



SAS® SQL 1: Essentials

Course Notes

SAS® SQL 1: Essentials Course Notes was developed by Peter Styliadis, Charlot Bennett, Johnny Johnson, and Mark Jordan. Additional contributions were made by Michele Austin, Brittany Coleman, Bruce Dawless, Davetta Dunlap, Marty Hultgren, John McCall, Rich Papel, Ross Richards, Lorilyn Russell, Allison Saito, Ian Sedgwick, Charu Shankar, Jim Simon, Theresa Stemler, Stacey Syphus, Chris Warters, and Anna Yarbrough. Instructional design, editing, and production support was provided by the Learning Design and Development team.

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.

SAS® SQL 1: Essentials Course Notes

Copyright © 2019 SAS Institute Inc. Cary, NC, USA. All rights reserved. Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

Book code E71409, course code LWSQ1M6/SQ194, prepared date 11Jun2019. LWSQ1M6_001

ISBN 978-1-64295-094-6

Table of Contents

Lesson 1 Essentials	1-1
1.1 Setting Up for the Course	1-3
1.2 What Is SQL?	1-10
1.3 Introduction to the SQL Procedure	1-13
Demonstration: Exploring the customer Table	1-21
1.4 Solutions	1-26
Solutions to Activities and Questions	1-26
Lesson 2 PROC SQL Fundamentals	2-1
2.1 Generating Simple Reports	2-3
Demonstration: Creating Simple Reports	2-17
Demonstration: Assigning Values Conditionally	2-24
Practice	2-27
2.2 Summarizing and Grouping Data	2-31
Demonstration: Summary Functions	2-35
Demonstration: Analyzing Groups of Data	2-41
Demonstration: Summarizing Data Using a Boolean	2-47
Practice	2-50
2.3 Creating and Managing Tables	2-53
2.4 Using DICTIONARY Tables	2-63
Demonstration: Using DICTIONARY Tables	2-66
Practice	2-69
2.5 Solutions	2-71
Solutions to Practices.....	2-71
Solutions to Activities and Questions	2-75

Lesson 3	SQL Joins	3-1
3.1	Introduction to SQL Joins	3-3
3.2	Inner Joins	3-8
	Demonstration: Performing an Inner Join with PROC SQL	3-10
	Demonstration: Performing an Inner Join with Four Tables	3-16
	Practice	3-23
3.3	Outer Joins	3-28
	Demonstration: Performing a Full Join with PROC SQL	3-35
	Practice	3-40
3.4	Complex Joins	3-45
	Demonstration: Performing a Reflexive Join.....	3-47
3.5	Solutions	3-54
	Solutions to Practices.....	3-54
	Solutions to Activities and Questions	3-60
Lesson 4	Subqueries.....	4-1
4.1	Subquery in WHERE and HAVING Clauses	4-3
	Demonstration: Subquery That Returns a Single Value	4-7
	Demonstration: Subquery That Returns Multiple Values	4-12
	Practice	4-19
4.2	In-Line Views (Query in the FROM Clause).....	4-25
	Demonstration: Using an In-Line View.....	4-29
	Practice	4-39
4.3	Subquery in the SELECT Clause	4-43
	Demonstration: Remerging Summary Statistics.....	4-46
	Practice	4-51
4.4	Solutions	4-54
	Solutions to Practices.....	4-54
	Solutions to Activities and Questions	4-61

Lesson 5 Set Operators.....	5-1
5.1 Introduction to Set Operators	5-3
5.2 INTERSECT, EXCEPT, and UNION	5-9
Demonstration: Using the UNION Set Operator.....	5-18
Practice	5-24
5.3 OUTER UNION.....	5-28
Demonstration: Using the OUTER UNION Set Operator	5-30
Practice	5-34
5.4 Solutions	5-36
Solutions to Practices.....	5-36
Solutions to Activities and Questions	5-39
Lesson 6 Using and Creating Macro Variables in SQL.....	6-1
6.1 Creating User-Defined Macro Variables	6-3
6.2 Creating Data-Driven Macro Variables with PROC SQL	6-6
Demonstration: Using a PROC SQL Data-Driven Macro Variable.....	6-10
Demonstration: Concatenating Values in Macro Variables	6-17
Practice	6-22
6.3 Solutions	6-28
Solutions to Practices.....	6-28
Solutions to Activities and Questions	6-33
Lesson 7 Accessing DBMS Data with SAS/ACCESS®	7-1
7.1 Overview of SAS/ACCESS Technology	7-3
7.2 SQL Pass-Through Facility.....	7-6
Demonstration: Using an SQL Pass-Through Query	7-13
7.3 SAS/ACCESS LIBNAME Statement.....	7-18
Demonstration: Using the SAS/ACCESS LIBNAME Statement.....	7-20

7.4 FEDSQL Procedure	7-27
7.5 Solutions	7-39
Solutions to Activities and Questions	7-39

To learn more...



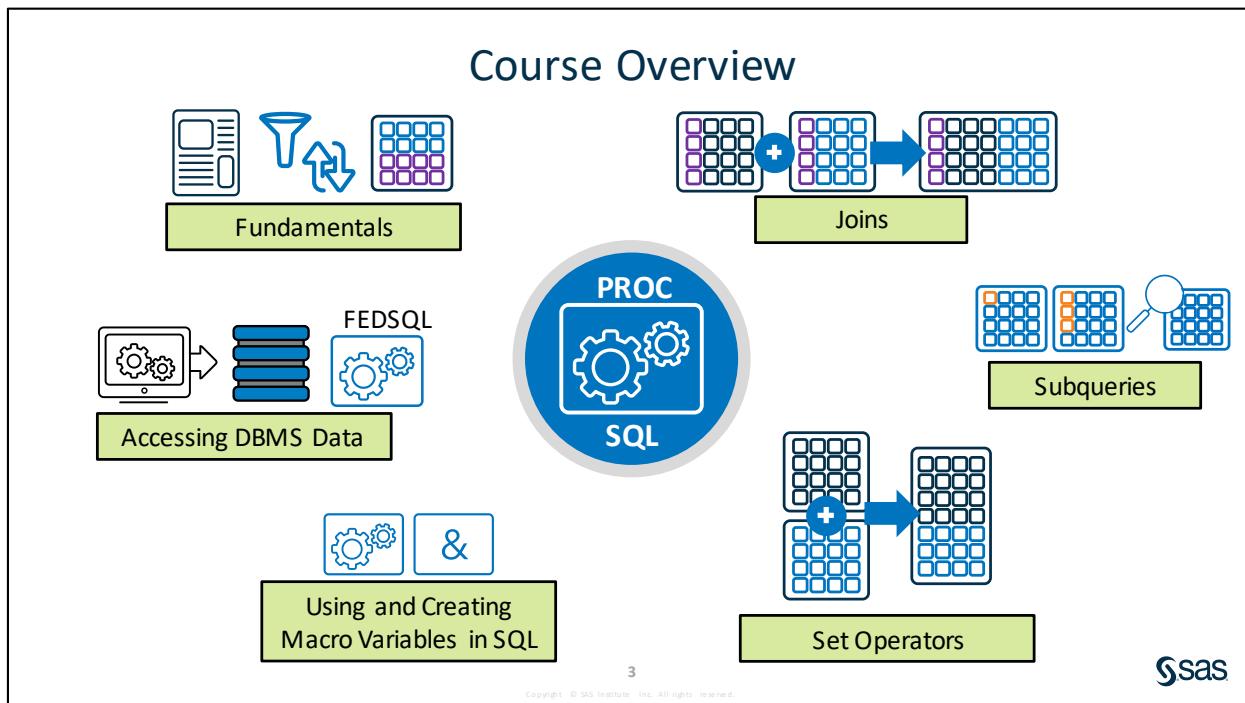
For information about other courses in the curriculum, contact the SAS Education Division at 1-800-333-7660, or send e-mail to training@sas.com. You can also find this information on the web at <http://support.sas.com/training/> as well as in the Training Course Catalog.

For a list of SAS books (including e-books) that relate to the topics covered in this course notes, visit <https://www.sas.com/sas/books.html> or call 1-800-727-0025. US customers receive free shipping to US addresses.

Lesson 1 Essentials

1.1	Setting Up for the Course	1-3
1.2	What Is SQL?.....	1-10
1.3	Introduction to the SQL Procedure	1-13
	Demonstration: Exploring the customer Table	1-21
1.4	Solutions	1-26
	Solutions to Activities and Questions.....	1-26

1.1 Setting Up for the Course



Course Overview

Demonstration	Performed by your instructor as an example for you to observe
Activity	Short practice opportunities for you to perform in SAS, either independently or with the guidance of your instructor
Practice	Extended practice opportunities for you to work on independently
Case Study	A comprehensive practice opportunity at the end of the class

4 Copyright © SAS Institute Inc. All rights reserved.

Sas

Choosing a Practice Level

Level 1	Solve basic problems with step-by-step guidance
Level 2	Solve intermediate problems with defined goals
Challenge	Solve complex problems independently with SAS Help and documentation resources

Choose one practice to do in class based on your interest and skill level.



5

Copyright © SAS Institute Inc. All rights reserved.

When you come to a practice, you can choose an appropriate level: basic, intermediate, or complex.

SAS Programming Interfaces

SAS Studio

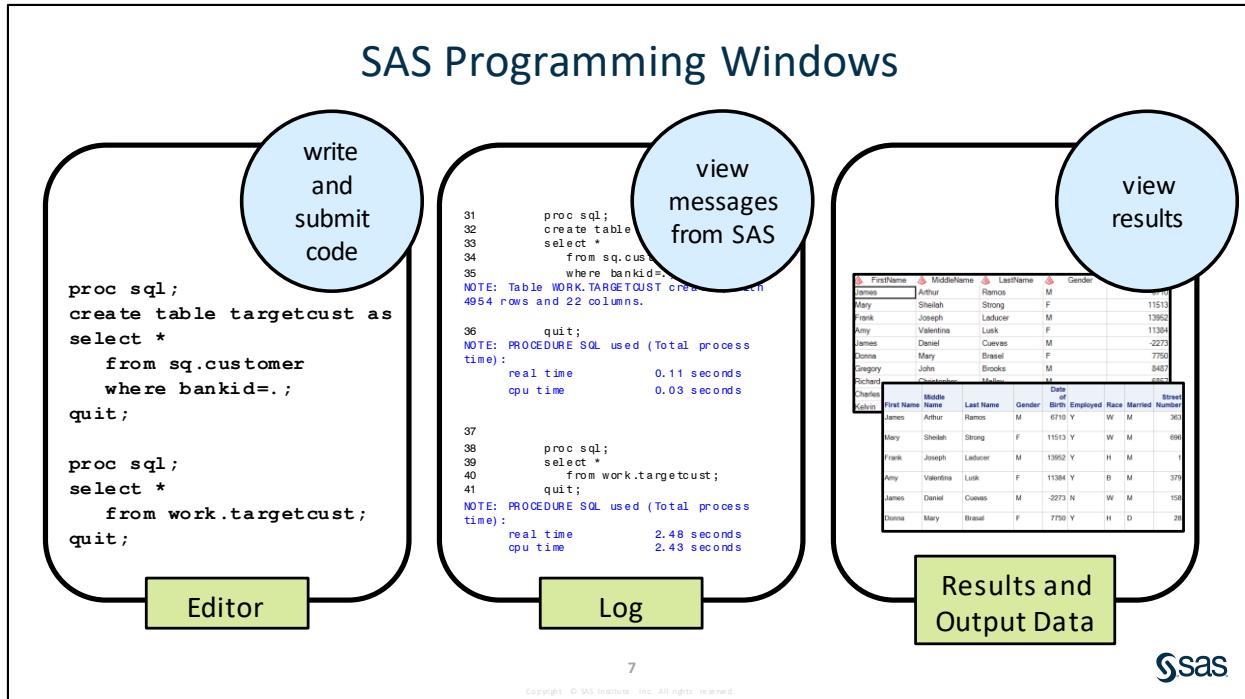
SAS Enterprise Guide

SAS Windowing Environment

6

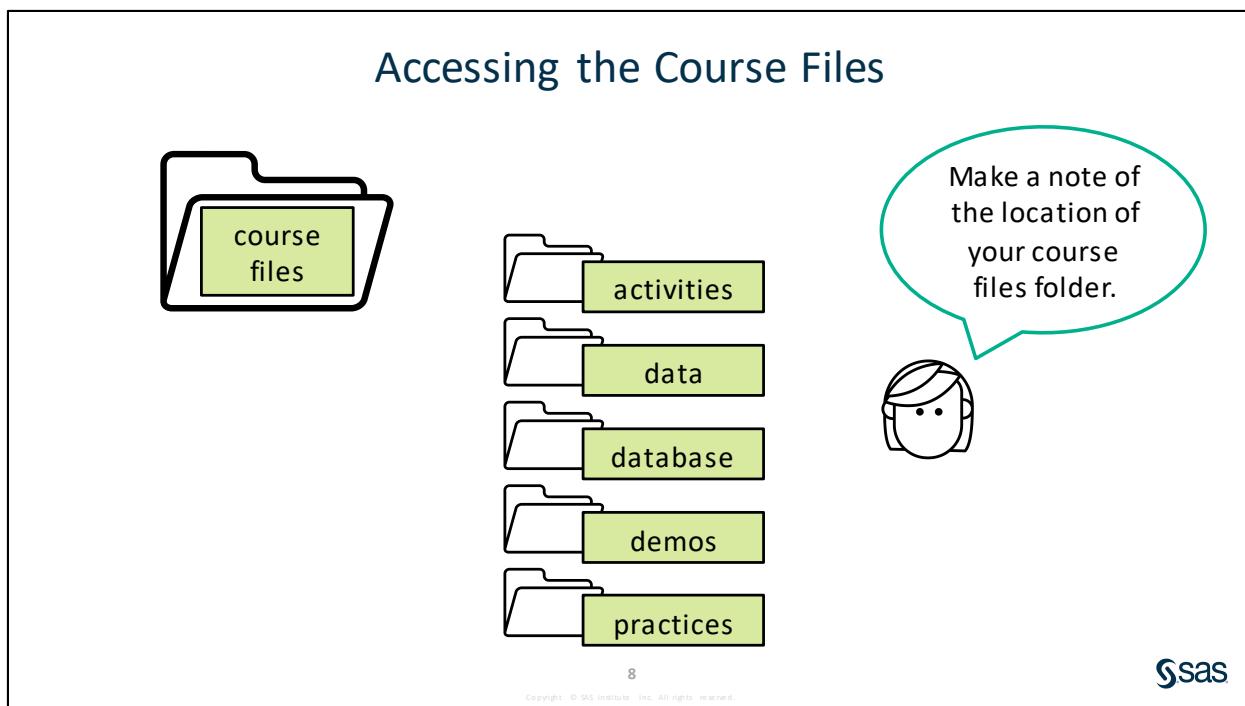
Copyright © SAS Institute Inc. All rights reserved.





7

Copyright © SAS Institute Inc. All rights reserved.



8

Copyright © SAS Institute Inc. All rights reserved.



For this course, you use a variety of data files and SAS programs. The SAS program files are organized into folders for activities, demos, and practices.

Accessing the Course Files

Programs in the activities, demos, and practices folders follow this naming convention.

s104d01.sas
SQL, Lesson 4, demo 1

9

Copyright © SAS Institute Inc. All rights reserved.

Sas

Creating the Course Data

cre8data.sas

10

Copyright © SAS Institute Inc. All rights reserved.

Sas

Open and run the **cre8data.sas** program in the data folder to create the tables for the course. If your files will not be in **s:/workshop**, change the value of PATH= in the %LET statement to reflect your actual course folder location.

1.01 Activity (Required)

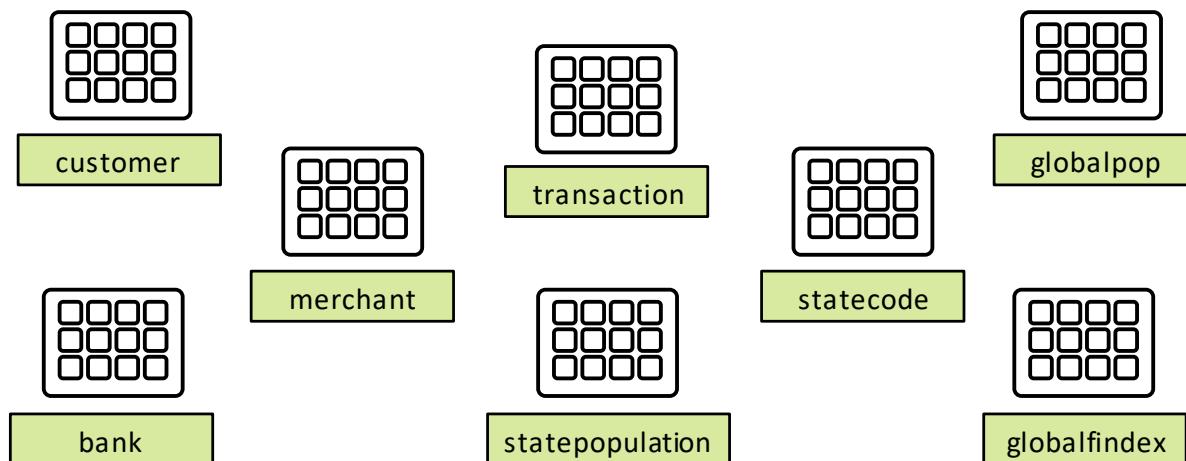
1. Navigate to the location of the course files.
SAS Studio: In the Navigation pane, expand **Files and Folders**.
SAS Enterprise Guide: In the Servers list, expand **Servers** ⇒ **Local** ⇒ **Files**.
2. Double-click the **cre8data.sas** file to open the program.
3. Find the %LET statement. As directed by your instructor, provide the path to your course files.
4. Run the program and verify that a report is created listing **27** generated tables.

11



Copyright © SAS Institute Inc. All rights reserved.

Sample of Data Used in This Course



13



Copyright © SAS Institute Inc. All rights reserved.

United States population data was obtained from the US Census Bureau:

- United States Census Bureau: US Census State Population Totals and Components of Change:
<https://www.census.gov/newsroom/press-kits/2018/pop-estimates-national-state.html>

Global data was obtained from the World Bank:

- The World Bank: Global Financial Inclusion (Global Findex) Database:
<https://datacatalog.worldbank.org/dataset/global-financial-inclusion-global-findex-database>

- The World Bank: Population estimates and projections:
<https://databank.worldbank.org/data/source/population-estimates-and-projections/Type/TABLE/preview/on>

Connecting to a Library to Read SAS Files (Review)

The diagram illustrates the structure of the LIBNAME statement:

```
LIBNAME libref engine "path";
```

The statement consists of three main parts:

- name of library**: An eight-character maximum, starting with a letter or underscore, followed by letters, numbers, or underscores.
- type of data**: Engines such as Oracle, Teradata, and Postgres enable you to read and write to other types of data.
- location of data**: The path where the data is located.

