110               | 2711   |
| 120106         | M               | \$26,960               | 2696   |
| 120108         | F               | \$27,660               | 2766   |
| 120109         | F               | \$26,495               | 2649.5 |
| 120110         | M               | \$28,615               | 2861.5 |
| 120111         | M               | \$26,895               | 2689.5 |
| 120112         | F               | \$26,550               | 2655   |



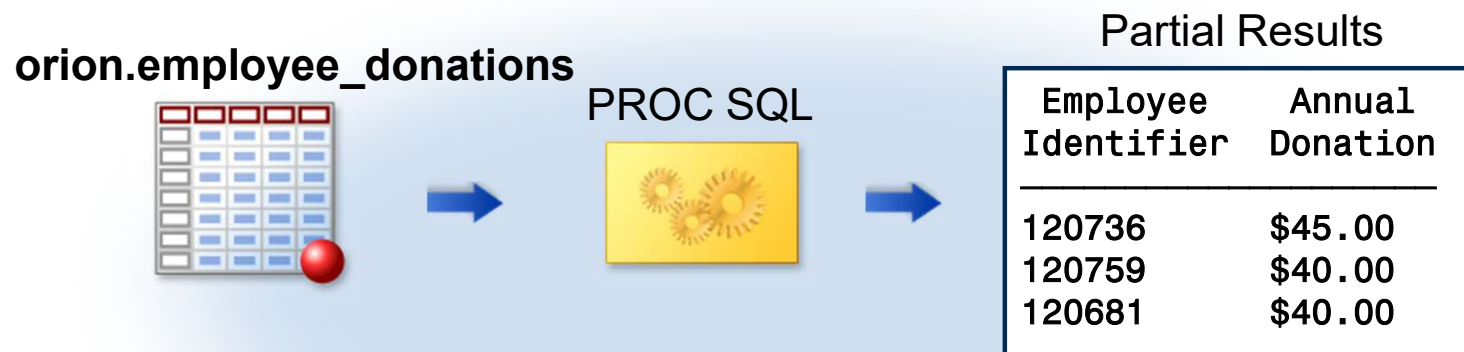
# Summarizing Data

# Objectives

- Group data and produce summary statistics for each group.
- Subset a query on summarized values.

# Business Scenario

Management requested a report containing the total annual donations for each employee.



# Summary Functions: Across a Row

Total each employee's annual cash donations.

**SUM(col1, ..., coln)**

```
proc sql;  
select Employee_ID  
       label='Employee Identifier',  
       Qtr1,Qtr2,Qtr3,Qtr4,  
       sum(Qtr1,Qtr2,Qtr3,Qtr4)  
       label='Annual Donation'  
       format=dollar5.  
from orion.employee_donations  
where Paid_By="Cash or Check"  
order by 6 desc;  
quit;
```

# Viewing the Output

Partial PROC SQL Output

| Employee Identifier | Qtr1 | Qtr2 | Qtr3 | Qtr4 | Annual Donation |
|---------------------|------|------|------|------|-----------------|
| 120736              | 25   | .    | .    | 20   | \$45            |
| 120759              | 15   | 20   | 5    | .    | \$40            |
| 120681              | 10   | 10   | 5    | 15   | \$40            |
| 120679              | .    | 20   | 5    | 15   | \$40            |
| 120777              | 5    | 15   | 5    | 15   | \$40            |

# Business Scenario

Management requested a report containing the total contributions for all employees in the first quarter.

**orion.employee\_donations**



**PROC SQL**



**Desired Results**

|                                          |
|------------------------------------------|
| <b>Total<br/>Quarter 1<br/>Donations</b> |
| <hr/>                                    |
| <b>1515</b>                              |

# Summary Functions: Summarize a Column

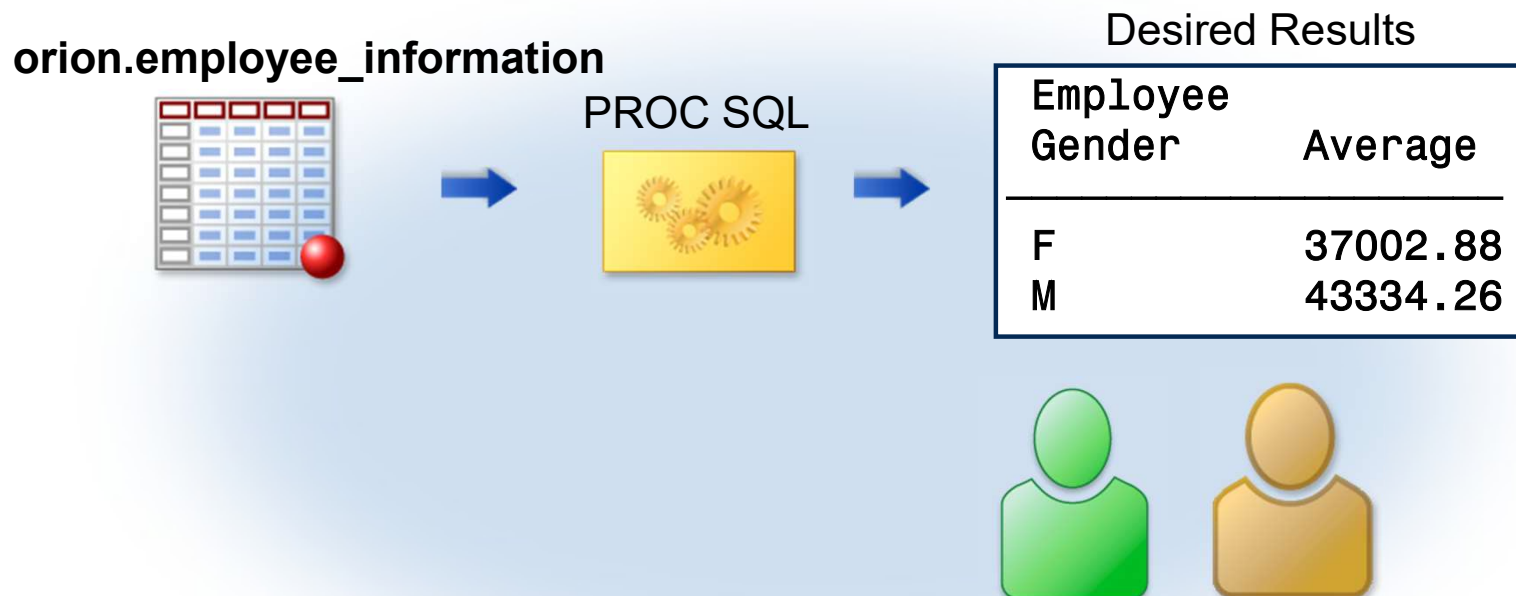
Total employee's annual cash donations.

**SUM(col1)**

```
proc sql;  
select sum(Qtr1)  
       label='Total Quarter 1 Donations'  
       from orion.employee_donations;  
quit;
```

# Business Scenario

Produce a report that determines the average salary by gender.



# Grouping Data

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

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

**GROUP BY** *group-by-item*<,..., *group-by-item*>

# Viewing the Output

PROC SQL Output

| Average Salary by Gender |          |
|--------------------------|----------|
| Employee<br>Gender       | Average  |
| F                        | 37002.88 |
| M                        | 43334.26 |



# Joining Tables

# Objectives

- Identify different ways to combine data horizontally from multiple tables.
- Distinguish between inner and outer SQL joins.
- Understand the Cartesian product.

# Exploring the Data

**customers**

| ID  | Name  |
|-----|-------|
| 101 | Smith |
| 104 | Jones |
| 102 | Blank |

**transactions**

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

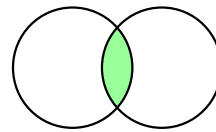
The **customers** table is representative of a customer dimension table. There would be additional columns with data about customers including address, age, and so on.

The **transactions** table is representative of a fact table. There would be columns holding all the key column data, **Product\_ID**, **Employee\_ID**, and so on.

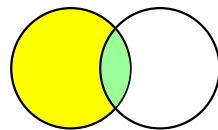
# Types of Joins

PROC SQL supports two types of joins:

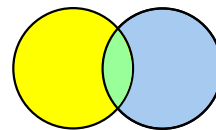
*Inner joins* return only matching rows.



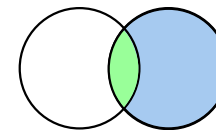
*Outer joins* return all matching rows, plus nonmatching rows from one or both tables.



Left



Full



Right

# 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*.

```
proc sql;  
select *  
  from customers, 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.

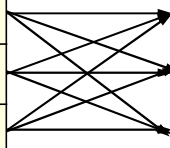
# Building the Cartesian Product

**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  |

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

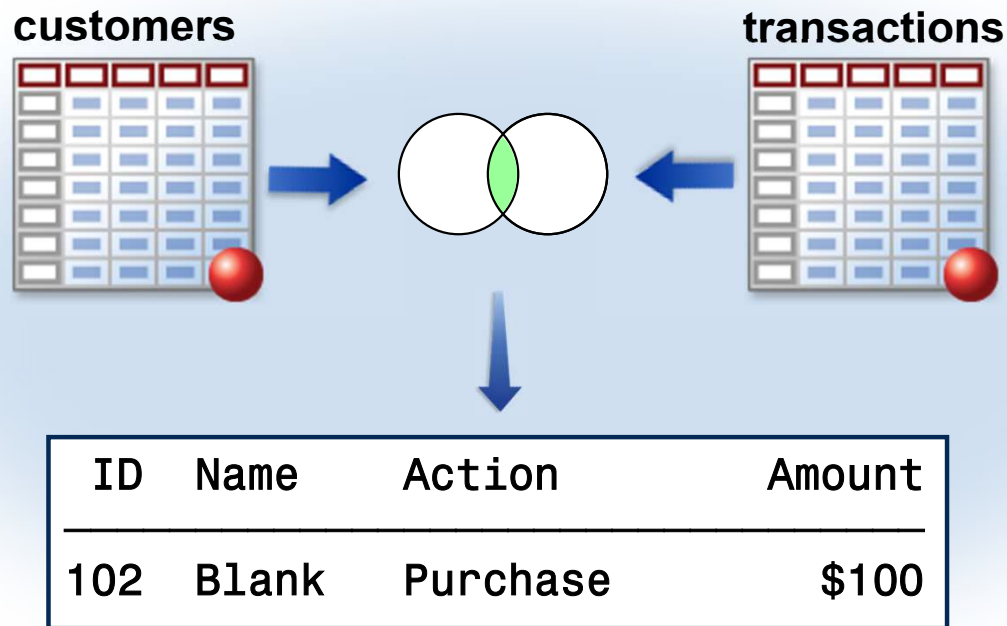


# Objectives

- Join two or more tables on matching columns.
- Qualify column names to identify specific columns.
- Use a table alias to simplify the SQL code.

# Report 1: Inner Join

Management has requested a report showing all valid order information.



# Inner Join

Specify the matching criteria in the WHERE clause.