A blue circle labeled "access data" is positioned to the left of the statement. A small note at the bottom right indicates "Copyright © SAS Institute Inc. All rights reserved." and shows the SAS logo.

For more information about the LIBNAME statement, see "LIBNAME Statement" in the SAS® 9.4 and SAS® Viya® 3.4 Programming documentation. You can also find the direct link in the Course Links section on the Extended Learning page.

Connecting to a Library to Read SAS Files (Review)

The diagram illustrates the structure of the LIBNAME statement:

```
libname sq base "s:/workshop/data";
```

The statement consists of three main parts:

- library name**: The identifier for the library, which is "sq".
- SAS tables (default engine)**: The type of data, which is "base".
- data located in s:/workshop/data**: The location of the data.

A blue circle labeled "connect to the library" is positioned to the left of the statement. A small icon of a computer monitor with four windows is shown below the library name. A callout bubble from a cartoon character's head contains the text: "LIBNAME is a global statement and does not require a RUN statement." The SAS logo is in the bottom right corner.

When reading SAS tables, the BASE engine is optional.

1.02 Activity

1. Open **libname.sas** from the main folder. Complete the LIBNAME statement to create a library named **sq** that reads SAS tables in the **data** folder. **The path should be the folder where your course files are located.**

Note: In Enterprise Guide, click **Libraries** ⇒ **Refresh** to update the library list.

```
libname sq "s:/workshop/data";
```

2. Run the code and verify that the library was successfully assigned in the log.
3. Save the updated **libname.sas** program.
4. Navigate to your list of libraries and expand the **sq** library. Verify that the library exists and tables are available.

16

Copyright © SAS Institute Inc. All rights reserved.



Extending Your Learning

Extended Learning - SAS® SQL 1: Essentials

Dashboard / Courses / Extended Learning - SAS® SQL 1: Essentials

Thank you for taking the SAS® SQL 1: Essentials course. You are invited to extend your learning experience by using the resources listed on this page.

▶ Open all ▼ Close all

- ▶ Course Materials
- ▶ Course Cheat Sheets
- ▶ Case Study
- ▶ PROC SQL and PROC FEDSQL Documentation
- ▼ SQL Course Links

Lesson 1: Essentials

- ◀ LIBNAME Statement
- ◀ PROC SQL and the ANSI Standard
- ◀ SQL Procedure Reference
- ◀ CONTENTS Procedure - SAS
- ◀ Dictionary of Data Set Options
- ◀ PROC SQL Options

Beyond L1 SQL Essentials

- ◀ SAS® Programming 1: Essentials

Use your **Extended Learning page (ELP)** to download course files and access additional videos, papers, and other helpful resources.

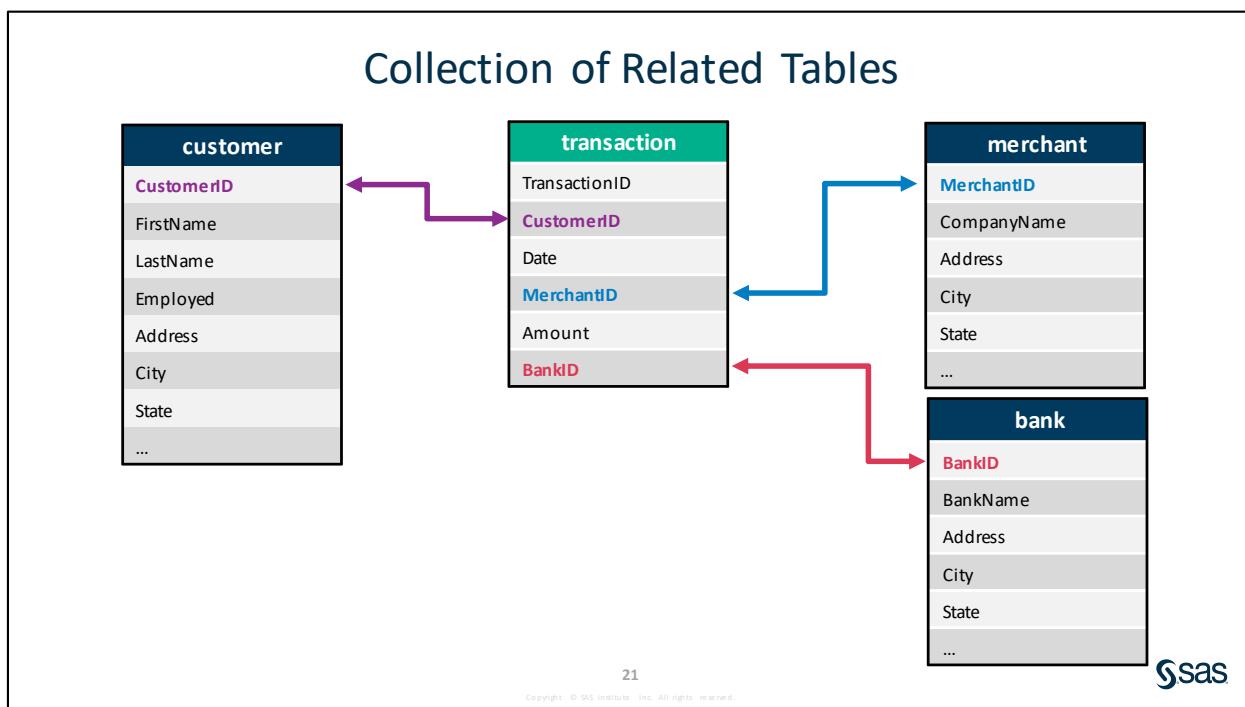
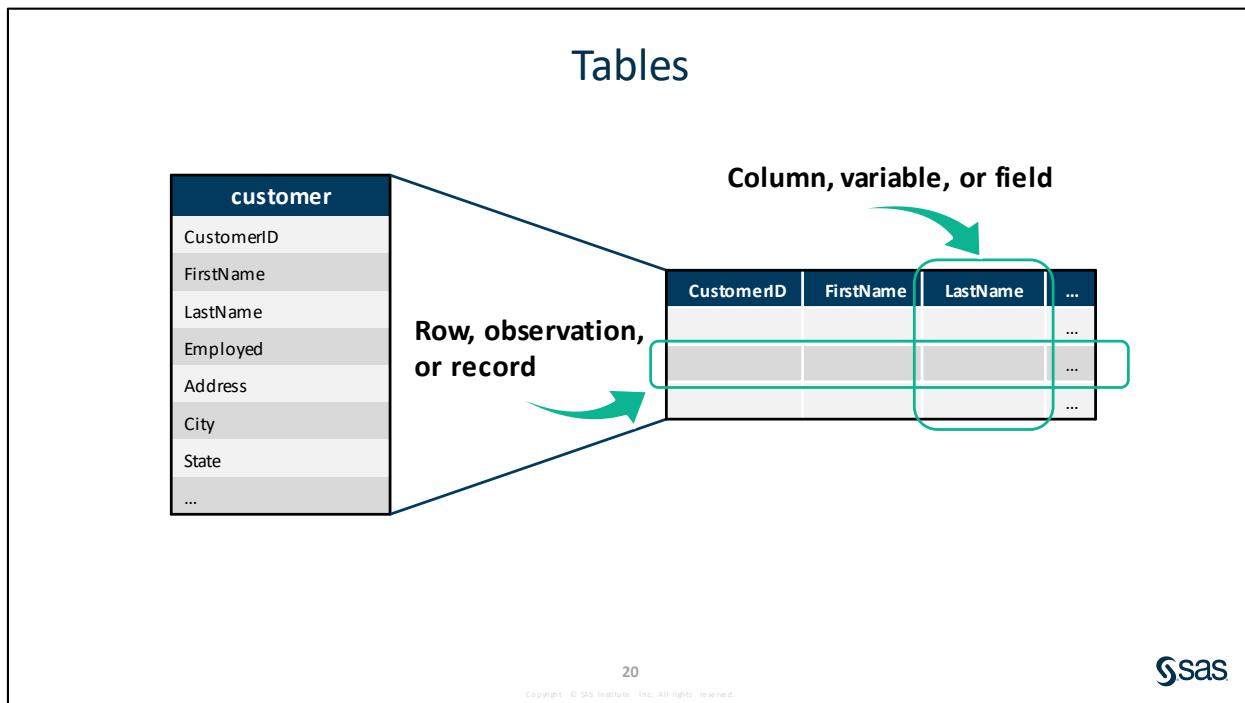


18

Copyright © SAS Institute Inc. All rights reserved.

Throughout each lesson, you will see references to topics in the SAS documentation. All links are provided on the ELP for each lesson.

1.2 What Is SQL?



Structured Query Language (SQL)

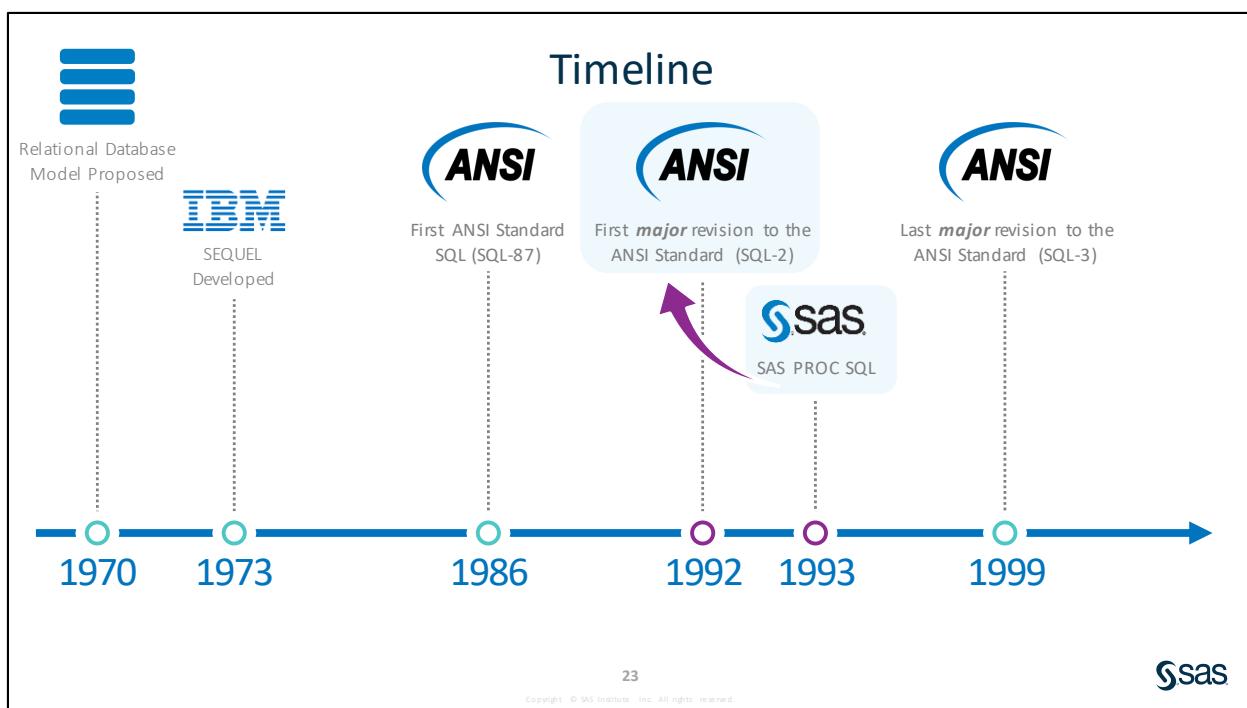


Structured Query Language (SQL) is a standardized, widely used language designed to query and manage data in relational databases.



22

Copyright © SAS Institute Inc. All rights reserved.



1970 - Dr. Edgar F. Codd proposes a relational data model ("A Relational Model of Data for Large Shared Data Banks")

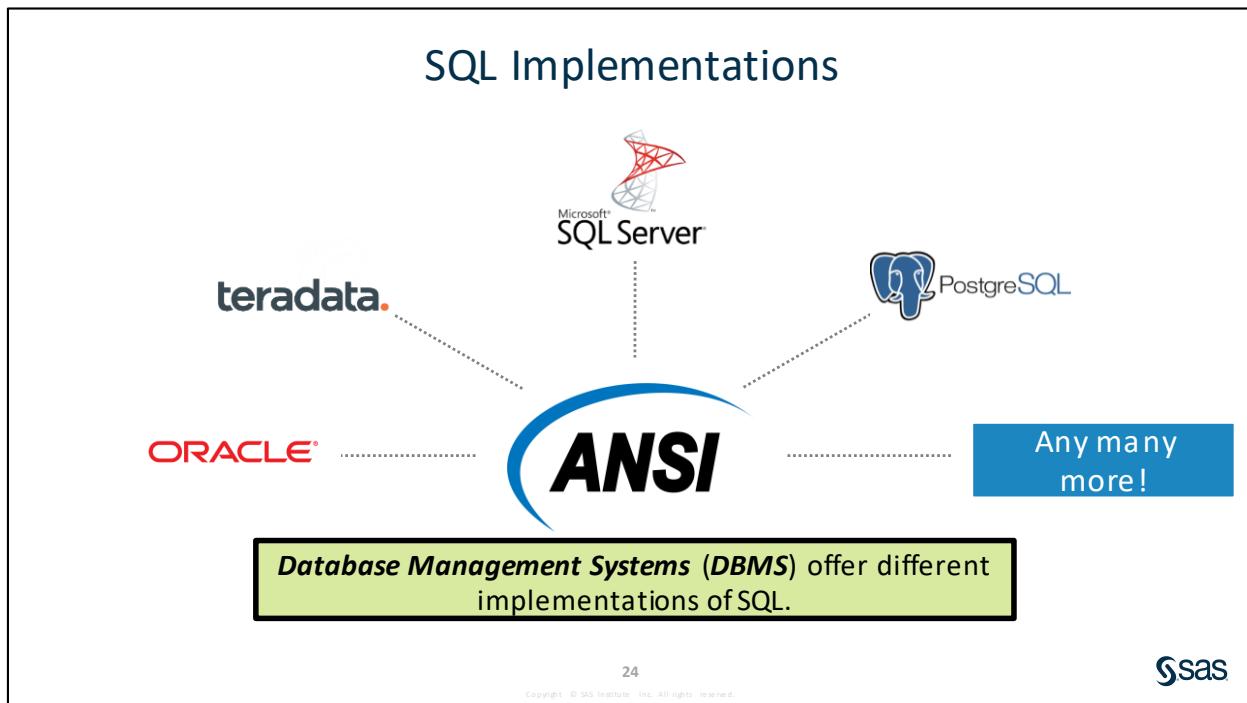
1973 - Donald D. Chamberlin and Raymond F. Boyce develop SEQUEL (Structured English Query Language) at IBM

1986 - First ANSI standard for SQL (SQL-87)

1992 - First major revision to the ANSI standard (SQL-2)

1993 - SAS SQL procedure added (based on SQL-2 standard). ***Although PROC SQL incorporated many of the standards, SAS SQL is not ANSI compliant.***

1999 - Last major revision to the ANSI standard (SQL-3)



1.3 Introduction to the SQL Procedure

SQL Procedure

SQL + SAS

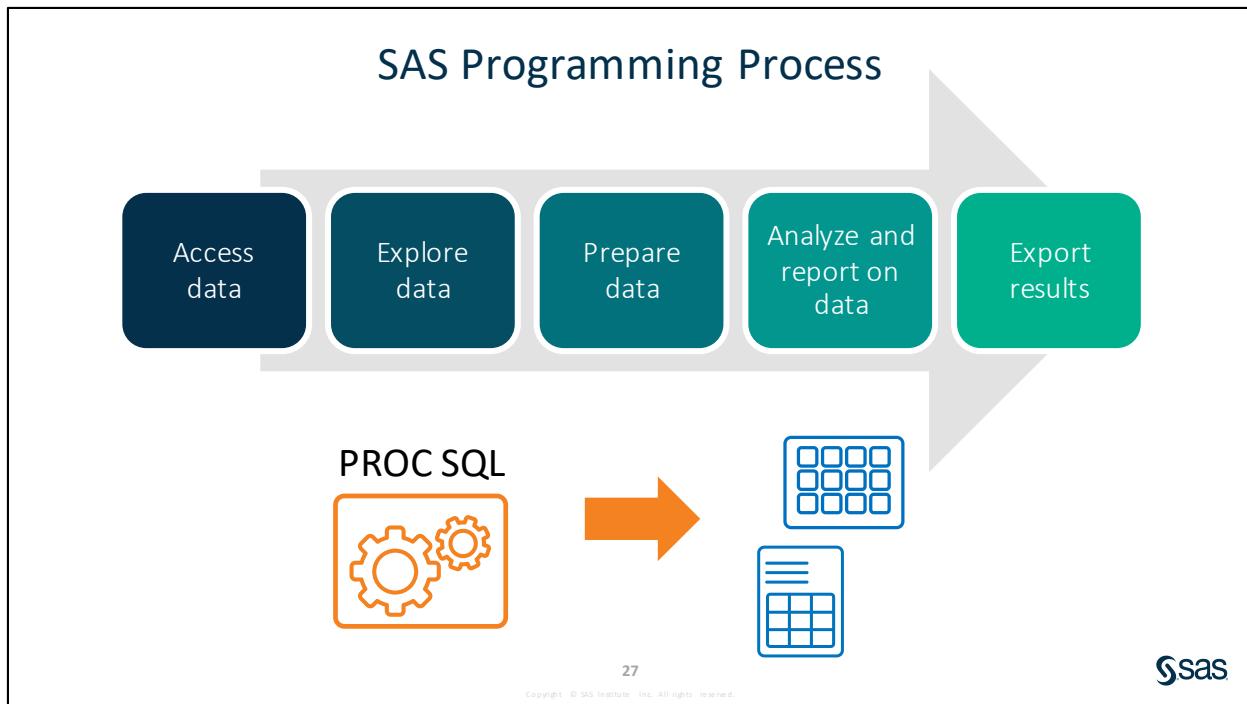
PROC SQL enables the use of SQL in SAS and includes non-ANSI-compliant *SAS enhancements*.

26

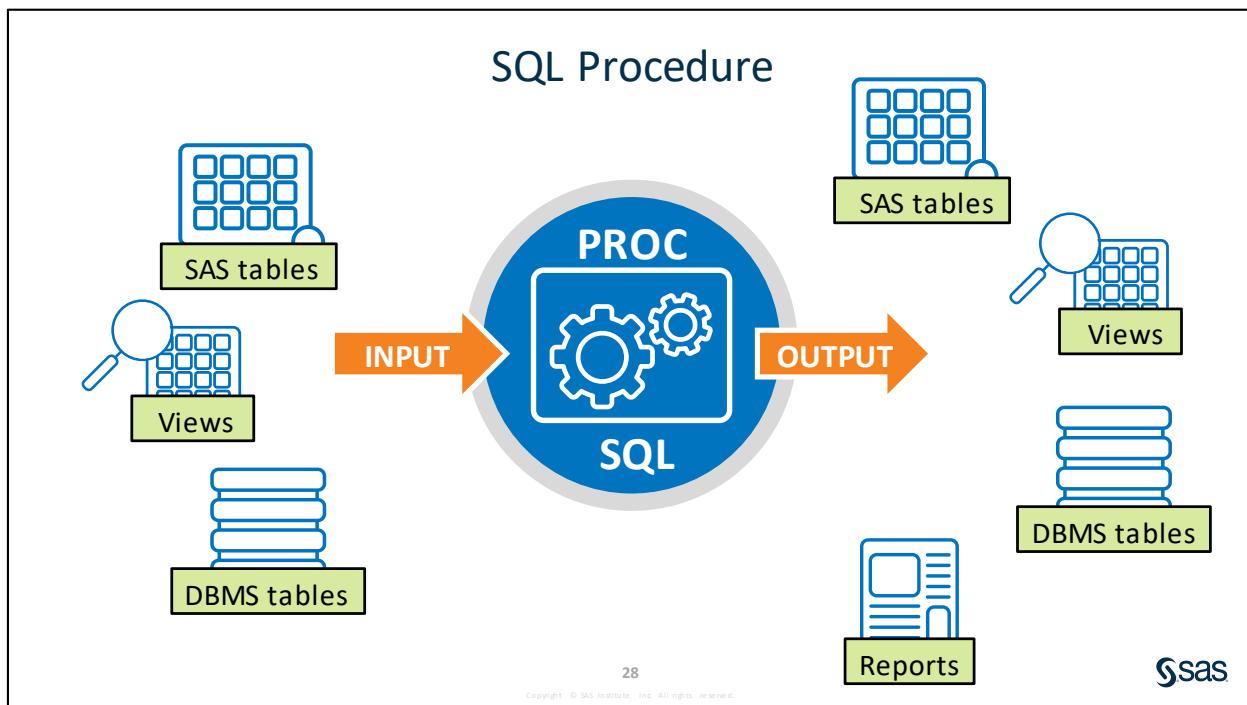
Copyright © SAS Institute Inc. All rights reserved.

PROC SQL follows most of the guidelines set by the American National Standards Institute (ANSI) in its implementation of SQL. However, it is not fully compliant with the current ANSI standard for SQL.

For more information about SAS and the ANSI standard, see "PROC SQL and the ANSI Standard" in the *SAS SQL Procedure User's Guide* documentation. You can also find the direct link in the Course Links section on the ELP.



As you go through the process of making data meaningful and actionable using SQL, you will likely follow these basic steps: access data or read it with a program, explore the data to see what is there and what you might need to add or change, prepare the data to get it ready for analysis, analyze and report on the data, and export results to various report and data formats.



To read unstructured data, use the SAS DATA step.

1.03 Multiple Choice Question

What is your experience with SQL?

- a. maintaining programs
- b. writing programs
- c. none at all

29

Copyright © SAS Institute Inc. All rights reserved.



SQL Procedure

PROC SQL <options>;

The SQL procedure is initiated with a **PROC SQL** statement.

```
SELECT col-name, col-name
      FROM input-table
      <WHERE clause>
      <GROUP BY clause>
      <HAVING clause>
      <ORDER BY clause>;
```

QUIT;

The SQL procedure is terminated with a **QUIT** statement.



30

Copyright © SAS Institute Inc. All rights reserved.



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

SELECT Statement in PROC SQL

```
PROC SQL <options>;
query
  SELECT col-name, col-name
    FROM input-table
    <WHERE clause>
    <GROUP BY clause>
    <HAVING clause>
    <ORDER BY clause>;
QUIT;
```

A **SELECT** statement or query consists of **clauses** and ends with a **semicolon**.

Think of a *query* as asking your data a question.



Sas

31

Copyright © SAS Institute Inc. All rights reserved.

The **SELECT** statement is the most commonly used SQL statement and is usually referred to as a *query*.

SELECT Statement Clauses in PROC SQL

```
PROC SQL <options>;
query
  SELECT col-name, col-name
    FROM input-table
    <WHERE clause>
    <GROUP BY clause>
    <HAVING clause>
    <ORDER BY clause>;
QUIT;
```

The **SELECT** clause lists the columns to appear in the results, in the order written.

The **FROM** clause specifies the data sources.

The SELECT statement must contain **SELECT** and **FROM** clauses.



Sas

32

Copyright © SAS Institute Inc. All rights reserved.

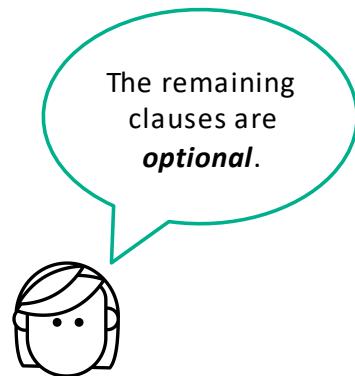
SELECT Statement Clauses in PROC SQL

PROC SQL <options>;

```
SELECT col-name, col-name
  FROM input-table
  <WHERE clause>
  <GROUP BY clause>
  <HAVING clause>
  <ORDER BY clause>;
```

QUIT;

If present, the other clauses
must be in this order.



sas

33

Copyright © SAS Institute Inc. All rights reserved.

When you construct a SELECT statement, you must specify the clauses in the following order:

- The SELECT clause selects columns.
- The FROM clause selects one or more source tables or views.
- The WHERE clause enables you to filter rows of data.
- The GROUP BY clause enables you to process data in groups.
- The HAVING clause works with the GROUP BY clause to filter grouped results.
- The ORDER BY clause specifies the order of rows.

Additional Statements in PROC SQL

`PROC SQL <options>;`

`statement(s);`

`QUIT;`

PROC SQL supports additional statements.
Each statement begins with the **keyword**
and ends with a **semicolon**.

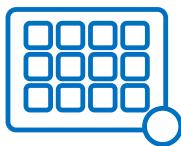
Visit the SAS
documentation for all
available statements.



34

For more information about the available PROC SQL statements, see "SQL Procedure" in the SAS® 9.4 SQL Procedure User's Guide documentation. You can also find the direct link in the Course Links section on the ELP.

Exploring Tables



`customer`

- column attributes
- preview first 10 rows

Explore the
customer table.



35

Viewing Column Attributes

Explore Data

DESCRIBE TABLE *table-name;*

```

proc sql;
  describe table sq.customer;
quit;

```

NOTE: SQL table SQ.CUSTOMER was created like:

```

create table SQ.CUSTOMER( bufsize=65536 )
(
  FirstName char(15) label='First Name',
  MiddleName char(15) label='Middle Name',
  LastName char(15) label='Last Name',
  Gender char(1),
  DOB num label='Date of Birth',
  Employed char(1),
  Race char(1),
  Married char(1),
  StreetNumber num label='Street Number',
  StreetName char(30) label='Street Name'.

```



SAS

The **DESCRIBE TABLE** statement lists all the columns in a table and their properties.

The DESCRIBE TABLE statement returns results comparable to PROC CONTENTS, which shows the contents of a SAS table and more in-depth table information, and prints the directory of the SAS library.

PROC CONTENTS DATA=*data-set*;

RUN;

For more information about the CONTENTS procedure, see “CONTENTS Procedure” in *Base SAS® 9.4 Procedures Guide, Seventh Edition*. You can also find the direct link in the Course Links section on the ELP.

Previewing the Table

SELECT col-name, col-name
FROM input-table <(data set options)>;

```
proc sql;
  select FirstName, LastName, State
    from sq.customer (obs=10);
quit;
```



 proc sql;
 select FirstName, LastName, State
 from sq.customer (obs=10);
quit;

First Name	Last Name	State
Rodney	Joyner	WI
Jeanne	Ballenger	WA
Brian	Harper	WI
Thomas	Henderson	WA
Becky	Cheers	WI
Alberto	Texter	WI
Peter	Schmand	WA
Danielle	Bell	WI
Robert	Brousseau	WI
Sharon	Howell	WA



To select all of a table's columns in the order in which they were stored, specify an asterisk (*) in a SELECT clause instead of column names.

The OBS= data set option specifies the last observation that SAS processes in a data set.

For more information about data set options, see “Dictionary of Data Set Options” in the *SAS® 9.4 and SAS® Viya® 3.4 Programming* documentation. You can also find the direct link in the Course Links section on the ELP.



Exploring the customer Table

Scenario

Use PROC SQL to explore the **customer** table.

Files

- **s101d01.sas**
- **customer** – a SAS table that contains one row per customer.

Syntax

```
PROC SQL;
DESCRIBE TABLE table-name;
QUIT;

PROC SQL;
SELECT col-name, col-name
    FROM input-table(OBS=n);
QUIT;
```

Notes

- The DESCRIBE TABLE displays each column attribute in the log.
- The SELECT statement is the primary tool of PROC SQL. You use it to identify, retrieve, and manipulate columns of data from a table. You can also use several optional clauses within the SELECT statement to place restrictions on a query.
- The SELECT clause lists the columns that will appear in the results. The asterisk (*) selects all columns.
- The FROM clause specifies source tables or views.

Demo

1. Open **s101d01.sas** from the **demos** folder and find the Demo section of the program. In the first SQL procedure step, add a DESCRIBE TABLE statement to see column attributes of the **sq.customer** table. Highlight the step and run the selected code. Examine the log and results.

```
proc sql;
describe table sq.customer;
quit;
```

2. In the second SQL procedure step, add a SELECT statement and select all the columns from the **sq.customer** table using an asterisk. Add the OBS=10 data set option to the table to limit the report to 10 rows.

Note: In this example, the **customer** table contains more than 100,000 rows. Running the query without the OBS= option displays all the rows and might cause system issues.

```
proc sql;
select *
    from sq.customer(obs=10);
quit;
```

3. Modify the SELECT statement to select the columns **FirstName**, **LastName**, and **DOB**. Highlight the step and run the selected code. Examine the log and results.

Note: PROC SQL displays the permanently assigned labels if they exist.

```
proc sql;
select FirstName, LastName, DOB
  from sq.customer(obs=10);
quit;
```

4. Modify the SELECT statement to select the columns **CustomerID**, **LastName**, **UserID**, and **DOB**. Highlight the step and run the selected code. Examine the log and results.

```
proc sql;
select CustomerID, LastName, UserID, DOB
  from sq.customer(obs=10);
quit;
```

End of Demonstration

SQL Options: Controlling Processing

PROC SQL INOBS=n;



PROC SQL OUTOBS=n;

Limits rows from each sourcetable
that contribute to a query

Restricts the number of rows
that a query outputs

```
proc sql inobs=10;
select CustomerID, LastName, UserID, DOB
  from sq.customer;
quit;
```

39


Copyright © SAS Institute Inc. All rights reserved.

SQL Options: Controlling Display

PROC SQL NUMBER;

controls whether the row number is displayed
as the first column in the query results

```
proc sql inobs=10 number;
select CustomerID, LastName, UserID, DOB
  from sq.customer;
quit;
```



Row	Customer ID	Last Name	User ID
1	1902986359	Joyner	rodmajoyner6611@n/a.c
2	1935367360	Ballenger	jeacaballenger638@fake
3	1455003144	Harper	bridaharper4714@invalid
4	1979102386	Henderson	thoerhenderson6322@is
5	1914860679	Cheers	herdacheers4524@n/a.c

40


Copyright © SAS Institute Inc. All rights reserved.

For more information about PROC SQL options, see “PROC SQL Options” in the *SAS® 9.4 SQL Procedure User’s Guide, Fourth Edition* documentation. You can also find the direct link in the Course Links section on the ELP.

1.04 Activity

Open **s101a04.sas** from the **activities** folder and perform the following tasks to explore the **sq.customer** table:

1. Remove the asterisk and select only the **FirstName**, **LastName**, and **State** columns. Run the query. View the log and results.
2. Remove the OBS=10 data set option and add the INOBS=10 PROCSQL option *after* the PROC SQL keywords and *before* the semicolon. Run the query. Are the results the same using the INOBS=10 option? What about the log?
3. After the INOBS= option, add the NUMBER option. Run the query. Which column was added to the results?

Comparing SQL and the DATA Step



DATA Step

- Reads unstructured data
- More control of how SAS processes data
- Can create multiple tables in one step
- Includes looping and array processing



PROC SQL

- Code can be more streamlined
- Describe what you want to do, not how to do it
- SQL optimizer

PROC SQL is a complement to the DATA step, not a replacement for the DATA step. Sometimes PROC SQL is the best tool to use, but in other situations, it is better to use the DATA step.

Syntax Summary

```
PROC SQL <options>;
SELECT col-name, col-name
  FROM input-table;
QUIT;
```

SELECT Statement

```
DESCRIBE TABLE table-name;
```

Explore a Table



input-table (**OBS=n**)

Data Set Options

```
PROC SQL INOBS=n OUTOBS=n NUMBER;
```

PROC SQL Options

46

Copyright © SAS Institute Inc. All rights reserved.



Beyond SQL Essentials

What if you want to ...

... learn about SAS programming essentials?

- Take the free e-learning course [SAS Programming 1: Essentials](#).

... learn about techniques to explore your data?

- Read the SAS paper [Know Thy Data: Techniques for Data Exploration](#).

... learn about reading unstructured data using the DATA Step?

- Read the SAS paper [Reading Delimited Text Files into SAS®9](#).
- Read the SAS paper [From Unfriendly Text File to SAS Dataset – Reading Character Strings](#).
- Visit the SAS blog [Turning text files into SAS data sets](#).

47

Copyright © SAS Institute Inc. All rights reserved.



1.4 Solutions

Solutions to Activities and Questions

1.01 Activity – Correct Answer

The CONTENTS Procedure

Directory			
Libref	SQ	Engine	V9
Physical Name	s:\workshop\data	Filename	s:\workshop\data
Owner Name	BUILTIN\Administrators	File Size	12KB
File Size (bytes)	12288		
#	Name	Member Type	File Size
1	AGEGROUP	DATA	128KB 06/10/2019 12:58:14
2	BANK	DATA	128KB 06/10/2019 12:58:13
3	CUSTOMER	DATA	39MB 06/10/2019 12:58:15
4	DIVISIONCODE	DATA	128KB 06/10/2019 12:58:14
5	EMPLOYEE	DATA	192KB 06/10/2019 12:58:17
6	ETHNICITYCODE	DATA	128KB 06/10/2019 12:58:13
7	GLOBALINDEX	DATA	1MB 06/10/2019 12:58:14
22	STATECODE	DATA	128KB 06/10/2019 12:58:13
23	STATEPOPULATION	DATA	128KB 06/10/2019 12:58:14
24	TAXBRACKET	DATA	128KB 06/10/2019 12:58:14
25	TRANSACTION	DATA	704KB 06/10/2019 12:58:14
26	TRANSACTIONFULL	DATA	7MB 06/10/2019 12:58:14
27	USPOPULATION	DATA	128KB 06/10/2019 12:58:14

Confirm that **27** SAS tables are created.



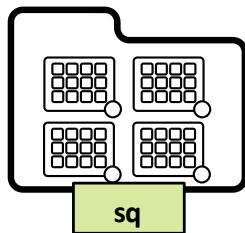
12

Copyright © SAS Institute Inc. All rights reserved.

1.02 Activity – Correct Answer

Main folder location

```
%let path=s:/workshop;
libname sq "s:/workshop/data";
```



Updates the **libname.sas** program to store the LIBNAME statement for the **sq** library

17



Copyright © SAS Institute Inc. All rights reserved.

continued...

1.04 Activity – Correct Answer

1. Remove the asterisk and select only the **FirstName**, **LastName**, and **State** columns. Run the query. View the log and results.

```
proc sql;
select FirstName, LastName, State
  from sq.customer(obs=10);
quit;
```

First Name	Last Name	State
Rodney	Joyner	WI
Jeanne	Ballenger	WA
Brian	Harper	WI
Thomas	Henderson	WA
Becky	Cheers	WI
Allison	Tucker	WA

42

Copyright © SAS Institute Inc. All rights reserved.



Using SAS data set options is a **SAS enhancement**.



continued...

1.04 Activity – Correct Answer

2. Are the results the same using the INOBS=10 option? **Yes, the results are the same.** What about the log? **The INOBS= option restricts the number of rows that PROC SQL retrieves from any single source. The results are the same, but the log returns a warning.**



```
proc sql inobs=10;
select FirstName, LastName, State
  from sq.customer;
quit;
```

```
WARNING: Only 10 records were read from SQ.CUSTOMER due to INOBS= option.
34      quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds
```

43

Copyright © SAS Institute Inc. All rights reserved.



1.04 Activity – Correct Answer

3. Which column was added to the results? **The ROW column.** The **NUMBER option specifies whether the SELECT statement includes the row number of the data as the rows are retrieved.**

```
proc sql inobs=10 number;
select FirstName, LastName, State
  from sq.customer;
quit;
```

Row	First Name	Last Name	State
1	Rodney	Joyner	WI
2	Jeanne	Ballenger	WA
3	Brian	Harper	WI
4	Thomas	Henderson	WA
5	Becky	Cheers	WI
...

44
Copyright © SAS Institute Inc. All rights reserved.



Options in the
PROC SQL
statement are **SAS
enhancements.**



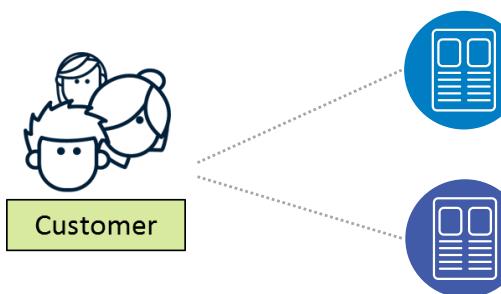
sas

Lesson 2 PROC SQL Fundamentals

2.1 Generating Simple Reports.....	2-3
Demonstration: Creating Simple Reports.....	2-17
Demonstration: Assigning Values Conditionally	2-24
Practice.....	2-27
2.2 Summarizing and Grouping Data	2-31
Demonstration: Summary Functions.....	2-35
Demonstration: Analyzing Groups of Data	2-41
Demonstration: Summarizing Data Using a Boolean.....	2-47
Practice.....	2-50
2.3 Creating and Managing Tables.....	2-53
2.4 Using DICTIONARY Tables.....	2-63
Demonstration: Using DICTIONARY Tables.....	2-66
Practice.....	2-69
2.5 Solutions	2-71
Solutions to Practices	2-71
Solutions to Activities and Questions.....	2-75

2.1 Generating Simple Reports

Creating Simple Reports



- Top 10 customers by income
- No BankID
- CreditScore greater than 700
- Born prior to December 31, 1940
- Employed

3



Copyright © SAS Institute Inc. All rights reserved.

Filtering Rows Using the WHERE Clause

```
PROC SQL <options>;
```

```
  SELECT col-name, col-name
  FROM input-table
  WHERE expression;
```

```
QUIT;
```

```
proc sql;
  select CustomerID, City, State
  from sq.customer
  where State='NC';
quit;
```

Customer ID	City	State
1959010258	Raleigh	NC
1915617620	Greensboro	NC
1955735202	Charlotte	NC
1965297194	Charlotte	NC
1934779042	Forest City	NC
1970405039	Clemmons	NC
1947321333	Durham	NC
1955888968	Charlotte	NC
1977797607	Rocky Mount	NC
1969781309	Dunn	NC
1968472695	King	NC
1951036454	Canton	NC

4



Copyright © SAS Institute Inc. All rights reserved.