```
proc sql;  
select *  
  from customers, transactions  
 where customers.ID=  
        transactions.ID;  
quit;
```

```
SELECT object-item<, ...object-item>  
FROM table-name, ... table-name  
WHERE join condition  
      <AND sql-expression>  
      <other clauses>;
```

PROC SQL Output

| ID  | Name  | ID  | Action   | Amount |
|-----|-------|-----|----------|--------|
| 102 | Blank | 102 | Purchase | \$100  |

# Completed Code for Report 1

To display the ID column only once in the results, qualify the ID column in the SELECT clause.

**customers**

| ID  | Name  |
|-----|-------|
| 101 | Smith |
| 104 | Jones |
| 102 | Blank |

**transactions**

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

```
select customers.ID, Name, Action, Amount  
from customers, transactions  
where customers.ID=transactions.ID;
```

| ID  | Name  | Action   | Amount |
|-----|-------|----------|--------|
| 102 | Blank | Purchase | \$100  |

# Abbreviating the Code with a Table Alias

```
proc sql;  
select c.ID, Name, Action, Amount  
       from customers as c, transactions as t  
       where c.ID=t.ID;  
quit;
```

## PROC SQL Output

| ID  | Name  | Action   | Amount |
|-----|-------|----------|--------|
| 102 | Blank | Purchase | \$100  |

# Compare SQL Join and DATA Step Merge

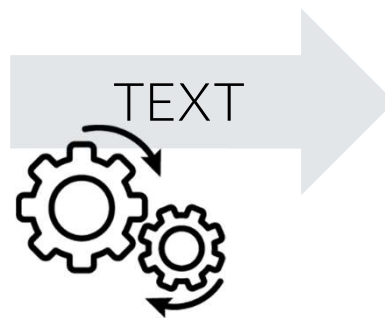
| Key Points                                  | SQL Join     | DATA Step Merge |
|---------------------------------------------|--------------|-----------------|
| Explicit sorting of data before join/merge  | Not required | Required        |
| Same-name columns in join/merge expressions | Not required | Required        |
| Equality in join or merge expressions       | Not required | Required        |



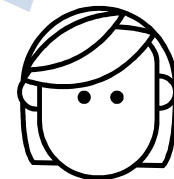
# Introduction to the Macro Facility

# Macro Programming

SAS Macro  
Facility



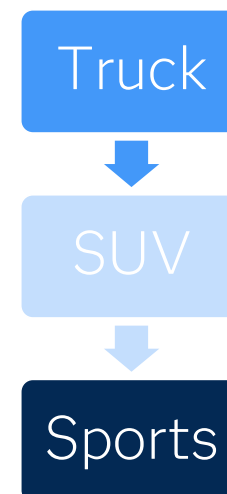
The SAS macro  
facility enables you  
to write code that  
rewrites itself!



# Substituting User-Defined Values

```
title "Trucks by Origin";  
proc freq data=sashelp.cars;  
  where Type="Truck";  
  table Origin;  
run;  
  
title "Average Highway MPG for Trucks";  
proc means data=sashelp.cars mean maxdec=1;  
  where Type="Truck";  
  var MPG_Highway;  
  class Origin;  
run;
```

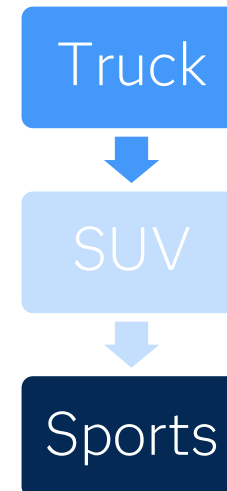
Easily replace repetitive values.



# Substituting User-Defined Values

Create a Macro Variable to replace repetitive values.

```
%let Type=Truck;  
title "&Type by Origin";  
proc freq data=sashelp.cars;  
    where Type="&Type";  
    table Origin;  
run;  
  
title "Average Highway MPG for &Type";  
proc means data=sashelp.cars mean maxdec=1;  
    where Type="&Type";  
    var MPG_Highway;  
    class Origin;  
run;
```

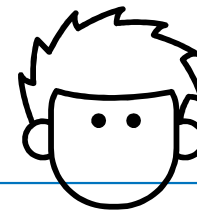


# Substituting System Values

Automatically substitute system values into a program.

```
title "Cars List";  
footnote "Created at 10:24 AM on 14SEP2025";  
title "Trucks by Origin";  
proc freq data=sashelp.cars;  
    where Type="Truck";  
    table Origin;  
run;  
  
title "Average Highway MPG for Trucks";  
proc means data=sashelp.cars mean maxdec=1;  
    where Type="Truck";  
    var MPG_Highway;  
    class Origin;  
run;
```

How can the macro language make your job easier as a SAS programmer?

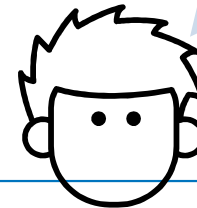


# Substituting System Values

Automatically substitute system values into a program.

```
title "Cars List";  
footnote "Created at &sytime on &sysdate9";  
title "Trucks by Origin";  
proc freq data=sashelp.cars;  
    where Type="Truck";  
    table Origin;  
run;  
  