The WHERE clause enables you to retrieve only the rows that satisfy a condition.

WHERE clauses can contain any of the columns in a table, including columns that are not specified in the SELECT clause.

The WHERE clause must come after the SELECT clause and the FROM clause.

Character and Numeric Values

WHERE expression;

ABC

Character

`where State = 'NC'`

`where State = "NC"`



123

Numeric

`where Income < 30000`

`where month(DOB) = 9`

Standard

digits 0 – 9
minus sign
decimal point
scientific notation

5
Copyright © SAS Institute Inc. All rights reserved.

Sas

SAS columns can be either character or numeric.

- Character values are case sensitive and must be enclosed in single or double quotation marks. Double quotation marks are a SAS enhancement for character strings that are typically not allowed in DBMSs.
- Numeric values are not enclosed in quotation marks and must be standard numeric values. You cannot include special symbols such as commas or dollar signs.

WHERE Comparison Operators

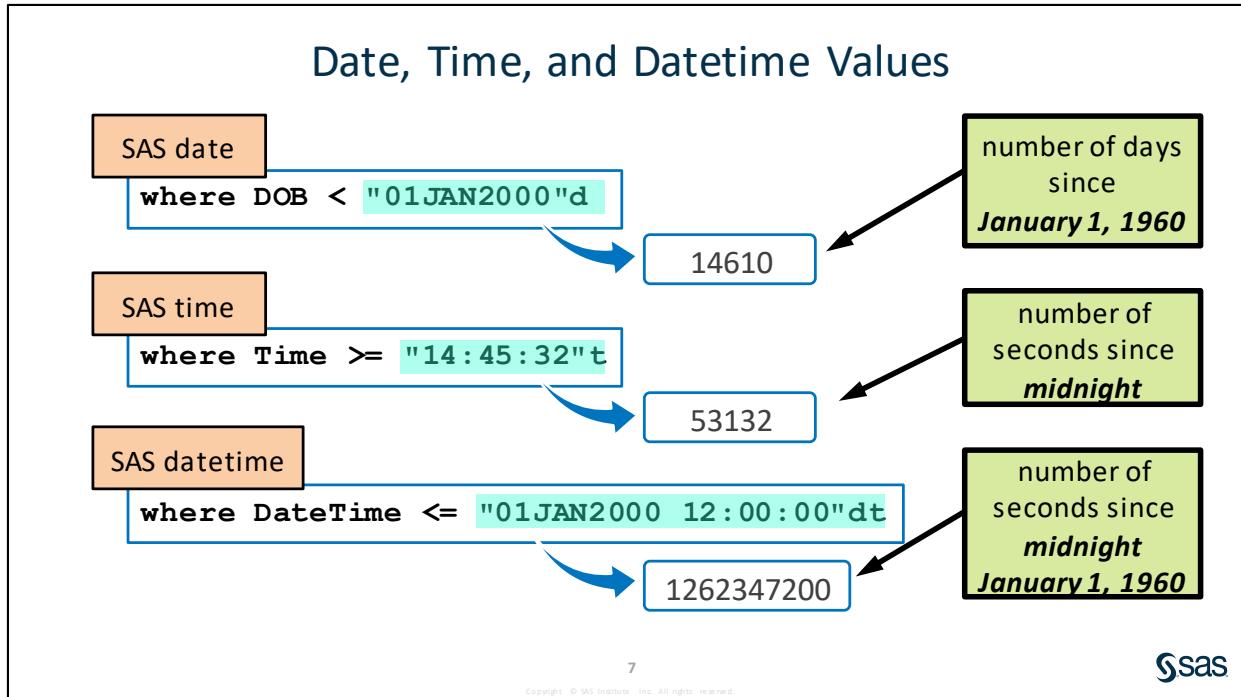


Mnemonic	Symbol
LT	<
GT	>
EQ	=
LE	<=
GE	>=
NE	<>
	~= (EBCDIC)
	^= (ASCII)



6
Copyright © SAS Institute Inc. All rights reserved.

Sas



A SAS *date value* is a date written in the following form: '**ddmmm<yy>yy'd**' or '**"ddmmm<yy>yy"'**d.

- date='1JAN2013'd;
- date='01jan09'D;

A SAS *time constant* is a time written in the following form: '**hh:mm<ss.s>t**' or '**"hh:mm<ss.s>"**t'.

- time='9:25't;
- time='9:25:19pm'T;

A SAS *datetime constant* is datetime value written in the following form:

'ddmmm<yy>yy:hh:mm<ss.s>dt' or '**"ddmmm<yy>yy:hh:mm<ss.s>"dt**'.

- begin='01may12:9:30:00'dt;
- end='31dec13:5:00:00'dt;
- dtime='18jan2003:9:27:05am'DT;

Common Date and Time Functions:

- MONTH(*date*) - returns the month as a numeric value from a SAS date value.
- DAY(*date*) - returns the numeric day of the month from a SAS date value.
- YEAR(*date*) - returns the numeric year from a SAS date value.
- YRDIF(*start-date*, *end-date*, *<basis>*) - returns the difference in years between two dates according to specified day count conventions.
- TODAY() - returns the current date as a numeric SAS date value.
- TIME() - returns the current time of day as a numeric SAS time value.
- DATETIME() - returns the current date and time of day as a SAS datetime value.
- DATEPART(*datetime*) - extracts the date from a SAS datetime value.
- TIMEPART(*datetime*) - extracts a time value from a SAS datetime value.
- MDY(*month*, *day*, *year*) - returns a SAS date value from month, day, and year values.

For more information about the SAS functions, see “SAS Functions and CALL Routines by Category” in the *SAS® 9.4 Functions and CALL Routines: Reference, Fifth Edition* documentation. You can also find the direct link in the Course Links section on the ELP.

Combining Expressions

WHERE expression-1 OR | AND expression-n;



```
proc sql;
  select CustomerID, DOB
    from sq.customer
   where State = 'NY' or
         State = 'NC' or
         State = 'CA';
quit;
```

If **any** expression is true,
select the row.



```
proc sql;
  select CustomerID, DOB
    from sq.customer
   where Income > 30000 and
         State = 'NC';
quit;
```

If **both** expressions are
true, select the row.

8

Copyright © SAS Institute Inc. All rights reserved.



IN Operator

WHERE col-name IN (value-1,...,value-n);
WHERE col-name NOT IN (value-1,...,value-n);

```
where State in ("NC", "GA", "NY")
```



State="NC" or State="GA" or State="NY";

```
where State not in ('NC' 'GA' 'NY')
```



The **NOT** operator forms a negative condition.

9

Copyright © SAS Institute Inc. All rights reserved.

Sas

The IN operator tests for values that match one of a list of values.

You can use the IN operator with character strings or numeric values to determine whether a variable's value is among a list of values. Character values must be enclosed in quotation marks. Numeric values must be in standard form.

2.01 Activity

Open **s102a01.sas** from the **activities** folder and perform the following tasks to find all customers in the states VT, SC, or GA:

1. Complete the WHERE clause to filter for customers in the state of VT and run the query.
2. Add another expression using the OR operator to select only customers from the state of VT **or** SC. How many customers are from either VT or SC?
3. Switch your current expression to use the IN operator. Add the state of GA. How many customers are from either VT, SC, or GA?

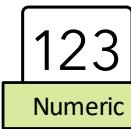
10


Copyright © SAS Institute Inc. All rights reserved.

Special WHERE Operators: Missing Values


ANSI

WHERE col-name IS NULL;
WHERE col-name IS NOT NULL;



Numeric



Character

where Income = .

where Married = " "

The **IS NULL** operator works for both **character** and **numeric** missing values.



12


Copyright © SAS Institute Inc. All rights reserved.

The IS NULL operator

- selects observations in which a variable has a missing value. It can be used for both character and numeric variables.

- is ANSI standard and can be used in DBMS environments that distinguish between missing and null values.
- is the same as, and interchangeable with, the IS MISSING operator in SAS. However, IS MISSING is not ANSI standard.
- can be prefixed with the NOT operator to form a negative condition.

In SAS,

- a blank represents a missing character value
- a . represents a missing numeric value.

2.02 Activity

Open **s102a02.sas** from the **activities** folder and perform the following tasks to find all customers with a nonmissing **CreditScore** value that is less than 500.

1. Examine the query. Add a WHERE clause to find all customers with a **CreditScore** value that is less than 500 and run the query. What do you notice about the values in the **CreditScore** column? How many rows are in your report?
2. Include the AND operator in the WHERE clause to find all rows that are less than 500 and not null. Use a method of your choice. How many rows are in your final report?

Special WHERE Operators: Range of Values

WHERE col-name BETWEEN value-1 AND value-2;

WHERE col-name NOT BETWEEN value-1 AND value-2;

`where CreditScore between 700 and 799`

`where CreditScore >= 700 and CreditScore <= 799`

The BETWEEN-AND operator can be used on **character** values.



Selects rows with values **between and including** the endpoints that you specify

16

Copyright © SAS Institute Inc. All rights reserved.

Special WHERE Operators: Pattern Matching

WHERE col-name LIKE "value";

`where FirstName like "Z%";`

Zachary
Zelda
Zulma
Zula
Zoe
Zandra

`where FirstName like "Z_1%";`

Zelda
Zulma
Zelma
Zola

wildcard for **any number of characters**

wildcard for a **single character**

17

Copyright © SAS Institute Inc. All rights reserved.

The LIKE operator tests for values that match a specified pattern.

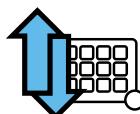
- a percent sign (%) to match any number of characters
- an underscore (_) to match one arbitrary character

Sorting the Output with the ORDER BY Clause

```
PROC SQL <options>;
  SELECT col-name, col-name
    FROM input-table
   WHERE expression
  ORDER BY col-name <DESC> <, col-name>;
QUIT;
```

Sort Methods

- name or alias
- multiple columns
- calculated column
- integer position



The **ORDER BY** clause defines the order in which rows are displayed in the results.



Sas

18

Copyright © SAS Institute Inc. All rights reserved.

Unless an ORDER BY clause is included in the SELECT statement, a particular order to the output rows (such as the order in which the rows are encountered in the queried table) cannot be guaranteed. Without an ORDER BY clause, the order of the output rows is determined by the internal processing of PROC SQL, the default collating sequence of SAS, and your operating environment. Therefore, if you want your query results to appear in a particular order, use the ORDER BY clause.

- The PROC SQL default sort order is ascending.
- When you use an ORDER BY clause, you change the order of the results but not the order of the rows that are stored in the source table.
- PROC SQL sorts missing values before nonmissing values. Therefore, when you specify ascending order, missing values appear first in the query results.
- If multiple ORDER BY columns are specified, the first one determines the major sort order.

2.03 Activity

Open **s102a03.sas** from the **activities** folder and perform the following tasks to sort the report by **CreditScore** and **LastName**:

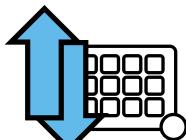
1. Complete the ORDER BY clause and sort by **CreditScore**. Run the query and examine the report. What is the default sort order?
2. Add the keyword DESC after the **CreditScore** column in the ORDER BY clause. Run the query and examine the report. What does the DESC option do?
3. Add a secondary sort column to sort by **LastName**. Run the query. Who is the first customer on the report?
4. Remove **LastName** from the SELECT clause and rerun the query. Are the results still sorted by **LastName** within **CreditScore**?

19



Copyright © SAS Institute Inc. All rights reserved.

Order Columns by Position



```

1   proc sql;
2   select FirstName, LastName, CreditScore
3   from sq.customer
4   where CreditScore > 830
5   order by 3 desc, 2;
6
7   quit;
  
```

First Name	Last Name	CreditScore
Donald	Leyva	848
Elsie	Mathe	848
Christopher	Miras	848
Christopher	Murello	848
Gladys	Taylor	848
Joan	Bekman	847
Helene	Fenton	847

You can also sort by *column position*.



21



Copyright © SAS Institute Inc. All rights reserved.

Enhancing Reports

First Name	Last Name	State	User ID	Income	Date of Birth
Gloria	Tisor	HI	glopetisor6918@invalid.com	91955.4	3456
Grace	Wright	HI	graelwright8418@notreal.com	91379.77	9057
Roberto	Robison	HI	robfrrobison8024@invalid.com	90920.97	7572
Laura	Dumoulin	HI	laukadumoulin5625@invalid.com	88578.71	-1102
Dan	Borgen	HI	dansaborgen9012@n/a.com	84554.62	1111
Howard	Gonzales	HI	howangonzales9219@voidemail.com	83240.89	
Laura	Gardner	HI	laurogardner9020@notreal.com	83230.53	
Sheila	Anson	HI	shedoanson7131@invalid.com	83116.78	
Taylor	Lopez	HI	tayeulopez5923@n/a.com	80413.02	

What does this report show?



Sas

22
Copyright © SAS Institute Inc. All rights reserved.

Enhancing Reports

First Name	Last Name	State	Email Address	Estimated Income	Date of Birth
Gloria	Tisor	HI	glopetisor6918@invalid.com	\$91,955.40	18JUN1969
Grace	Wright	HI	graelwright8418@notreal.com	\$91,379.77	18OCT1984
Roberto	Robison	HI	robfrrobison8024@invalid.com	\$90,920.97	24SEP1980
Laura	Dumoulin	HI	laukadumoulin5625@invalid.com	\$88,578.71	25DEC1956
Dan	Borgen	HI	dansaborgen9012@n/a.com	\$84,554.62	12MAR1990
Howard	Gonzales	HI	howangonzales9219@voidemail.com	\$83,240.89	19FEB1992
Laura	Gardner	HI	laurogardner9020@notreal.com	\$83,230.53	20APR1990
Sheila	Anson	HI	shedoanson7131@invalid.com	\$83,116.78	31DEC1971
Taylor	Lopez	HI	tayeulopez5923@n/a.com	\$80,413.02	23AUG1959
Patricia	Nickey	HI	patmanickey642@notreal.com	\$78,487.53	02JUL1964

Add title

Add labels

Add footnote

Add formats

March 6

Copyright © SAS Institute Inc. All rights reserved.

Adding Titles and Footnotes

TITLE<n> "title-text";

FOOTNOTE<n> "footnote-text";



```
title1 "Customers from Hawaii";
footnotel "March 6";
```

sql query

```
title;
footnote;
```

Adds a title and a footnote

Clears the title and footnote

24

Sas

Copyright © SAS Institute Inc. All rights reserved.

TITLE is a global statement that establishes a permanent title for all reports created in your SAS session. The syntax is the keyword **TITLE** followed by the title text enclosed in quotation marks. You can have up to 10 titles. Specify a number 1 through 10 after the keyword **TITLE** to indicate the line number. **TITLE1** and **TITLE2** are equivalent. **FOOTNOTE** follows the same rules as **TITLE**.

Adding Column Labels



SELECT col-name <LABEL='LABEL'> <FORMAT=formatw.d>

```
select Firstname, LastName, State,
UserID "Email Address",
Income label="Estimated Income" format=dollar16.2,
DOB format=date9.
```



State	Email Address	Estimated Income	Date of Birth
IL	alonotisor6018@invalid.com	\$01,055.40	18-II-1060

25

Sas

Copyright © SAS Institute Inc. All rights reserved.

Adding Column Formats



```
SELECT col-name <LABEL='LABEL'> <FORMAT=formatw.d>
```

```
select FirstName, LastName, State,
       UserID "Email Address",
       Income label="Estimated Income" format=dollar16.2,
       DOB format=date9.
```

State	Email Address	Estimated Income	Date of Birth
II	glopetisor6918@invalid.com	\$91,955.40	18JUN1969
II	graelwright8418@notreal.com	\$91,379.77	18OCT1984
II	robfrrobison8024@invalid.com	\$90,920.97	24SEP1980
II	laukadumoulin5625@invalid.com	\$88,578.71	25DEC1956
II	dansaborgen9012@n/a.com	\$84,554.62	12MAR1990
II	howanganzalec0210@voidemail.com	\$82,240.80	10FEB1900

SAS displays an asterisk if the specified width is not wide enough.

26



Format names always contain a period (.) as a required delimiter. The *w* specifies the total width of the value, including decimal places and special characters. If you do not specify a format width that is large enough to accommodate a value, SAS automatically adjusts to display as much of the stored value as possible.

Depending on the particular format, the *d* specifies the number of decimal places in the value. The *d* is optional for numeric formats. For example, to display the value 1234 as \$1,234 in a report, you can use the DOLLAR6.0 or DOLLAR6. format.

2.04 Activity

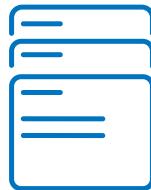
Open **s102a04.sas** from the **activities** folder and perform the following tasks to enhance a report:

1. Examine the query. Add the title "*Customers from Hawaii*" and a footnote using today's date. Run the program and examine the new title and footnote in your report.
2. Apply LABEL="Email Address" to the **UserID** column and LABEL="Estimated Income" to the **Income** column.
3. Apply FORMAT=DATE9. to the **DOB** column and FORMAT=DOLLAR16.2 to the **Income** column. Run the program and examine the report.
4. Change the DOLLAR16.2 format to DOLLAR7.2. Run the program. What happens to the values in the **Income** column?

27



SAS Formats



SAS Formats



PROC FORMAT



You can use the **FORMAT** procedure to create your own format.

30
Copyright © SAS Institute Inc. All rights reserved.

sas

For more information about SAS formats, see “Formats by Category” in the *SAS® 9.4 Formats and Informats: Reference* documentation. You can also find the direct link in the Course Links section on the ELP.

For more information about PROC FORMAT, see “FORMAT Procedure” in the *Base SAS® 9.4 Procedures Guide, Seventh Edition* documentation. You can also find the direct link in the Course Links section on the ELP.

Common Numeric Formats:

Format Name	Example Value	Format Applied	Formatted Value
w.d	12345.67	5.	12346
w.d	12345.67	8.1	12345.7
COMMAw.d	12345.67	COMMA8.1	12,345.7
DOLLARw.d	12345.67	DOLLAR10.2	\$12,345.67
DOLLARw.d	12345.67	DOLLAR10.	\$12,346
YENw.d	12345.67	YEN7.	¥12,346
EUROXw.d	12345.67	EUROX10.2	€12.345,67

Common Date Formats:

Example Value	Format Applied	Formatted Value
21199	DATE7.	15JAN18
21199	DATE9.	15JAN2018
21199	MMDDYY10.	01/15/2018
21199	DDMMYY8.	15/01/18
21199	MONYY7.	JAN2018
21199	MONNAME.	January
21199	WEEKDATE.	Monday, January 15, 2018



Creating Simple Reports

Scenario

Use the WHERE and ORDER BY clauses to create three reports.

Files

- **s102d01.sas**
- **customer** – a SAS table that contains one row per customer

Syntax

```

TITLE<n> 'title-text';
PROC SQL OUTOBS=n;
SELECT col-name <FORMAT=formatw.d> <LABEL='LABEL'>
      FROM input-table(OBS=n)
      WHERE expression
      ORDER BY col-name <DESC>;
QUIT;
TITLE;

```

Notes

- The WHERE clause filters rows based on the expression (or expressions).
- The ORDER BY clause arranges rows based on the listed columns. The default order is ascending. Use DESC after a column name to reverse the sort sequence.
- The OBS= data set option specifies the last rows that SAS processes from a table.
- The OUTOBS= option restricts the number of rows in the results.
- TITLE is a global statement that establishes a permanent title for all reports created in your SAS session.
- The FORMAT= column modifier specifies a SAS format for determining how character and numeric values in a column are displayed by the query expression.
- The LABEL= column modifier specifies a column label.

Demo

1. Open the **s102d01.sas** program in the **demos** folder and find the **Demo** section. Move to **Report 1**. Complete the query.

- a. Complete the WHERE clause to filter for a missing **BankID** value and a value of **CreditScore** greater than 700.

```
proc sql;
select FirstName, LastName, State,
       Income, UserID
  from sq.customer (obs=100)
 where BankID is null and CreditScore > 700;
quit;
```

- b. Complete the ORDER BY clause to arrange rows by descending **Income**.

```
proc sql;
select FirstName, LastName, State, Income, UserID
  from sq.customer (obs=100)
 where BankID is null and CreditScore > 700
   order by Income desc;
quit;
```

- c. Add the column modifiers **FORMAT=DOLLAR16.** to the **Income** column and **LABEL='Email'** to the **UserID** column. Remove the **OBS=** data set option and add the **OUTOBS=10** option in the PROC SQL statement.

```
proc sql outobs=10;
select FirstName, LastName, State,
       Income format=dollar16.,
       UserID label='Email'
  from sq.customer (obs=100)
 where BankID is null and CreditScore > 700
   order by Income desc;
quit;
```

Top 10 Customers by Income without a BankID and CreditScore Over 700 Marketing Report

First Name	Last Name	State	Income	Email
Wade	Estrade	MN	\$102,435	wadmiestrade535@n/a.com
Chester	Dinora	CA	\$101,684	chedadinora8223@fakeemail.com
Stella	Adams	TX	\$96,700	steliadams6814@notreal.com
Lawrence	Duval	CA	\$91,189	lawsdudu19021@fakeemail.com

2. Move to **Report 2**. Complete the query.

- a. Complete the WHERE clause to filter **DOB** prior to 31DEC1940 and where **Employed** equals Y.

```
proc sql;
select CustomerID, State, Zip,
       DOB, UserID,
       HomePhone, CellPhone
  from sq.customer(obs=100)
 where DOB < '31DEC1940'd and Employed='Y';
quit;
```

- b. Complete the ORDER BY clause to arrange rows by descending **DOB**. Run the query and view the results.

```
proc sql;
select CustomerID, State, Zip,
       DOB, UserID,
       HomePhone, CellPhone
  from sq.customer(obs=100)
 where DOB < '31DEC1940'd and Employed='Y'
   order by DOB desc;
quit;
```

- c. Add the column modifiers FORMAT= to the **DOB** and **Zip** columns. Remove the OBS= data set option and highlight and run the query. Examine the log and results.

Note: The Z format writes standard numeric data with leading 0s. Scroll in the results and show ZIP codes with fewer than five digits.

```
proc sql;
select CustomerID, State, Zip format=z5.,
       DOB format=date9., UserID,
       HomePhone, CellPhone
  from sq.customer(obs=100)
...
```

DOB Prior to December 31, 1940 Retirement Campaign						
Customer ID	State	Zip	Date of Birth	User ID	HomePhone	CellPhone
1932141031	OR	97058	16OCT1940	harisfrenette4016@ismissing.com	(541)5006189	(541)4823143
1954791154	MI	48001	17JUL1940	donrohudgins4017@n/a.com	(810)4801076	(810)3649700
1972188828	MN	55001	14JUL1940	maralamack4014@n/a.com	(651)2520753	(651)2685233
1921723973	SC	29690	26JUN1940	danjosumlin4026@voidemail.com	(864)6836887	(864)7331163
1900110357	IL	84664	09JUN1940	radsoniania109@proreal.com	(801)9901301	

End of Demonstration

Creating a New Column

```
proc sql;
  select FirstName, LastName, UserID,
    yrdif(dob, '01jan2019'd) as Age
  from sq.customer(obs=100);
quit;
```

First Name	Last Name	User ID	Age
Rodney	Joyner	rodmjoyner6611@n/a.com	52.9726
Jeanne	Ballenger	jeacaballenger638@fakeemail.com	55.56712
Brian	Harper	bridaharper4714@invalid.com	71.55068
Thomas	Henderson	thoerhenderson6322@ismissing.com	55.10959
Clara	Chen	clarachen15012@n/a.com	72.6044

Create a report that contains customers who are age 70 and over.



Sas

32

Copyright © SAS Institute Inc. All rights reserved.

In addition to selecting columns that are stored in a table, you can create new columns that exist for the duration of the query. These columns can contain text or calculations. PROC SQL writes the columns that you create as if they were columns from the table.

The YRDIF function enables us to calculate the age of all of our customers. We begin with the start date, which is their date of birth value. Then we use the constant of January 1, 2019, as the end date. If you want today's date, you can use the TODAY function(). The third argument, *basis*, describes how SAS calculates the date difference. We specify **Age** to indicate that we want the person's age computed. We will name the column **Age** and run the query to create a new column with each customer's age.

For more information about the YRDIF function, see "YRDIF Function" in the *SAS® 9.4 Functions and CALL Routines: Reference, Fifth Edition* documentation. You can also find the direct link in the Course Links section on the ELP.

2.05 Activity

Open **s102a05.sas** from the **activities** folder and perform the following tasks to find all customers 70 years old and older:

1. Examine and run the query. View the results.
2. Add the expression **yrdif(dob,'01jan2019'd)** in the SELECT clause after **UserID** to create a new column. Run the query and examine the results. What is the name of the new column?
3. Add **as Age** after your function. Run the query and examine the results. What changes?
4. Remove the OBS= data set option in the FROM clause and add a WHERE clause to return rows where **Age** is greater than or equal to 70. Run the query. Did the query run successfully?

33



Copyright © SAS Institute Inc. All rights reserved.

Subsetting Calculated Values

```
proc sql;
select FirstName, LastName, UserID,
       yrdif(dob, '01jan2019'd) as Age
  from sq.customer
 where yrdif(dob, '01jan2019'd)>=70;
quit;
```

```
proc sql;
select FirstName, LastName, UserID,
       yrdif(dob, '01jan2019'd) as Age
  from sq.customer
 where calculated Age>=70;
quit;
```

35



Copyright © SAS Institute Inc. All rights reserved.

When you use a column alias to refer to a calculated value, you must use the CALCULATED keyword with the alias to inform PROC SQL that the value is calculated within the query. You can use an alias to refer to a calculated column in a SELECT clause, a WHERE clause, or an ORDER BY clause. The ORDER BY clause does not require the calculated keyword.

Assigning Values to a New Column Conditionally

CODE	VALUE
M	Married
S	Single
D	Divorced
W	Widowed

CATEGORY	RANGE
Excellent	750+
Good	700 – 749
Fair	650 – 699
Poor	550 – 649
Bad	550 & below

Customer Partial

First Name	Last Name	State	Married	MarriedCategory	CreditScore	Category
Rodney	Joyner	WI	M	Married	711	Good
Jeanne	Ballenger	WA		Unknown	750	Excellent
Brian	Harper	WI	M	Married	790	Excellent
Thomas	Henderson	WA	S	Single	635	Poor
Becky	Cheers	WI	M	Married	716	Good
Alberto	Texter	WI	S	Single	684	Fair
Peter	Schmand	WA	M	Married	617	Poor
Danielle	Bell	WI	M	Married	639	Poor
Robert	Brousseau	WI	M	Married	687	Fair
Sharon	Howell	WA	S	Single	.	Unknown

36



Simple CASE Expression

```
proc sql;
select FirstName, LastName, State, CreditScore,
case
when CreditScore >= 750 then "Excellent"
when CreditScore >= 700 then "Good"
when CreditScore >= 650 then "Fair"
when CreditScore >= 550 then "Poor"
when CreditScore >= 0 then "Bad"
else "Unknown"
end as Category
from sq.customer(obs=1000);
quit;
```



37



Simple CASE Expression

```
proc sql;
  select FirstName, LastName, State, CreditScore,
    case
      when CreditScore >= 750 then "Excellent"
      when CreditScore >= 700 then "Good"
      when CreditScore >= 650 then "Fair"
      when CreditScore >= 550 then "Poor"
      when CreditScore >= 0 then "Bad"
      else "Unknown"
    end as Category
  from sq.customer(obs=1000);
quit;
```

ELSE provides alternate action if no WHEN expressions are true.

The first **WHEN** clause evaluated as *true* determines which value the CASE expression returns.

38


Copyright © SAS Institute Inc. All rights reserved.

CASE-Operand Form

```
proc sql;
  select FirstName, LastName, State, CreditScore,
    case Married
      when "M" then "Married"
      when "D" then "Divorced"
      when "S" then "Single"
      when "W" then "Widowed"
      else "Unknown"
    end as Category
  from sq.customer(obs=1000);
quit;
```

equivalent of
Married="D"

A test of **equality** is implied.



39

Copyright © SAS Institute Inc. All rights reserved.



Assigning Values Conditionally

Scenario

Use the CASE expression to create columns conditionally.

Files

- **s102d02.sas**
- **customer** – a SAS table that contains one row per customer

Syntax

```
PROC SQL;
  SELECT <col-name>, <col-name>,
    CASE <case-operand>
      WHEN condition THEN result-expression
      <WHEN condition THEN result-expression>
      <ELSE result-expression>
    END <AS column>
  FROM input-table;
QUIT;
```

Notes

- CASE expressions enable you to interpret and change some or all the data values in a column to make the data more useful or meaningful.
- You can use a CASE expression anywhere that you can use a column name.

Demo

1. Open the **s102d02.sas** program in the **demos** folder and find the **Demo** section.
2. In the **Simple Case Expression** section:
 - a. Highlight and run the query. Examine the log and results.
 - b. Complete the WHEN and ELSE expressions in the simple CASE expression. Highlight and run the query. Examine the log and results.

```
proc sql;
  select FirstName, LastName, State, CreditScore,
    case
      when CreditScore >=750 then 'Excellent'
      when CreditScore >=700 then 'Good'
      when CreditScore >=650 then 'Fair'
      when CreditScore >=550 then 'Poor'
      when CreditScore >=0 then 'Bad'
    end as CreditCategory
  from sq.customer(obs=1000);
quit;
```

- c. Add the ELSE expression to change remaining values to *Unknown*. Highlight and run the query. Examine the log and results.

```
...
else 'Unknown'
end as CreditCategory
...
```

- d. Add a WHERE clause to filter the table for customers with *Excellent* credit and remove the OBS=1000 data set option. Highlight and run the query. Examine the log and results.

```
proc sql;
select FirstName, LastName, State, CreditScore,
case
when CreditScore >=750 then 'Excellent'
when CreditScore >=700 then 'Good'
when CreditScore >=650 then 'Fair'
when CreditScore >=550 then 'Poor'
when CreditScore >=0 then 'Bad'
else 'Unknown'
end as CreditCategory
from sq.customer(obs=1000)
where calculated CreditCategory = 'Excellent';
quit;
```

3. Move to the **CASE-OPERAND FORM** section.

- Highlight and run the query. Examine the log and results.
- Complete the WHEN and ELSE expressions in the simple CASE-Operand expression. Highlight and run the query. Examine the log and results.

```
proc sql;
select FirstName, LastName, State, CreditScore, Married,
case Married
when 'M' then 'Married'
when 'S' then 'Single'
when 'D' then 'Divorced'
when 'W' then 'Widowed'
else 'Unknown'
end as MarriedCategory
from sq.customer(obs=1000);
quit;
```

End of Demonstration

Syntax Summary

```
SELECT col-name, col-name
  FROM input-table
  WHERE expression
  ORDER BY col-name <DESC>;
```

Query

```
TITLE<n> "title-text";
FOOTNOTE<n> "footnote-text";
FORMAT=formatw.d
LABEL = 'LABEL'
```

Enhance Reports

```
WHERE col-name IS NULL
WHERE col-name LIKE
WHERE col-name BETWEEN-AND
WHERE CALCULATED column
```

Filter Data

```
column AS alias
CASE EXPRESSION
```

Create Columns

```
"01JAN2000 12:00:00"dt
"01JAN2019"d
"14:45:32"t
```

SAS Dates and Times

41

Copyright © SAS Institute Inc. All rights reserved.





Practice

If you restarted your SAS session, open and submit the **libname.sas** program in the course files.

Level 1

1. Querying a Table

The **sq.transactionfull** table contains customer and transaction information. Write a PROC SQL step to generate a report for large transactions that are not related to tuition payments to universities. Use the following requirements as you generate the report:

- a. Write a query to display the following columns in this order: **CustomerName**, **MerchantName**, **Type**, **Service**, and **Amount** from the **sq.transactionfull** table.
 - 1) Select rows that have a transaction **Amount** value greater than \$1,000 and a **Service** value not equal to *University*.
 - 2) Order the rows such that the largest transaction is listed first.
 - 3) Format the **Amount** column with the DOLLAR10.2 format.
 - 4) Label **CustomerName** as **Customer Name**, and **Amount** as **Transaction Amount**.
 - 5) Add the following title: **Large Non-Educational Transactions**
 - 6) Run the program and review the results.

Partial Results

Large Non-Educational Transactions				
Customer Name	MerchantName	Type	Service	Transaction Amount
Bower, Omar Randy	Big Box Store	Department Store	Warehouse/Big Box	\$4,072.08
Lefeld, Linda Erica	Elegant Goods Department Store	Department Store	High End	\$3,699.23
Lefeld, Linda Erica	Happy Home Insurance, Inc.	Insurance	Home Insurance	\$3,541.74
Bower, Omar Randy	Big Box Store	Department Store	Warehouse/Big Box	\$3,489.68
Bower, Omar Randy	Big Box Store	Department Store	Warehouse/Big Box	\$3,179.95

- b. How much did Pamela Maiden spend on Home Insurance?

2. Using a Function in a Filter

The **sq.customer** table contains customer demographic and banking information. Write a query to create a report for customers with May birthdays in the state of North Carolina. Use the following requirements as you generate the report:

- a. Using the **sq.customer** table as input, display the following columns in this order: **FirstName**, **LastName**, **DOB**
 - 1) Add a filter to subset the rows based on the month value 5 for **DOB** and a **State** value of NC.
 - 2) Order the rows by descending **DOB**.
 - 3) Format the **DOB** column with DATE9.

- 4) Add the following title: **Customers with May Birthdays in North Carolina**
- 5) Run the program and review the results.

Partial Results

Customers with May Birthdays in North Carolina		
First Name	Last Name	Date of Birth
Suzanne	Kaplan	21MAY2010
Virginia	James	26MAY2005
Edward	Lotspeich	24MAY2005
Timothy	Montgomery	01MAY2005
Sandra	Sparling	21MAY2004
Patrick	Sanchez	23MAY2001
Guadalupe	Giuliani	17MAY2001

- b. What is Amy Murray's date of birth?

Level 2

3. Working with Datetime Values

The **sq.transactionfull** table contains a list of customer and transaction information. Write a query to create a report displaying transactions that took place in November and December of any year. Use the following requirements as you generate the report:

- a. Use the **sq.transactionfull** table to select the following columns in this order: **CustomerName, MerchantName, Amount**.
 - 1) Create a new column named **TransactionDate** by using the DATEPART function to extract the SAS date value from the **DateTime** column. Format the new column using the DATE9. format.
 - 2) Filter the data to select rows where the month of the transaction date is November or December and the **Service** value is not equal to *University*.
 - 3) Order the report by the original **DateTime** column.
 - 4) Format the **Amount** column with DOLLAR10.2.
 - 5) Label **CustomerName** as **Customer Name**, **MerchantName** as **Merchant Name**, **Amount** as **Transaction Amount**, and **TransactionDate** as **Transaction Date**.
 - 6) Add the following title: **November/December Transactions**
 - 7) Run the program and review the results.

Partial Results

November/December Transactions			
Customer Name	Merchant Name	Transaction Amount	Transaction Date
Kennedy, Daniel Eric	Big Burgers, Inc.	\$38.69	01NOV2018
Lefeld, Linda Erica	Alar Air, Inc.	\$272.50	01NOV2018
Balo, Crystal Diane	Big Burgers, Inc.	\$33.65	01NOV2018
Bowers, Douglas Tim	Economical Superstore	\$22.66	01NOV2018
Lefeld, Linda Erica	Livable Landscaping, LLC	\$149.23	01NOV2018
Bowers, Iva Betty	Big Burgers, Inc.	\$36.01	01NOV2018

- b. What value of **Merchant Name** is on the first documented transaction in December?

Challenge

4. Conditional Processing with a Dynamic Title

Write a query to conditionally create a new column based on a customer's age. Use the following requirements as you generate the report:

- a. Using the **sq.customer** file as input, use a character format to display the first initial of **FirstName** labeled as **Initial** and to display **LastName** labeled as **Last Name**. In addition, display **CreditScore** with the label **Credit Score**.
 - 1) Create a new column named **Generation** based on the customer's date of birth (**DOB**), using the following logic:
 - a) **DOB** between 01JAN1928 and 31DEC1945 **Generation** = *Silent*.
 - b) **DOB** between 01JAN1946 and 31DEC1964 **Generation** = *Boomer*.
 - c) **DOB** between 01JAN1965 and 31DEC1979 **Generation** = *GenX*.
 - d) **DOB** between 01JAN1980 and 31DEC1996 **Generation** = *Millennial*.
 - e) **DOB** on or after 01JAN1997 **Generation** = *Post-Millennial*.
 - f) Otherwise, set **Generation** = *Unknown*.
 - 2) Filter the data to select rows where **CreditScore** is not missing and **State** = *VT*.
 - 3) Order the report by descending **CreditScore** within each value of **Generation**.
 - 4) Add a dynamic title that specifies the exact date the report was run: **Created on Dynamic Date Value** (Hint: Research the %QSYSFUNC macro function). The date value in the title shown below is formatted using the WEEKDATE format.
 - 5) Run the program and review the results.

Partial Results (The date in the title changes based on when the program is run.)

Created on Friday, May 24, 2019				
First Name	Last Name	Date of Birth	Credit Score	Generation
G	Reece	22FEB1964	784	Boomer
C	Hebert	22APR1957	749	Boomer
J	Hockema	27JUN1953	734	Boomer
M	Santone	20NOV1951	722	Boomer
C	Smitley	13MAY1961	717	Boomer
R	Angst	30JUL1946	703	Boomer
L	Parks	12JUN1954	702	Boomer
K	Armistead	28JAN1952	693	Boomer
C	Wiseley	09DEC1964	690	Boomer

- b. Is the date in your title today's date?