title "Average Highway MPG for Trucks";  
proc means data=sashelp.cars mean maxdec=1;  
    where Type="Truck";  
    var MPG_Highway;  
    class Origin;  
run;
```

Using the macro facility will make your programs more *dynamic* and *maintenance-free*!



# Efficiency of Macro-Based Applications



The macro facility processes the text in a program to automate and customize the code.

The macro language won't make your code run faster, but it can reduce your development and maintenance time.



# Handy Links

- [SAS 9.4 PROC SQL user's guide](#)
- [Video - Step-by-step PROC SQL](#)
- [Go home on time with 5 PROC SQL tips](#)
- [Ask The Expert Webinar – Top 5 Handy PROC SQL Tips](#)
- [Know thy data: Dictionary tables SAS Global Forum Paper](#)
- [SAS YouTube Video - Mastering the WHERE clause in PROC SQL](#)
- [SAS YouTube Video - Power of SAS SQL –SAS Global Forum 2021](#)
- [SAS YouTube Video - Step by step PROC SQL – SAS Global forum 2020](#)
- [“Ask the Expert Webinar - Why choose between SAS data Step & PROC SQL When You Can Have Both](#)

# Recommended Courses From This Presentation

- SAS® SQL 1: Essentials
- SAS® Macro Language: Essentials

# Thank You

Rebecca Callaway  
SAS Institute San Diego

EMAIL      [Rebecca.Callaway@sas.com](mailto:Rebecca.Callaway@sas.com)

LINKEDIN   <https://www.linkedin.com/in/rebeccazcallaway/>

✓ Did you  
enjoy this  
session, Let us  
know in the  
evaluation



# 2 for 1 Advanced Macro & SQL

University of Iowa SAS User Group  
October 3, 2025

Rebecca Z Callaway  
SAS Institute Inc



# Instructor

Rebecca Callaway, SAS® Institute

With a background in Mathematics and Statistics, SAS Instructor Rebecca Callaway engages with logic, visuals, and analogies to spark critical thinking since 2000.

Rebecca teaches classes on SAS programming, SQL, SAS Visual Analytics, SAS Viya, etc. to support users in the adoption of SAS software.

When not working, Rebecca enjoys spending time outdoors enjoying the lovely San Diego weather and hanging out with her husband Ken and their cat Zigmo.



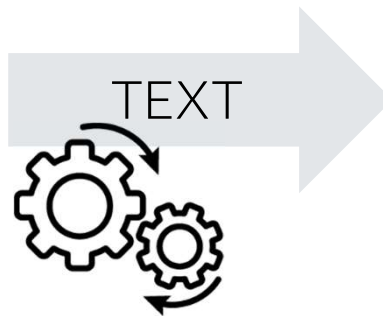
# Agenda

- ? Why Macro
- ✂ Create Macro Variables For Text Substitution
- 🔍 Using Macro Variables For Text Substitution
- 🗄 Create Macro Variables with PROC SQL
- 🔗 Handy Links

# Why SAS Macro?

# Macro Programming

SAS Macro  
Facility



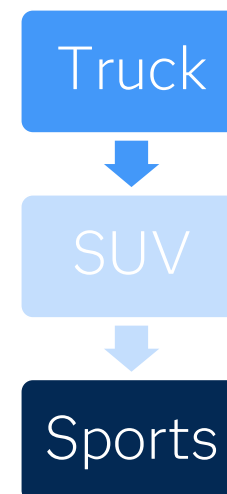
The SAS macro  
facility enables you  
to write code that  
rewrites itself!



# Substituting User-Defined Values

```
title "Trucks by Origin";  
proc freq data=sashelp.cars;  
  where Type="Truck";  
  table Origin;  
run;  
  
title "Average Highway MPG for Trucks";  
proc means data=sashelp.cars mean maxdec=1;  
  where Type="Truck";  
  var MPG_Highway;  
  class Origin;  
run;
```

Easily replace repetitive values.

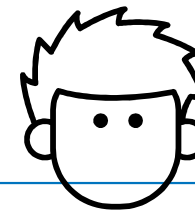


# Substituting System Values

Automatically substitute system values into a program.

```
title "Cars List";  
footnote "Created at 10:24 AM on October 3, 2025";  
title "Trucks by Origin";  
proc freq data=sashelp.cars;  
    where Type="Truck";  
    table Origin;  
run;  
  
title "Average Highway MPG for Trucks";  
proc means data=sashelp.cars mean maxdec=1;  
    where Type="Truck";  
    var MPG_Highway;  
    class Origin;  
run;
```

How can the macro language make your job easier as a SAS programmer?



# Efficiency of Macro-Based Applications



The macro facility processes the text in a program to automate and customize the code.

The macro language won't make your code run faster, but it can reduce your development and maintenance time.



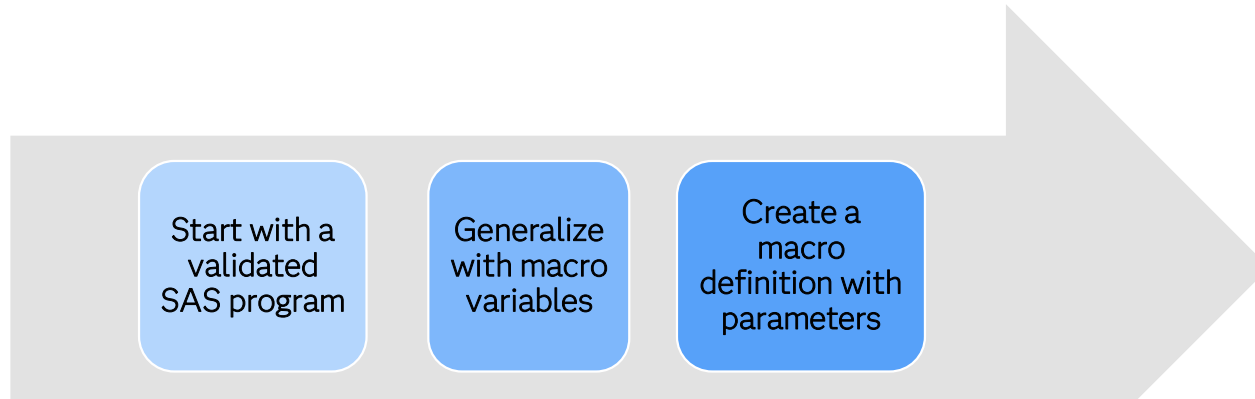
# Creating Macro Variables

# SAS Programming Languages

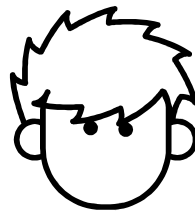
|                    |                                 |
|--------------------|---------------------------------|
| DATA Step          | data manipulation               |
| PROC SQL Step      | data manipulation and reporting |
| SAS Procedures     | data analysis and reporting     |
| SAS Macro Language | generate SAS program code       |



# Developing a Macro Variable



We'll use this process to start with regular SAS code and produce a macro variable.



# Macro Variables