End of Practices

2.2 Summarizing and Grouping Data

Eliminating Duplicate Rows

customer

First Name	Last Name	State
Kendra	Mchaney	GA
Lance	Geldmacher	TX
Arnold	Gulla	TX
Joshua	Klavuhn	UT
Jose	Lange	TX
Tommy	Gangwer	NY
Tiffany	Paulson	NY
Frances	Smith	FL
Aurelia	Pierce	FL
Tim	Salisbury	CA
Bruce	Kroon	NY
John	Dawkins	TX
Lee	Rasinski	FL
David	Springston	CA
Jeanne	Kulaga	TX

44

Copyright © SAS Institute Inc. All rights reserved.

sas

Distinct Keyword

```
PROC SQL;
  SELECT DISTINCT col-name <,col-name>
    FROM input-table
  QUIT;
```

The **DISTINCT** keyword applies to all columns in the **SELECT** list.

```
proc sql;
  select distinct State
    from sq.customer;
  quit;
```

State
AK
AL
AR
AZ
CA
CO
CT

45

Copyright © SAS Institute Inc. All rights reserved.

sas

Use the **DISTINCT** keyword to eliminate duplicate rows. The **DISTINCT** keyword applies to all columns in the **SELECT** list. One row is displayed for each unique combination of values.

When you specify all of a table's columns in a SELECT clause with the DISTINCT keyword, PROC SQL eliminates duplicate rows, or rows in which the values in all of the columns match, from the results.

2.06 Activity

Open **s102a06.sas** from the **activities** folder and perform the following tasks to eliminate duplicate values in a table:

1. Examine and run the query. View the results.
2. Change the **State** column in the SELECT clause to the **Employed** column. Run the query. What does this query show?
3. Add the **Married** column in the SELECT clause after the **Employed** column. Run the query. What does this query show?

46



Summarizing Data

statepopulation

Name	PopEstimate1	PopEstimate2	PopEstimate3
AL	4864745	4875120	4887871
AK	741504	739786	737438
AZ	6945452	7048876	7171646
AR	2990410	3002997	3013825
CA	39209127	39399349	39557045
CO	5540921	5615902	5695564
CT	3578674	3573880	3572665
DE	949216	957078	967171
DC	686575	695691	702455
FL	20629982	20976812	21299325

$f(\cdot)$

summarize data

48



Summary Functions: Down a Column

ANSI

SELECT summary function(column);

```
proc sql;
select max(PopEstimate1) as MaxEst format=comma16.,
       min(PopEstimate1) as MinEst format=comma16.,
       avg(PopEstimate1) as AvgEst format=comma16.
  from sq.statepopulation;
quit;
```

MaxEst	MinEst	AvgEst
39,209,127	584,290	6,278,420

49


Copyright © SAS Institute Inc. All rights reserved.

When you use a summary function with a single argument, nonmissing values are totaled down a column.

Summary Functions: Across a Row

SELECT summary function(column1, column-n);

```
proc sql;
select Name, PopEstimate1, PopEstimate2, PopEstimate3,
       max(PopEstimate1, PopEstimate2, PopEstimate3)
       as MaxEst format=comma16.
  from sq.statepopulation;
quit;
```

Name	PopEstimate1	PopEstimate2	PopEstimate3	MaxEst
AL	4864745	4875120	4887871	4,887,871
AK	741504	739786	737438	741,504
AZ	6945452	7048876	7171646	7,171,646
AR	2990410	3002997	3013825	3,013,825
CA	39209127	39399349	39557045	39,557,045
CO	5540921	5615902	5695564	5,695,564
CT	3578674	3573880	3572665	3,578,674

50


Copyright © SAS Institute Inc. All rights reserved.

A summary function with multiple arguments, nonmissing values are totaled across a row.

Commonly Used Summary Functions

SQL	SAS	Returned Value
AVG	MEAN	Mean (average) value
COUNT	FREQ, N	Number of nonmissing values
MAX	MAX	Largest value
MIN	MIN	Smallest nonmissing value
SUM	SUM	Sum of nonmissing values
	NMISS	Number of missing values
	STD	Standard deviation
	VAR	Variance

When more than one argument is used within an SQL aggregate function, the function is no longer considered to be an SQL aggregate or summary function. If there is a like-named Base SAS function, then PROC SQL executes the Base SAS function, and the results that are returned are based on the values for the current row. If no like-named Base SAS function exists, then an error occurs. For example, if you use multiple arguments for the AVG function, an error occurs because there is no AVG function for Base SAS.

For more information about SAS functions, see the *SAS® 9.4 Functions and CALL Routines: Reference, Fifth Edition* documentation. You can also find the direct link in the Course Links section on the ELP.



Summary Functions

Scenario

Use summary functions to analyze a table.

Files

- **s102d03.sas**
- **statepopulation** – a SAS table that contains forecasted US population information by states

Syntax

```
SELECT summary function(column);
SELECT summary function(column1, column-n);
```

Notes

- When you use a summary function with **multiple arguments**, nonmissing values are totaled **across a row**.
- When you use a summary function with a **single argument**, nonmissing values are totaled **down a column**.

Demo

1. Open the **s102d03.sas** program in the **demos** folder and find the **Demo** section. Highlight and run the DESCRIBE TABLE statement and query in the **Explore the sq.statepopulation table** section. Examine the log and results.

```
proc sql inobs=10;
describe table sq.statepopulation;
select Region, Division, Name, PopEstimate1, PopEstimate2,
       PopEstimate3
      from sq.statepopulation;
quit;
```

2. Move to **Method 1 – Down a Column**. In the query complete the following:

- a. Run the query and examine the results.

```
proc sql;
select count(PopEstimate1) as TotalStates,
       mean(PopEstimate1) as Mean format=commal6.
      from sq.statepopulation;
quit;
```

- b. In the SELECT clause, add three columns to find the standard deviation, minimum, and maximum of **PopEstimate1** using the STD, MIN, and MAX functions. Use the COMMA16. format for all new columns. Highlight and run the query. Examine the log and results.

```
proc sql;
select count(PopEstimate1) as TotalStates,
       mean(PopEstimate1) as Mean format=comma16.,
       std(PopEstimate1) as StdDev format=comma16.,
       min(PopEstimate1) as Min format=comma16.,
       max(PopEstimate1) as Max format=comma16.
  from sq.statepopulation;
quit;
```

TotalStates	Mean	StdDev	Min	Max
52	6,278,420	7,182,307	584,290	39,209,127

- c. Move to **SAS Method – PROC MEANS** below the query. SAS has procedures to do similar summarization. Highlight and run the MEANS procedure. Examine the log and results.

```
proc means data=sq.statepopulation maxdec=0;
  var PopEstimate1;
run;
```

Analysis Variable : PopEstimate1				
N	Mean	Std Dev	Minimum	Maximum
52	6278420	7182307	584290	39209127

3. Move to **Method 2 – Across a Column**. In the query, complete the following:

- a. The new column named **Mean** generates the average population estimate for the next three years for each state using the AVG function. In addition, the MIN and MAX functions are used to create new columns that generate the minimum and maximum population estimate. Highlight and run the query. Examine the log and results.

Note: When more than one argument is used within an SQL aggregate function, the function is no longer considered to be an SQL aggregate or summary function. This causes an error.

```
proc sql;
select Name,
       PopEstimate1 format=comma16.,
       PopEstimate2 format=comma16.,
       PopEstimate3 format=comma16.,
       avg(PopEstimate1, PopEstimate2, PopEstimate3) as Mean
             format=comma16.,
       min(PopEstimate1, PopEstimate2, PopEstimate3) as Min
             format=comma16.,
       max(PopEstimate1, PopEstimate2, PopEstimate3) as Max
             format=comma16.
  from sq.statepopulation;
quit;
```

- b. Replace the AVG function with the MEAN function. Highlight and run the query. Examine the log and results.

```
...
mean(PopEstimate1, PopEstimate2, PopEstimate3) as Mean
    format=comma16.
...
```

- c. In the MAX function, change the arguments to **of PopEstimate1-PopEstimate3**. Highlight and run the query. Examine and discuss the syntax error.

```
...
mean(PopEstimate1, PopEstimate2, PopEstimate3) as Mean
    format=comma16.,
min(PopEstimate1, PopEstimate2, PopEstimate3) as Min
    format=comma16. ,
max(of PopEstimate1-PopEstimate3) as Max
    format=comma16.
...
```

End of Demonstration

Summarizing Data Using the COUNT Function

An asterisk specifies all rows.

SELECT COUNT(argument);

```
proc sql;
  select count(*) as TotalCustomers format=comma12.
    from sq.customer;
quit;
```

TotalCustomers
100,004

53



The *COUNT function* counts the number of rows returned by a query.

- The * (asterisk) counts all rows in a table or group.
- A column name as the argument counts the number of nonmissing values in that column.

2.07 Activity

Open **s102a07.sas** from the **activities** folder and perform the following tasks to summarize a table using the COUNT function:

1. Examine and run the query. View the results. Why is the value of **MaritalStatus** different from the value of **TotalRows**?
2. Inside the COUNT function, add the DISTINCT keyword in front of the **Married** column and run the query. What does the new report show?

54



Grouping Data

How many customers are in each **state**?

What is the average **credit score** of those customers?

Customer

56

Copyright © SAS Institute Inc. All rights reserved.

sas

Grouping Data

The **GROUP BY** clause summarizes groups of data by a specified **column or columns**.

```
SELECT col-name, summary function(column)
FROM input-table
WHERE expression
GROUP BY col-name <,col-name>
ORDER BY col-name DESC;
```

```
select State, count(*) as TotalCustomers format=comma7 .
  from sq.customer
 where BankID is not null
 group by State
 order by TotalCustomers desc;
```

State	TotalCustomers
CA	17,224
TX	9,416
NY	6,508
IL	5,427
FL	4,852
OH	2,531

57

Copyright © SAS Institute Inc. All rights reserved.

sas

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 .
- group by multiple columns, separating the column names with commas within the GROUP BY clause. You can use aggregate functions with any of the columns that you select.

When you use a GROUP BY clause without an aggregate function, PROC SQL treats the GROUP BY clause as if it were an ORDER BY clause, displaying a corresponding message in the log.

Filtering Grouped Data

The **HAVING** clause instructs PROC SQL how to *filter* the data after the data is summarized.

```
SELECT col-name, summary function(column)
      FROM input-table
      WHERE expression
      GROUP BY col-name <,col-name>
      HAVING expression
      ORDER BY col-name DESC;
```

```
select State, count(*) as TotalCustomers format=comma7.
      from sq.customer
      where BankID is not null
      group by State
      having TotalCustomers > 6000
      order by TotalCustomers desc;
```

State	TotalCustomers
CA	17,224
TX	9,416
NY	6,508



You can use a HAVING clause with a GROUP BY clause to filter grouped data. The HAVING clause affects groups in a way that is similar to how a WHERE clause affects individual rows. When you use a HAVING clause, PROC SQL displays only the groups that satisfy the HAVING expression.

Because the WHERE clause is evaluated before a row is available for processing and determines which individual rows are available for grouping, you cannot use a WHERE clause to subset grouped rows by referring to the calculated summary column.

You can still use the WHERE clause with the HAVING clause. The WHERE clause filters the rows coming from the input table or tables, and the HAVING clause filters the aggregated data post-summarization.



Analyzing Groups of Data

Scenario

Use the GROUP BY clause to group data and produce summary statistics for each group.

Files

- **s102d04.sas**
- **customer** – a SAS table that contains US population information by country, region, and state

Syntax

```
PROC SQL;
  SELECT col-name, col-name
    FROM input-table
    WHERE expression
    GROUP BY col-name
    HAVING expression
    ORDER BY col-name <DESC>;
QUIT;
```

Notes

- The GROUP BY clause enables you to break query results into subsets of rows. When you use the GROUP BY clause, you use an aggregate function in the SELECT clause
- The HAVING clause is a “WHERE” clause for grouped data.

Demo

1. Open the **s102d04.sas** program in the **demos** folder and find the **Demo** section. Notice that the query creates a report of the **State** column in the **customer** table and limits the output to 1,000 rows. Highlight and run the query. Examine the log and results.

Note: The table is limited for development purposes because the **customer** table has more than 100,000 rows. After we finalize our query, we can run it on the entire table.

Note: When you use a GROUP BY clause without an aggregate function, PROC SQL treats the GROUP BY clause as if it were an ORDER BY clause, displaying a corresponding message in the log.

```
proc sql;
  select State
    from sq.customer(obs=1000)
    group by State;
quit;
```

2. Modify the query to count the total number of customers in each state by using the COUNT function. Name the column **TotalCustomers**. Highlight and run the query. Examine the log and results.

```
proc sql;
select State, count(*) as TotalCustomers format=comma7.
  from sq.customer(obs=1000)
  group by state;
quit;
```

3. Add an ORDER BY clause to the query to sort the report by descending **TotalCustomers**. Remove the OBS=1000 data set option and run the final query. Examine the log and results.

```
proc sql;
select State, count(*) as TotalCustomers format=comma7.
  from sq.customer(obs=1000)
  group by state
  order by TotalCustomers desc;
quit;
```

4. Replace **State** in the SELECT and GROUP BY clauses with **BankID**. Highlight and run the query. Examine the log and the results.

Note: Missing values are grouped and summarized.

```
proc sql;
select BankID, count(*) as TotalCustomers format=comma7.
  from sq.customer
  group by BankID
  order by TotalCustomers desc;
quit;
```

5. Add the **Employed** column after **BankID** in the SELECT and GROUP BY clauses. Highlight and run the query. Examine the log and the results.

```
proc sql;
select BankID, Employed,
      count(*) as TotalCustomers format=comma7.
  from sq.customer
  group by BankID, Employed
  order by TotalCustomers desc;
quit;
```

6. Add a WHERE clause to filter for **TotalCustomers** greater than 10,000. Highlight and run the query. Examine the log and the results.

Note: Because the WHERE clause is evaluated before a row is available for processing and determines which individual rows are available for grouping, you cannot use a WHERE clause to subset grouped rows by referring to the calculated summary column.

```
proc sql;
select BankID, Employed,
      count(*) as TotalCustomers format=comma7.,
      avg(CreditScore) as avgCreditScore format=3.
  from sq.customer
  where calculated TotalCustomers > 10000
  group by BankID, Employed
  order by TotalCustomers desc;
quit;
```

7. Remove the WHERE clause and insert a HAVING clause below the GROUP BY clause. Highlight and run the query. Examine the log and the results.

Note: The order of the clauses is required.

```
proc sql;
  select BankID, Employed,
         count(*) as TotalCustomers format=comma7 .
    from sq.customer
  where calculated TotalCustomers > 10000
  group by BankID, Employed
  having TotalCustomers > 10000
  order by TotalCustomers desc;
quit;
```

BankID	Employed	TotalCustomers
101010101	Y	22,923
101010101	N	17,252
202020202	Y	16,930
303030303	Y	13,954
202020202	N	13,011
303030303	N	10,980

End of Demonstration

Extracting Data from a Datetime Value

DATEPART(datetime-value)

TIMEPART(datetime-value)

```
select DateTime,
       datepart(DateTime) as Date format=date9.,
       timepart(DateTime) as Time format=time.,
       Amount
  from sq.transaction;
```

Date Time	Date	Time	Amount
01JAN18:11:21:01	01JAN2018	11:21:01	88.65
01JAN18:12:05:32	01JAN2018	12:05:32	16437.22
01JAN18:12:12:30	01JAN2018	12:12:30	149.23
01JAN18:12:26:20	01JAN2018	12:26:20	29.9
01JAN18:13:18:01	01JAN2018	13:18:01	614.53
01JAN18:14:50:26	01JAN2018	14:50:26	18025.07

Extract the **date** and **time** values from the **DateTime** column.



Sas

60

Copyright © SAS Institute Inc. All rights reserved.

Summarizing Data by Month

```
select month(datepart(DateTime)) as Month,
       Median(Amount) as MedianSpent format=dollar16.
  from sq.transaction
 group by Month;
```

Wrap the DATEPART function inside the MONTH function to extract the numeric month.

Month	MedianSpent
1	\$34
2	\$46
3	\$37
4	\$28
5	\$28
6	\$28
7	\$28
8	\$26
9	\$27

61

Copyright © SAS Institute Inc. All rights reserved.

Sas

2.08 Activity

Open **s102a08.sas** from the **activities** folder and perform the following tasks to summarize data using date functions:

1. Examine and run the query. View the results. Which month has the highest value for **MedianSpent**?
2. Replace the MONTH function with the QTR function. Change the name of the **Month** column to **Qtr**. Run the query. What is the error?
3. Replace **Month** in the GROUP BY clause with **Qtr**. Run the query. Which quarter has the highest value for **MedianSpent**?

62



Copyright © SAS Institute Inc. All rights reserved.

Counting Rows That Meet a Specified Criterion

customer

FirstName	MiddleName	LastName	Gender	DOB	Employed
Rodney	Matthew	Joyner	M	2202	Y
Jeanne	Carol	Ballenger	F	1254	N
Brian	Dallas	Harper	M	4584	N
Thomas	Eric	Henderson	M	1421	N
Becky	Danna	Cheers	F	5365	N
Alberto	Daryl	Texter	M	15193	N
Peter	Douglas	Schmand	M	3971	Y
Danielle	Julie	Bell	F	11446	Y

customercount

State	Under25	Over64
AK	60	57
AL	299	238
AR	169	135
AZ	677	558
CA	3867	3176
CO	401	366

Create a table that retrieves the number of customers in each state who are **under 25** and **over 64**.



64



Copyright © SAS Institute Inc. All rights reserved.

Using Boolean Expressions

```
create table CustomerCount as  
select State,  
       yrdif(DOB, "01JAN2020"d, 'age') < 25 as Under25  
  from sq.customer;
```

If Age is less than 25, the value is 1 (true).

If Age is greater than 25, the value is 0 (false).



State	Under25
WI	0
WA	0
WI	0
WA	0
WI	0
WI	1
WA	0

65

Copyright © SAS Institute Inc. All rights reserved.



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



Summarizing Data Using a Boolean

Scenario

Use functions to summarize the number of customers under 25 and over 64 for each state.

Files

- **s102d05.sas**
- **customer** – a SAS table that contains customer information

Syntax

```
YRDIF(start-date, end-date <,'basis'>)
```

Notes

YRDIF Function

- The start date and end date specify a SAS data value.
- The *basis* value identifies a character constant or variable that describe how SAS calculates a date difference.
- If you do not specify a third argument, **Age** becomes the default value for *basis*.
- The **Age** basis specifies that a person's age is computed.

Demo

1. Open the **s102d05.sas** program in the **demos** folder and find the **Demo** section. Run the query and examine the results.

```
proc sql inobs=1000;
create table CustomerCount as
select State,
       yrdif(DOB,'01JAN2020'd,'age') as Age
     from sq.customer;
quit;
```

2. Add the less than operator after the YRDIF function test if the row is under 25 years old. Rename the column **Under25**. Run the query and examine the results.

```
proc sql inobs=1000;
create table CustomerCount as
select State,
       yrdif(DOB,'01JAN2020'd,'age')<25 as Under25
     from sq.customer;
quit;
```

3. Copy the expression. Replace the < comparison operator with the > comparison operator. Change the value from **25** to **64** and the name from **Under25** to **Over64**. Run the query and examine the results.

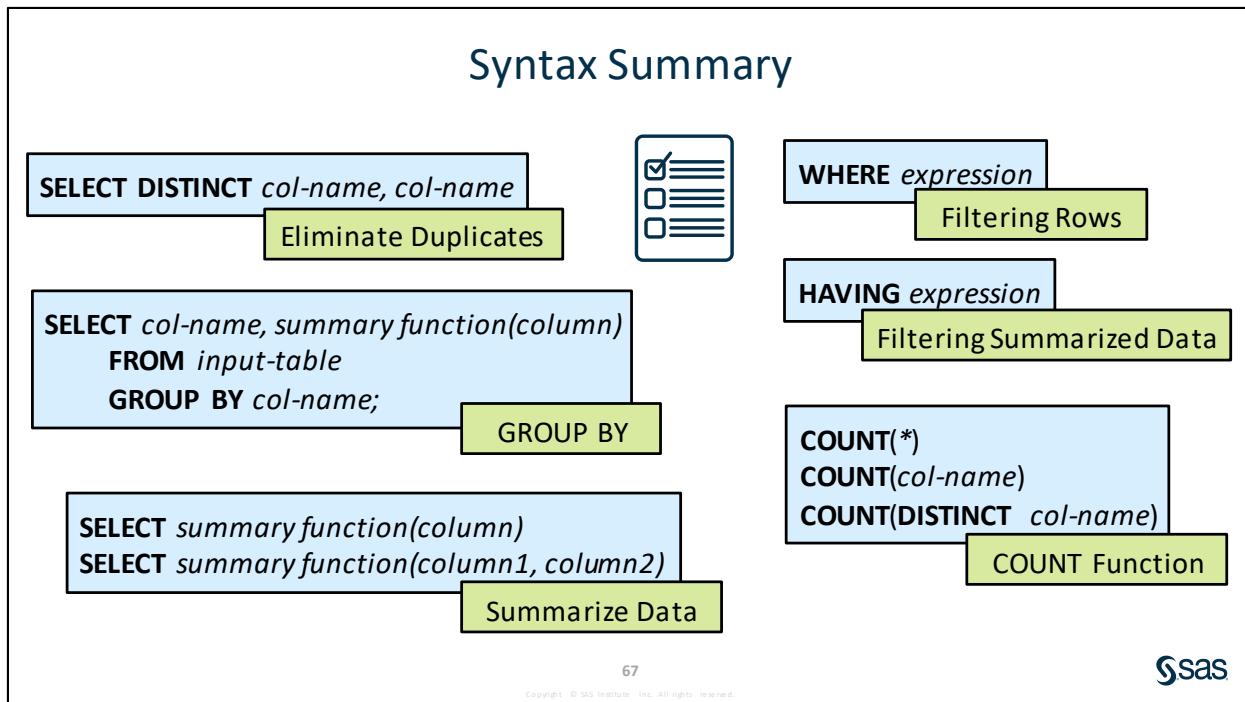
```
proc sql inobs=1000;
create table CustomerCount as
select State,
       yrdif(DOB,'01JAN2020'd,'age')<25 as Under25,
       yrdif(DOB,'01JAN2020'd,'age')>64 as Over64
  from sq.customer;
quit;
```

4. Summarize the data by wrapping each new column with the SUM function to add all the values of 1 to count the number of customers. Add a GROUP BY clause with the **State** column. Remove the INOBS= option. Run the query and examine the results.

```
proc sql inobs=1000;
create table CustomerCount as
select State,
       sum(yrdif(DOB,'01JAN2020'd,'age')<25) as Under25,
       sum(yrdif(DOB,'01JAN2020'd,'age')>64) as Over64
  from sq.customer
 group by State;
quit;
```

State	Under25	Over64
AK	60	57
AL	299	238
AR	169	135
AZ	677	558
CA	3867	3176
CO	401	363
CT	158	127
DC	77	58
DE	18	24
FL	1052	951

End of Demonstration





Practice

Level 1

5. Eliminating Duplicates

The **sq.globalfull** table contains estimated financial information by geographic region and country for the population age 15 years and older.

- a. Write a query to generate a report that displays the unique **CountryCode** values from **sq.globalfull**.
 - 1) Order the report by the **CountryCode**.
 - 2) Add the title **Unique Country Codes**.
 - 3) Run the program and review the results.

Partial Results

Unique Country Codes
CountryCode
AFG
AGO
ALB
ARE
ARG
ARM
AUS
AUT

- b. Modify the query to produce a count of the unique **CountryCode** values.
 - 1) Name the result of the count **CountryCount**.
 - 2) Add the title **Count of Unique Country Codes**.
 - 3) Run the program and review the results.

Results

Count of Unique Country Codes
CountryCount
151

- c. How many unique country codes are in the **sq.globalfull** table?

Level 2

6. Grouping and Summarizing Data

Create a new program.

- a. Write a query to generate a report that identifies which customers have the greatest percentage of suspiciously large transactions (over \$500). Use the following requirements as you generate the report:
- 1) Using the **sq.transactionfull** table as input, select and group the report by **CustomerID**.
 - 2) Create the following columns:
 - a) **TotalTransactions** using the COUNT(*) function to count the number of transactions for each value of **CustomerID**.
 - b) **SuspiciousTransactions** as SUM(**Amount** >= 500) to count the number of transactions greater than 500.
 - c) **PCTSuspicious** by dividing **SuspiciousTransactions** by **TotalTransactions**. Format the new column with PERCENT8.2.
 - 3) Select only transactions where the **Service** value is not equal to *University*.
 - 4) Filter the output to display only summary rows where **PCTSuspicious** > .05.
 - 5) Order the report by descending **PCTSuspicious**.
 - 6) Add the title **Customers with High Percentage of Suspicious Transactions**.
 - 7) Run the program and review the results.

Results

Customers with High Percentage of Suspicious Transactions				
Customer ID	TotalTransactions	SuspiciousTransactions	PCTSuspicious	
1973179983	362	70	19.34%	
1998323808	442	48	10.86%	
1990559364	425	42	9.88%	
1989612017	419	39	9.31%	
1978669535	398	23	5.78%	

- b. Which customer ID had the highest percentage of suspicious transactions?

Challenge

7. Grouping and Summarizing Data with Calculations

The **sq.globalfull** table contains estimated financial information by geographic region and country for the population age 15 years and older.

- a. Write a query to generate a report that displays how many people in each region ages 15 or older will borrow for health or medical purposes next year and three years from now. The report will include the one-year and three-year forecasted counts and a percent increase between the two.

- 1) Use the **sq.globalfull** table and select and group by **Region**.
- 2) Calculate three new columns.
 - a) The estimated number of people who will borrow for health or medical purposes by finding the mean of **EstYear1Pct**, converting it to a percentage by dividing by 100, and multiplying it by the sum of **EstYear1Pop**.
 - (1) Name the column **EstCount1** and format it using commas.
 - (2) Use this expression as a guide: ***mean(EstYear1PCT)/100)*sum(EstYear1Pop)***
 - b) The estimated number of people who will borrow for health or medical purposes by finding the mean of **EstYear3Pct**, converting it to a percentage by dividing by 100, and multiplying it by the sum of **EstYear3Pop**.
 - (1) Name the column **EstCount3** and format it using commas.
 - (2) Hint: Copy and modify the expression used in step 1 to refer to the three-year population estimate columns.
 - c) Subtract **EstCount1** from **EstCount3** and divide the result by **EstCount1** to determine the percent increase. Name the column **PctIncrease** and format it using the PERCENT format.
- 3) Filter for rows where **IndicatorName** is *Borrowed for health or medical purposes (% age 15+)*.
- 4) Order by descending **PctIncrease**.
- 5) Add an appropriate title.
- 6) Run the program and review the results.

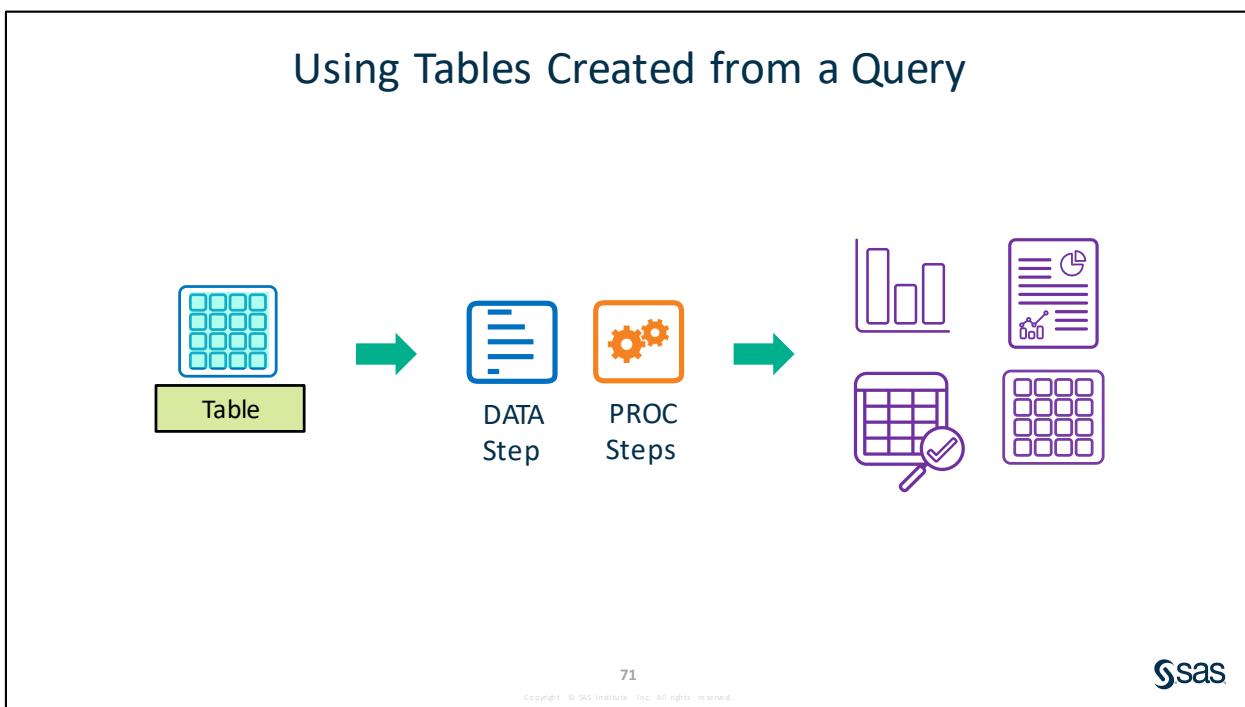
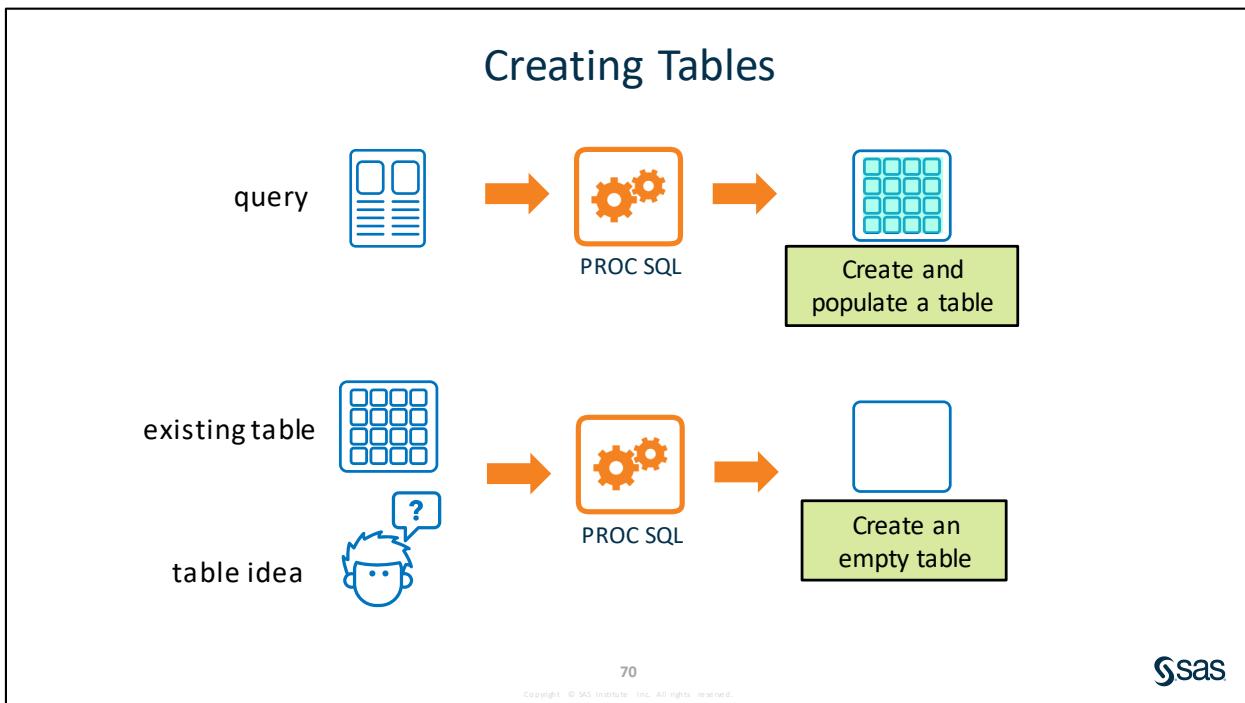
Results

Forecasted Regions who will Borrow for Health or Medical Purpose			
Region	EstCount1	EstCount3	PctIncrease
Europe & Central Asia	45,423,668	49,006,648	7.89%
Middle East & North Africa	29,723,940	30,317,485	2.00%
South Asia	165,724,251	159,684,044	(3.64%)
Sub-Saharan Africa	103,320,403	94,698,557	(8.34%)
Latin America & Caribbean	53,674,814	48,289,383	(10.0%)
North America	16,078,011	13,314,371	(17.2%)
East Asia & Pacific	179,144,288	133,436,020	(25.5%)

- b. Which region had the largest percent decrease from year 1 to year 3?

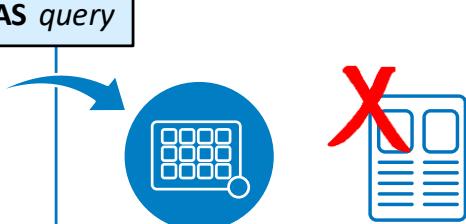
End of Practices

2.3 Creating and Managing Tables



Creating a Table from a Query Result

```
CREATE TABLE table-name AS query
proc sql;
  create table work.highcredit as
    select FirstName, LastName,
           UserID, CreditScore
      from sq.customer
     where CreditScore > 700;
quit;
```



work.highcredit

NOTE: Table WORK.HIGHCREDIT created, with 26006 rows and 4 columns.



To create a table from a query result, use a CREATE TABLE statement with the AS keyword, and place it before the SELECT statement. When a table is created this way, its data is derived from the table or view that is referenced in the query's FROM clause. The new table's column names are as specified in the query's SELECT clause list. The new table's column attributes (the type, length, informat, format, and extended attributes) are the same as the selected source columns.

You can create only one table in a PROC SQL step. Use the DATA step if you want to create multiple output tables in a single step:

```
DATA output-table1 output-table<n>;
  SET input-table;
  <additional SAS statements>;
RUN;
```

2.09 Activity

Open **s102a09.sas** from the **activities** folder and perform the following tasks:

1. Examine and run the query in the **Create a Table from a Query** section. View the results.
2. Add the CREATE TABLE statement and create a table named **Top5States**. Run the query and confirm that the table was created successfully.
3. Run the code below your SQL query. What did the code produce?

Copying the Structure of an Existing Table

```
proc sql;
create table work.higccredit
    like sq.customer(keep=FirstName LastName
                    UserID CreditScore);
quit;
```

**CREATE TABLE table-name
LIKE existing-table;**

Creates a copy of
the table
structure

NOTE: Table WORK.HIGHCCREDIT created, with 0 rows and 4 columns.

To create an empty table that has the same columns and attributes as an existing table or view, use the LIKE clause in the CREATE TABLE statement.

Creating a Table by Defining Columns

```
CREATE TABLE table-name
  (column-name type(length)
  <, ...column-name type(length)>);
```

```
proc sql;
create table work.employee
  (FirstName char(20),
  LastName char(20),
  DOB date format=mmddyy10.,
  EmpID num format=z6.);
quit;
```

Specify column
names and
attributes.

NOTE: Table WORK.EMPLOYEE created, with 0 rows and 4 columns.


Copyright © SAS Institute Inc. All rights reserved.

You can create a new table without rows by using the CREATE TABLE statement to define the columns and their attributes. You can specify a column's name, type, length, informat, format, and label. The table definition is enclosed in parentheses. Individual column definitions are separated by commas.

Creating Tables with PROC SQL: Summary

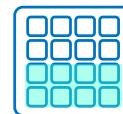
Description	Syntax
Creating a Table from a Query Result	CREATE TABLE table-name AS SELECT ...;
Copying the Structure from an Existing Table	CREATE TABLE table-name LIKE old-table-name;
Creating a Table by Defining Columns	CREATE TABLE table-name (column-name type(length) <, ...column-name type(length)>);



For more information about the CREATE TABLE statement, see “CREATE TABLE Statement” in the *SAS® 9.4 SQL Procedure User’s Guide, Fourth Edition* documentation. You can also find the direct link in the Course Links section on the ELP.

Inserting Rows into Tables

Use the **INSERT** statement to insert data values into tables.



79

Copyright © SAS Institute Inc. All rights reserved.

After tables are created, you can use the **INSERT** statement to insert data values into tables. You can use an empty table or a table that is already populated.

Inserting Rows with a Query

```
INSERT INTO table-name <(column list)>
SELECT columns
FROM table-name;
```

```
proc sql;
insert into work.highcredit
(FirstName, LastName, UserID, CreditScore)
select FirstName, LastName,
UserID, CreditScore
from sq.customer
where CreditScore > 700;
quit;
```

Columns from the query **must** be in the same position as in the **INSERT** column list.

80



Inserting Rows with the VALUES Clause

```
INSERT INTO table-name <(column list)>
VALUES (value,value,...);
```

```
proc sql;
insert into employee
  (FirstName, LastName, DOB, EmpID)
values ("Diego", "Lopez", "01SEP1980"d, 1280)
values ("Omar", "Fayed", "21MAR1989"d, 1310);
quit;
```

Data values align with column names
in the INSERT column list.

With the VALUES clause, you assign values to a column by position.