```
title "Trucks with Horsepower > 250";  
proc print data=sashelp.cars;  
  var Make Model MSRP Horsepower;  
  where 'type'="Sedan" and Horsepower>200;  
run;
```

**Truck**

**Sedan**

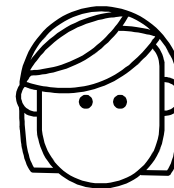
**SUV**

**250**

**150**

**200**

Macro variables  
store text that  
can be used  
anywhere in our  
SAS programs.



# Macro Variables



Macro variables  
each have a name  
and value that are  
stored in a  
memory-based  
symbol table.



**Global Symbol Table**

| Name | Value |
|------|-------|
|      |       |
|      |       |

# Creating Macro Variables with %LET

```
%LET name=value;
```

Macro variable names:

- follow SAS naming rules
- are stored as uppercase
- are not case sensitive

```
%let type=Truck;  
%let hp=250;
```



**Global Symbol Table**

| Name | Value |
|------|-------|
| TYPE | Truck |
| HP   | 250   |

# Creating Macro Variables with %LET

```
%LET name=value;
```

- Case is preserved.
- Leading and trailing blanks are removed.
- It stores 0 to 65,534 (64K) characters.
- The length is dynamically set each time a value is assigned.

```
%let type=Truck;  
%let hp=250;
```



**Global Symbol Table**

| Name | Value |
|------|-------|
| TYPE | Truck |
| HP   | 250   |

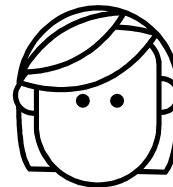
# Creating Macro Variables with %LET

```
%let type=Truck;  
%let hp=250;  
%let type= Sports ;  
%let origin=" Europe " ;  
%let value=;  
%let sum=3+4;  
%let varlist=Make Model Type;
```

Global Symbol Table

| Name | Value |
|------|-------|
| TYPE | Truck |
| HP   | 250   |

Macro variables don't have a type of character or numeric. All values are stored as text.



# Creating Macro Variables with %LET

```
%let type=Truck;  
%let hp=250;  
%let type= Sports ;  
%let origin=" Europe ";  
%let value=;  
%let sum=3+4;  
%let varlist=Make Model Type;
```

Leading and trailing spaces are removed. The value of an existing macro variable is replaced.

Global Symbol Table

| Name | Value  |
|------|--------|
| TYPE | Sports |
| HP   | 250    |

# Creating Macro Variables with %LET

```
%let type=Truck;  
%let hp=250;  
%let type= Sports ;  
%let origin=" Europe " ;  
%let value=;  
%let sum=3+4;  
%let varlist=Make Model Type;
```

Quotation marks are stored  
as part of the value.

Global Symbol Table

| Name   | Value      |
|--------|------------|
| TYPE   | Sports     |
| HP     | 250        |
| ORIGIN | " Europe " |

# Creating Macro Variables with %LET

```
%let type=Truck;  
%let hp=250;  
%let type= Sports ;  
%let origin=" Europe " ;  
%let value=;  
%let sum=3+4;  
%let varlist=Make Model Type;
```

A null value is stored.

Global Symbol Table

| Name   | Value      |
|--------|------------|
| TYPE   | Sports     |
| HP     | 250        |
| ORIGIN | " Europe " |
| VALUE  |            |

# Creating Macro Variables with %LET

```
%let type=Truck;  
%let hp=250;  
%let type= Sports ;  
%let origin=" Europe " ;  
%let value=;  
%let sum=3+4;  
%let varlist=Make Model Type;
```

Mathematical expressions  
are not evaluated.

Global Symbol Table

| Name   | Value      |
|--------|------------|
| TYPE   | Sports     |
| HP     | 250        |
| ORIGIN | " Europe " |
| VALUE  |            |
| SUM    | 3+4        |

# Creating Macro Variables with %LET

```
%let type=Truck;  
%let hp=250;  
%let type= Sports ;  
%let origin=" Europe " ;  
%let value=;  
%let sum=3+4;  
%let varlist=Make Model Type;
```

The variable list is stored  
as a text string.

Global Symbol Table

| Name    | Value           |
|---------|-----------------|
| TYPE    | Sports          |
| HP      | 250             |
| ORIGIN  | " Europe "      |
| VALUE   |                 |
| SUM     | 3+4             |
| VARLIST | Make Model Type |

# Quiz

What would be stored as the value of Mylib?

```
%let mylib=libname mc1 "s:/workshop";
```

**Global Symbol Table**

| Name  | Value |
|-------|-------|
| MYLIB |       |

# Quiz – Correct Answer

What would be stored as the value of Mylib?

```
%let mylib=libname mc1 "s:/workshop";
```

**Global Symbol Table**

| Name  | Value                     |
|-------|---------------------------|
| MYLIB | libname mc1 "s:/workshop" |

The semicolon is treated as the conclusion of the %LET statement and is not stored in the macro variable value.

# Using Macro Variables

# Resolving Macro Variables

***&name***

substitutes the macro variable value into the program

Global Symbol Table

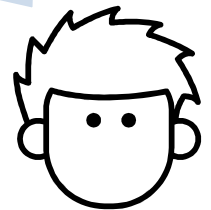
| Name | Value |
|------|-------|
| TYPE | Truck |
| HP   | 250   |

```
%let type=Truck;  
%let hp=250;  
  
proc print data=sashelp.cars;  
  var Make Model MSRP Horsepower;  
  where Type="&type" and Horsepower>&hp;  
run;
```

# Resolving Macro Variables

```
%let type=Truck;  
%let hp=250;  
title1 "Car Type: &type";  
proc print data=sashelp.cars;  
    var Make Model MSRP Horsepower;  
    where Type="&type" and Horsepower>&hp;  
run;
```

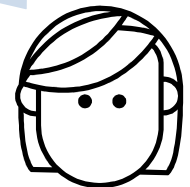
Why is &type  
in quotation  
marks not  
&hp?



# Resolving Macro Variables

```
%let type=Truck;  
%let hp=250;  
title1 "Car Type: &type";  
proc print data=sashelp.cars;  
    var Make Model MSRP Horsepower;  
    where Type="&type" and Horsepower>&hp;  
run;
```

Formulate the  
Where  
statement  
correctly



**where Type="Truck" and Horsepower>250;**

character  
expression

numeric  
expression

# Resolving Macro Variables

```
%let type=Truck;  
%let hp=250;  
title1 "Car Type: &type";  
proc print data=sashelp.cars;  
    var Make Model MSRP Horsepower;  
    where Type="&type" and Horsepower>&hp;  
run;
```

Typically, macro variable values don't include quotation marks.

Use double quotation marks where necessary when resolving macro variables.

# Troubleshooting

## OPTIONS SYMBOLGEN|NOSYMBOLGEN;

```
options symbolgen;
```

```
%let type=Truck;
```

```
%let hp=250;
```

```
title1 "Car Type: &type";
```

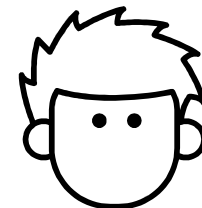
```
proc print data=sashelp.cars;
```

```
  var Make Model MSRP Horsepower;
```

```
  where Type="&type" and Horsepower>&hp;
```

```
run;
```

The SYMBOLGEN option writes information to the log when macro variable references resolve.



```
80 where Type="&type" and Horsepower>&hp;  
SYMBOLGEN: Macro variable TYPE resolves to Truck  
SYMBOLGEN: Macro variable HP resolves to 250
```

# Quotation Marks

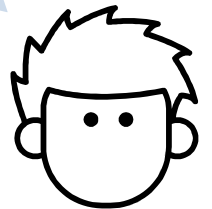
```
title1 "Car Type: &type";  
title2 'Car&Power Report';
```

Macro triggers in double quotation marks are sent to the macro processor.



```
Car Type: Truck  
Car&Power Report
```

Macro triggers in single quotation marks are treated as regular text and are not resolved.



# Delimiting Macro Variable

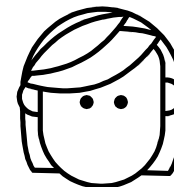
Ref

```
...  
%let type=Truck;  
title "&types with Horsepower > &hp";  
...
```

| Trucks with Horsepower > 250 |           |                |          |            |
|------------------------------|-----------|----------------|----------|------------|
| Obs                          | Make      | Model          | MSRP     | Horsepower |
| 63                           | Cadillac  | Escalade EXT   | \$52,975 | 345        |
| 85                           | Chevrolet | Avalanche 1500 | \$36,100 | 295        |
| 88                           | Chevrolet | Silverado SS   | \$40,340 | 300        |

desired results

What happens if a macro variable reference is concatenated with trailing text?



# Delimiting Macro Variable

Ref

```
...  
%let type=Truck;  
title "&types with Horsepower > &hp";  
...
```

TYPES is not found.

Global Symbol Table

| Name | Value |
|------|-------|
| TYPE | Truck |
| HP   | 250   |

```
74 %let type=Truck;  
75 %let hp=250;  
WARNING: Apparent symbolic reference TYPES not resolved.  
SYMBOLGEN: Macro variable HP resolves to 250  
76 title "&types with Horsepower > &hp";
```

# Delimiting Macro Variable References

```
%let type=Truck;  
title "&type.s with Horsepower > &hp";  
...
```



Use a period to delimit the macro variable name from the text.

```
title "Trucks with Horsepower > 250";
```

The period does not appear in the resolved text.

# Delimiting Macro Variable References

```
footnote "Data Source: &lib..CARS";  
proc print data=&lib..cars;
```

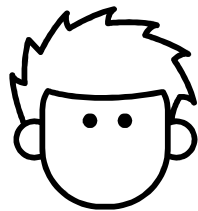
Use two periods between  
the macro variable and  
table name.

The first period is a  
delimiter and is removed  
when &lib resolves. The  
second period remains  
as text.



# Updating Macro Variables

```
%let type=Truck;  
%let hp=250;  
title "&type.s with Horsepower > &hp";  
footnote "Report Created on &sysday, &sysdate";  
proc print data=sashelp.cars;  
    var Make Model MSRP Horsepower;  
    where Type="&type" and  
           Horsepower>&hp;  
run;
```

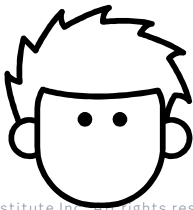


What must be modified  
in the program to  
generate a list of SUVs  
with horsepower greater  
than 300, and then print  
the date in the footnote?

# Updating Macro Variables

```
%let type=SUV;  
%let hp=300;  
title "&type.s with Horsepower > &hp";  
footnote "Report Created on &sysday, &sysdate";  
proc print data=sashelp.cars;  
    var Make Model MSRP Horsepower;  
    where Type="&type" and  
           Horsepower>&hp;  
run;
```

Simply update the  
%LET statements!



| SUVs with Horsepower > 300        |          |                      |          |            |
|-----------------------------------|----------|----------------------|----------|------------|
| Obs                               | Make     | Model                | MSRP     | Horsepower |
| 28                                | BMW      | X5 4.4i              | \$52,195 | 325        |
| 57                                | Cadillac | SRX V8               | \$46,995 | 320        |
| 119                               | Ford     | Excursion 6.8 XLT    | \$41,475 | 310        |
| 144                               | GMC      | Yukon XL 2500 SLT    | \$46,265 | 325        |
| 167                               | Hummer   | H2                   | \$49,995 | 316        |
| 231                               | Lincoln  | Aviator Ultimate     | \$42,915 | 302        |
| 300                               | Nissan   | Pathfinder Armada SE | \$33,840 | 305        |
| 331                               | Porsche  | Cayenne S            | \$56,665 | 340        |
| 378                               | Toyota   | Land Cruiser         | \$54,765 | 325        |
| Report Created on Friday, 01NOV19 |          |                      |          |            |



## Creating and using Macro Variables

This demonstration illustrates creating and using macro variables

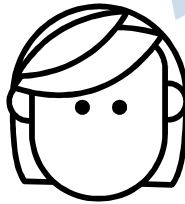
# Creating Macro Variables with PROC SQL

# Creating Macro Variables

%LET  
statement

PROC  
SQL

PROC SQL can  
create and assign  
macro variables  
based on your data.



# SELECT Statement: Syntax Order Mnemonic

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.

# PROC SQL Query (Review)

```
proc sql;  
select Model, MPG_Highway  
  from sashelp.cars  
 where MPG_Highway>50  
 order by MPG_Highway;  
quit;
```

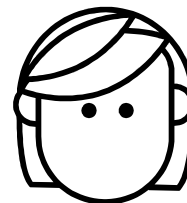
| Model                                  | MPG (Highway) |
|----------------------------------------|---------------|
| Prius 4dr (gas/electric)               | 51            |
| Civic Hybrid 4dr manual (gas/electric) | 51            |
| Insight 2dr (gas/electric)             | 66            |

# Creating Macro Variables with PROC SQL

```
PROC SQL;  
SELECT <DISTINCT> item-1 < , item-2, ...>  
  <INTO : macvar-1 < ..., : macvar-n>  
  FROM clause  
    <WHERE clause>  
    <ORDER BY clause>;  
QUIT;
```

The INTO clause assigns values produced by the query to macro variables.

Be sure to precede each macro variable name with a colon.



# Creating Macro Variables with PROC SQL

## Syntax 1 – Storing Value of First Row in Declared Macro Variables

```
proc sql noprint;  
select make, msrp into :expmake, :maxmsrp  
from sashelp.cars  
order by msrp desc  
;  
%put &=expmake;  
%put &=maxmsrp;
```

Store the first row of the query into 2 macro variables & then request the variable values.

Global Symbol Table

| Name    | Value     |
|---------|-----------|
| MAKE    | Porsche   |
| MAXMSRP | \$192,465 |

MAKE & MAXMSRP are created and stores the make & MSRP of car with highest MSRP

| Make          | MSRP      |
|---------------|-----------|
| Porsche       | \$192,465 |
| Mercedes-Benz | \$128,420 |
| Mercedes-Benz | \$126,670 |
| Mercedes-Benz | \$121,770 |
| Mercedes-Benz | \$94,820  |
| Mercedes-Benz | \$90,520  |
| Acura         | \$89,765  |
| Jaguar        | \$86,995  |
| Mercedes-Benz | \$86,970  |
| Audi          | \$84,600  |
| Porsche       | \$84,165  |
| Jaguar        | \$81,995  |
| Dodge         | \$81,795  |
| Porsche       | \$79,165  |
| Mercedes-Benz | \$76,870  |
| Porsche       | \$76,765  |
| Cadillac      | \$76,200  |
| Volkswagen    | \$75,000  |
| Jaguar        | \$74,995  |
| Jaguar        | \$74,995  |
| Mercedes-Benz | \$74,320  |
| BMW           | \$73,195  |
| Land Rover    | \$72,250  |

# Creating Macro Variables with PROC SQL

## Syntax 1 – Storing Values from Multiple Rows in Declared Macro Variables

```
proc sql noprint;  
select distinct Origin  
    into :origin1-:origin3  
    from sashelp.cars;  
quit;
```

suppresses the report

creates a series of macro variables for the three distinct values of Origin

**Global Symbol Table**

| Name    | Value  |
|---------|--------|
| ORIGIN1 | Asia   |
| ORIGIN2 | Europe |
| ORIGIN3 | USA    |

# Creating Macro Variables with PROC SQL

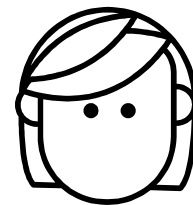
## Syntax 2 - Storing Values from Multiple Rows in a List of Macro Variables

```
proc sql noprint;  
select distinct Type  
    into :type1-  
    from sashelp.cars;  
quit;
```

**Global Symbol Table**

| Name  | Value  |
|-------|--------|
| TYPE1 | Hybrid |
| TYPE2 | SUV    |
| ...   |        |
| TYPE6 | Wagon  |

If you don't know  
how many macro  
variables to create,  
you can omit the  
upper bound.



# Creating Macro Variables with PROC SQL

## Syntax 3 - Storing Values of All Rows in One Macro Variable

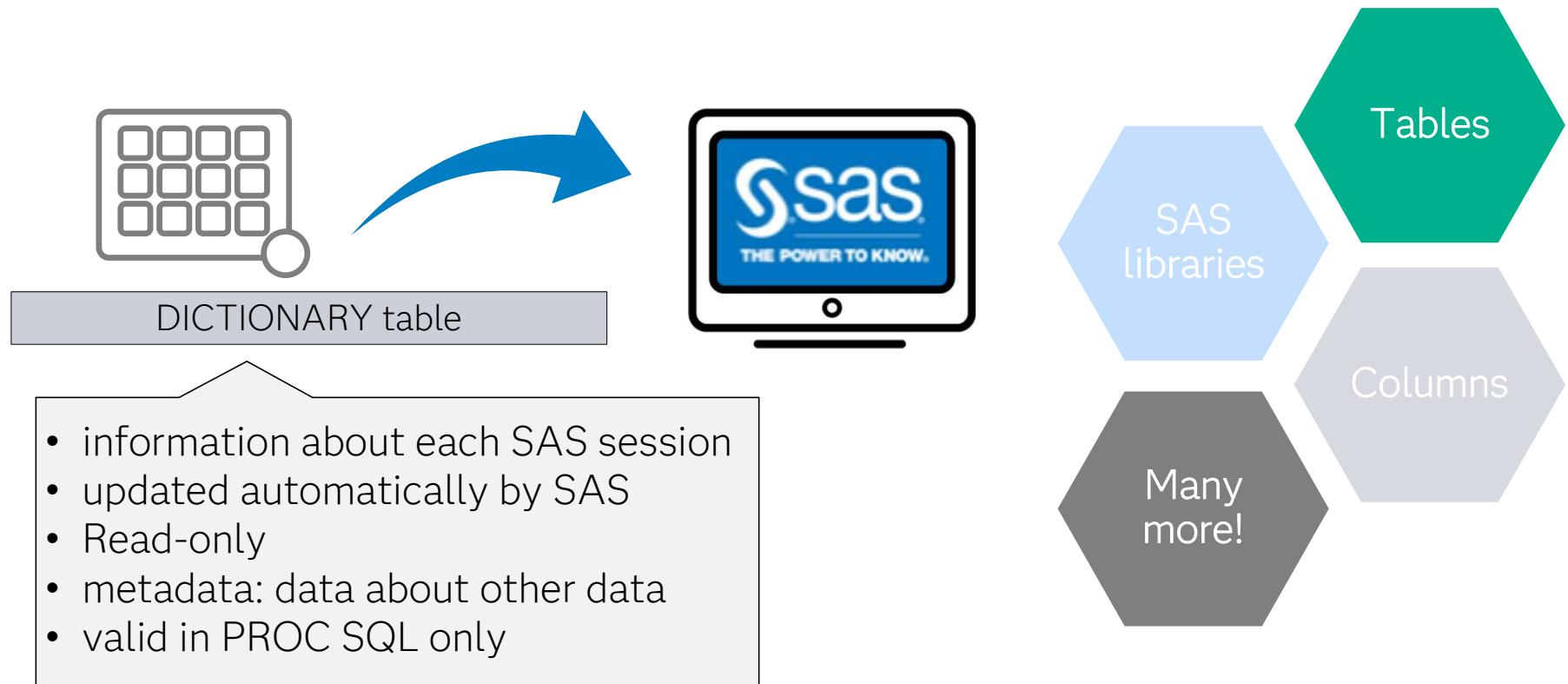
```
proc sql noprint;  
select distinct Origin  
    into :originlist separated by ", "  
    from sashelp.cars;  
quit;
```

Use SEPARATED BY to assign multiple values to a single macro variable.

### Global Symbol Table

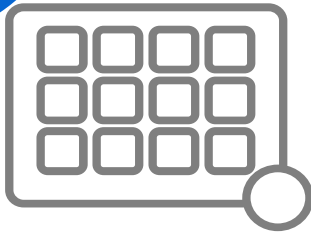
| Name       | Value             |
|------------|-------------------|
| ORIGINLIST | Asia, Europe, USA |
| SQLOBS     | 3                 |

# PROC SQL: DICTIONARY Tables

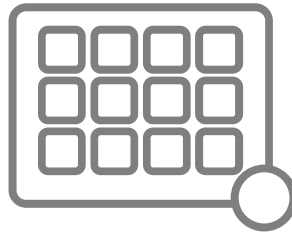


# PROC SQL: DICTIONARY Tables

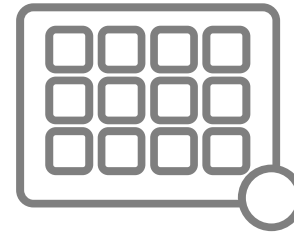
Explore  
Data



DICTIONARY.tables



DICTIONARY.columns



DICTIONARY.libnames

# Want to learn more?

Please visit [learn.sas.com](https://learn.sas.com) to browse our catalog. Details from this presentation are derived from the following courses:

- SAS Macro 1: Essentials
- SAS SQL 1: Essentials

✓ Did you enjoy this session, Let us know in the [evaluation](#)



# Thank You

Rebecca Callaway  
SAS Institute San Diego

EMAIL      [Rebecca.Callaway@sas.com](mailto:Rebecca.Callaway@sas.com)

LINKEDIN   <https://www.linkedin.com/in/rebeccazcallaway/>

✓ Did you  
enjoy this  
session, Let us  
know in the  
[evaluation](#)