Note the following features of VALUES clauses:

- As with other SQL clauses, use commas to separate columns. In addition, you must use a semicolon after the last VALUES clause only.
- If you omit data for a column without indicating a missing value, then you receive an error message and the row is not inserted.
- To specify that a value is missing, use a space in single quotation marks for character values and a period for numeric values.

Inserting Rows with the SET Clause

```
INSERT INTO table-name
SET column-name=value,
    column-name=value,...;
```

```
proc sql;
insert into employee
  set FirstName= "Diego",
      LastName= "Lopez",
      DOB = "01SEP1980"d,
      EmpID = 1280;
quit;
```

Columns within the SET clause **must** exist in the table.

With the SET clause, you assign values to columns by name. The columns can appear in any order in the SET clause.

Note the following features of SET clauses:

- As with other SQL clauses, use commas to separate columns. In addition, you must use a semicolon after the last SET clause only.
- If you omit data for a column, then the value in that column is a missing value.
- To specify that a value is missing, use a blank in single quotation marks for character values and a period for numeric values.

Inserting Rows with PROC SQL: Summary

Description	Syntax
A query returning multiple rows based on positional values	INSERT INTO <i>table-name</i> <i><(column list)></i> SELECT <i>columns</i> FROM <i>table-name</i> ;
One clause per row using positional values	INSERT INTO <i>table-name</i> <i><(column list)></i> VALUES (<i>value1,value2,...</i>);
One clause per row using column-value pairs	INSERT INTO <i>table-name</i> SET <i>column-name=value,</i> <i>column-name=value,...</i> ;

For more information about the INSERT Statement, see “INSERT Statement” in the *SAS® 9.4 SQL Procedure User’s Guide, Fourth Edition* documentation. You can also find the direct link in the Course Links section on the ELP.

Dropping Tables in SQL

```
DROP TABLE table-name;
```

```
proc sql;
drop table work.employee;
quit;
```

NOTE: Table WORK.EMPLOYEE has been dropped.

This is useful if you are working with DBMSs that do not allow you to overwrite existing tables.



2.10 Activity

Open **s102a10.sas** from the **activities** folder and perform the following tasks to create a new table and insert rows into it:

1. Examine the CREATE TABLE statement and run the query only. Confirm that an empty table was created.
2. In the **Inserting Rows with a Query** section, enter the correct column names to complete the INSERT INTO statement. Run the query. How many rows were inserted into the table **highcredit**?
3. In the **Inserting Rows with the SET Clause** section, complete the INSERT INTO statement with the SET clause and insert yourself as a customer into the **highcredit** table. Run the query. What does the note in the log say?
4. Complete the code to drop the **highcredit** table.

85

Copyright © SAS Institute Inc. All rights reserved.



Additional Statements

Statement	Description
ALTER TABLE	Adds columns to, drops columns from, and changes column attributes in an existing table
UPDATE	Modifies a column's values in existing rows of a table or view
DELETE	Removes one or more rows from a table or view that is specified in the FROM clause

88

Copyright © SAS Institute Inc. All rights reserved.



For more information about the ALTER TABLE, UPDATE, or DELETE Statement, see “SQL Procedure Reference” in the *SAS® 9.4 SQL Procedure User’s Guide, Fourth Edition* documentation. You can also find the direct link in the Course Links section on the ELP.

A SAS method to alter table metadata is the DATASETS procedure. For more information about PROC DATASETS, see “DATASETS Procedure” in the *Base SAS® 9.4 Procedures Guide, Seventh Edition* documentation. You can also find the direct link in the Course Links section on the ELP.

Syntax Summary

DROP TABLE *table-name*;

Drop Tables



CREATE TABLE *table-name* **AS** *query*
CREATE TABLE *table-name* **LIKE** ...
CREATE TABLE *table-name* (...)

Create Tables

INSERT INTO *table-name* <(column list)> *query*
INSERT INTO *table-name* <(column list)> **VALUES** ...
INSERT INTO *table-name* **SET** ...

Insert Rows

2.4 Using DICTIONARY Tables

Business Scenario



What tables are defined?

Which tables contain a specific column?

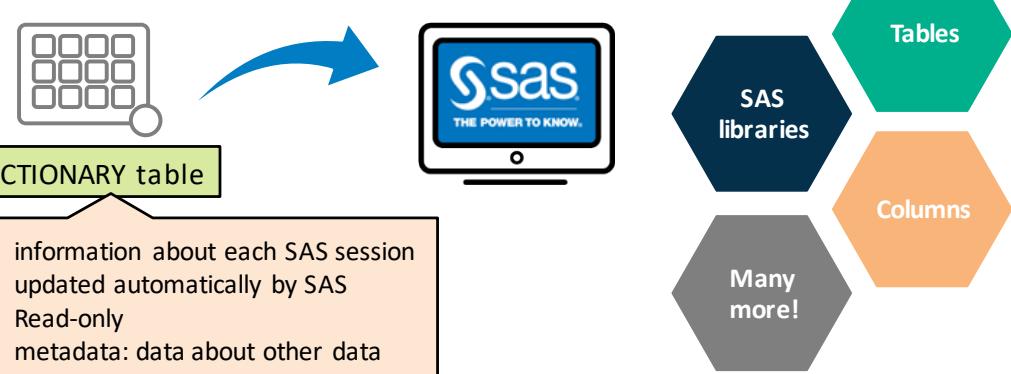
Are related columns the same type and length in all tables?

What libraries are assigned?

91

Sas

DICTIONARY Tables



DICTIONARY table

- information about each SAS session
- updated automatically by SAS
- Read-only
- metadata: data about other data
- valid in PROC SQL only

92

Copyright © SAS Institute Inc. All rights reserved.

Sas

DICTIONARY tables are special Read-only PROC SQL tables or views. They retrieve information about all the SAS libraries, SAS data sets, SAS system options, and external files that are associated with the current SAS session. For example, the **DICTIONARY.columns** table contains information such as name, type, length, and format, about all columns in all tables that are known to the current SAS session. PROC SQL automatically assigns the **DICTIONARY** libref. To get information from DICTIONARY tables, specify **DICTIONARY.table-name** in the FROM clause in a SELECT statement in PROC SQL.

For more information about DICTIONARY tables, see “Accessing SAS Information by Using DICTIONARY Tables” in the *SAS® 9.4 SQL Procedure User’s Guide, Fourth Edition* documentation. You can also find the direct link in the Course Links section on the ELP.

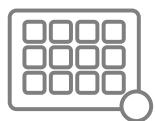
2.11 Activity

Open **s102a11.sas** from the **activities** folder and perform the following tasks to find all the available DICTIONARY tables in your SAS session:

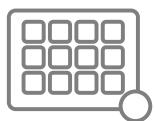
1. Examine and run the program. View the log and results.
2. Note the column labels for the first two columns: **Member Name** is the DICTIONARY table, and **Data Set Label** is the description of that table.
3. Replace the asterisk in the SELECT clause and select the **DISTINCT memname** and **memlabel** columns. Run the query and examine all the available DICTIONARY tables in your SAS session.
4. What is the data set label of the **members** DICTIONARY table?

DICTIONARY Tables

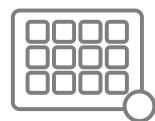
Explore
Data



DICTIONARY.tables



DICTIONARY.columns



DICTIONARY.libnames

96

Copyright © SAS Institute Inc. All rights reserved.

sas



Using DICTIONARY Tables

Scenario

Use SQL to query DICTIONARY information.

Files

- **s102d06.sas**
- **DICTIONARY.tables**, **DICTIONARY.columns**, **DICTIONARY.libnames**

Syntax

```
PROC SQL;
DESCRIBE TABLE DICTIONARY.<input-table>;
SELECT *
  FROM DICTIONARY.<input-table>
 WHERE expression
 ORDER BY col-name <DESC>;
QUIT;
```

Notes

- **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** provides detailed information about tables.
- **DICTIONARY.columns** provides detailed information about all columns in all tables.
- **DICTIONARY.libnames** provides detailed information about current SAS LIBNAME connections.

Demo

1. Open the **s102d06.sas** program in the **demos** folder and find the **Demo** section.
2. In the **Explore dictionary.tables** section:
 - a. Highlight and run the procedure. Examine the log and the results.
 - b. Add a WHERE clause to subset the **Libname** column for libraries named **SQ** and remove the INOBS= option. Highlight and run the procedure. Examine the log and the results.

Note: The **Libname** and **Memname** columns are stored in all uppercase.

```
proc sql inobs=100;
describe table dictionary.tables;
select *
  from dictionary.tables
 where Libname = 'SQ';
quit;
```

- c. Discuss the code for the SAS equivalent of **DICTIONARY.tables**. Highlight and run the procedure. Examine the log and the results.

```
proc print data=sashelp.vtable(obs=100);
  where Libname = 'SQ';
run;
```

3. Move to the **Explore dictionary.columns** section.
 - a. Highlight and run the procedure. Examine the log and the results.
 - b. Modify the WHERE clause to subset the **Name** column by **BankID** and use the **UPCASE** function on the **Name** column. Highlight and run the procedure. Examine the log and the results.

```
proc sql;
describe table dictionary.columns;
select *
  from dictionary.columns
  where Libname = 'SQ' and upcase(Name) = 'BANKID';
quit;
```

- c. Discuss the code for the SAS equivalent of **DICTIONARY.columns**. Highlight and run the procedure. Examine the log and the results.

```
proc print data=sashelp.vcolumn(obs=100);
  where Libname = 'SQ';
run;
```

4. Move to the **Explore dictionary.libnames** section.
 - a. Highlight and run the procedure. Examine the log and the results.
 - b. Modify the SELECT clause by replacing the asterisks and adding the DISTINCT keyword on the **Libname** column. Highlight and run the procedure. Examine the log and the results.

```
proc sql;
describe table dictionary.libnames;
select distinct libname
  from dictionary.libnames;
quit;
```

- c. Modify the SELECT clause by replacing **distinct libname** with an asterisk. Add a WHERE clause to subset the **Libname** column for the **SQ** library. Highlight and run the procedure. Examine the log and the results.

```
proc sql;
describe table dictionary.libnames;
select *
  from dictionary.libnames
  where Libname = 'SQ';
quit;
```

- d. Discuss the code for the SAS equivalent of **DICTIONARY.libnames**. Highlight and run the procedure. Examine the log and the results.

```
proc print data=sashelp.vlibnam;
  where Libname = 'SQ';
run;
```

End of Demonstration

Syntax Summary

`FROM DICTIONARY.table`

DICTIONARY
Library



`dictionary.dictionaries`

`dictionary.tables`

`dictionary.columns`

`dictionary.libnames`

DICTIONARY Tables



Beyond SQL Essentials

What if you want to ...

... learn more about
managing data
efficiently?

- Read the SAS paper [PROC DATASETS: Managing Data Efficiently](#).
- View the [SAS® 9.4 DATASETS Procedure Tip Sheet](#).

... learn more about
using DICTIONARY
tables?

- View the [SQL Dictionary Tables](#) section in the ELP for additional SAS papers.

... learn more about
data visualization
using PROC SGPLOT?

- Visit the SAS blog series [Getting started with SGPlot](#).
- Visit the SAS [Data Visualization](#) blogs.





Practice

Level 1

8. Counting the Number of Tables in a Library

- Write a query to create a report that displays the count of the number of tables in the **SQ** library.
 - Use **DICTIONARY.tables** as input.
 - Name the calculated column **TableCount**.
 - Add an appropriate title and run the program to review the results.

Results

Count of SQ Tables	
	TableCount
	27

- How many tables are in the **sq** library?

Level 2

9. Counting the Number of Tables in All Libraries

- Write a query to create a report that displays the count of the number of tables in all libraries.
 - Use **DICTIONARY.tables** as input.
 - Name the calculated column **TableCount**.
 - Group the results by the library name.
 - Add an appropriate title and display the library name and table count as shown below. (Your library list and counts might differ.)

Results

Count of All Tables	
Library Name	TableCount
MAPS	335
MAPSGFK	428
MAPSSAS	335
SASHHELP	222
SQ	27
WORK	1

- Which library has the most tables?

Challenge

10. Finding Tables with a Column Using Pattern Matching

- a. Write a query to create a report that displays the list of all tables in the **SQ** library with a column containing **ID** in its name.

Display the table name and the column name containing **ID** as shown below.

Partial Results

Member Name	Column Name
BANK	BankID
CUSTOMER	AccountID
CUSTOMER	BankID
CUSTOMER	CustomerID
CUSTOMER	StateID
CUSTOMER	UserID
EMPLOYEE	EmployeeID
EMPLOYEE	ManagerID
EMPLOYEE	StreetID
MERCHANT	MerchantID
SALESEMAIL	CustomerID

- b. How many unique tables have a column that contains **ID** in its name?

End of Practices

2.5 Solutions

Solutions to Practices

1. Querying a Table

```
/*s102s01.sas*/

title 'Large Non-Educational Transactions';
proc sql;
select CustomerName label='Customer Name',
   MerchantName, Type, Service,
   Amount format=dollar10.2 label='Transaction Amount'
from sq.transactionfull
where Amount >1000 and
      Service ne 'University'
order by Amount desc;
quit;
title;
```

How much did Pamela Maiden spend on Home Insurance? **\$1,816.57**

2. Using a Function in a Filter

```
/*s102s02.sas*/

title 'Customers with May Birthdays in North Carolina';
proc sql;
select FirstName, LastName, DOB format=date9.
   from sq.customer
  where month(DOB)= 5 and
        State='NC'
  order by DOB desc;
quit;
title;
```

What is Amy Murray's date of birth? **05MAY1995**

3. Working with Datetime Values

```
/*s102s03.sas*/

title 'November/December Transactions';
proc sql;
select CustomerName label='Customer Name',
   MerchantName label='Merchant Name',
   Amount format=dollar10.2 label='Transaction Amount',
   datepart(DateTime) as TransactionDate format=date9.
                                label='Transaction Date'
from sq.transactionfull
where Service ^= 'University' and
      month(calculated TransactionDate) in (11, 12)
  order by DateTime;
```

```
quit;
title;
```

What value of **Merchant Name** is on the first documented transaction in December?
Big Burgers, Inc.

4. Conditional Processing with a Dynamic Title

```
/*s102s04.sas*/
title "Created on %left(%qsysfunc(today(),weekdate.))";
proc sql;
select FirstName 'First Name' format=$1.,
       LastName 'Last Name',
       DOB 'Date of Birth' format=date9.,
       CreditScore 'Credit Score',
       case
         when DOB between '01jan1928'd and '31dec1945'd
              then 'Silent'
         when DOB between '01jan1946'd and '31dec1964'd
              then 'Boomer'
         when DOB between '01jan1965'd and '31dec1979'd
              then 'GenX'
         when DOB between '01jan1980'd and '31dec1996'd
              then 'Millennial'
         when DOB >= '01jan1997'd then 'Post-Millennial'
         else 'Unknown'
       end as Generation
from sq.customer
where CreditScore is not null and State='VT'
order by Generation, CreditScore desc;
quit;
title;
```

Is the date in your title today's date? **Yes**

5. Eliminating Duplicates

```
/*s102s05.sas*/
title 'Unique Country Codes';
proc sql;
select distinct CountryCode
  from sq.globalfull
 order by CountryCode;
quit;

title 'Count of Unique Country Codes';
proc sql;
select count(distinct CountryCode) as CountryCount
  from sq.globalfull;
quit;
title;
```

How many unique country codes are in the **sq.globalfull** table? **151**

6. Grouping and Summarizing Data

```
/*s102s06.sas*/
title 'Customers with High Percentage of Suspicious Transactions';
proc sql;
select CustomerID,
       count(*) as TotalTransactions,
       sum(Amount >= 500) as SuspiciousTransactions,
       calculated SuspiciousTransactions/calculated
              TotalTransactions as PCTSuspicious format=percent8.2
from sq.transactionfull
where Service ^= 'University'
group by CustomerID
having PCTSuspicious>.05
order by PCTSuspicious desc;
quit;
title;
```

Which customer ID had the highest percentage of suspicious transactions? **1973179983**

7. Grouping and Summarizing Data with Calculations

```
/*s102s07.sas*/
title 'Forecasted Regions who will Borrow for Health or Medical
      Purpose';
proc sql;
select Region,
       (mean(EstYear1PCT)/100)*sum(EstYear1Pop) as EstCount1
              format=commal6.,
       (mean(EstYear3PCT)/100)*sum(EstYear3Pop) as EstCount3
              format=commal6.,
       (calculated EstCount3 - calculated EstCount1)/calculated
              EstCount1 as PctIncrease format=percent7.2
from sq.globalfull
where IndicatorName = 'Borrowed for health or medical purposes
      (% age 15+)'
group by Region
order by PctIncrease desc;
quit;
title;
```

Which region had the largest percent decrease from year 1 to year 3? **East Asia & Pacific**

8. Counting the Number of Tables in a Library

```
/*s102s08.sas*/
title 'Count of SQ Tables';
proc sql;
select count(*) as TableCount
  from dictionary.tables
  where libname='SQ';
quit;
title;
```

How many tables are in the **sq** library? **27**

9. Counting the Number of Tables in All Libraries

```
/*s102s09.sas*/
title 'Count of All Tables';
proc sql;
select libname, count(*) as TableCount
  from dictionary.tables
 group by libname;
quit;
title;
```

Which library has the most tables? **MAPSGFK (Results might differ.)**

10. Finding Tables with a Column Using Pattern Matching

```
/*s102s10.sas*/
title 'SQ Tables containing ID columns';
proc sql;
select distinct MemName, Name
  from dictionary.columns
  where Libname = 'SQ' and Name contains 'ID';
quit;
title;
```

How many unique tables have a column that contains **ID** in its name? **13**

End of Solutions

Solutions to Activities and Questions

2.01 Activity – Correct Answer

1. Complete the WHERE clause to filter for customers in the state of VT and run the query.

```
where State = "VT"
```

2. How many customers are from either VT or SC? **792 customers**

```
where State = "VT" or State = "SC"
```

3. How many customers are from either VT, SC, or GA? **2,704 customers**

```
where State in ("VT", "SC", "GA")
```

11

Copyright © SAS Institute Inc. All rights reserved.



continued...

2.02 Activity – Correct Answer

1. What do you notice about the values in the **CreditScore** column? How many rows are in your report? **Missing values are included in the results, for a total of 2,197 rows.**

```
where CreditScore < 500
```

Row	First Name	Last Name	User ID	CreditScore
1	Sharon	Howell	shajuhowell1215@voidemail.com	
2	Richard	Morgan	ricjimorgan8923@n/a.com	
3	Irene	Sullivan	iretisullivan6719@fakeemail.com	
4	Gloria	Smith	gloansmith9626@fakeemail.com	
5	Gregory	Regal	greviregal5328@fakeemail.com	
6	Monica	Pennington	moncapennington8628@invalid.com	489
7	Michael	Mentkowski	micstmentkowski7612@fakeemail.com	

SAS treats missing values as **smaller than** nonmissing values.

14

Copyright © SAS Institute Inc. All rights reserved.



2.02 Activity – Correct Answer

2. How many rows are in your report? **Missing values are *not* included in the results.** The new report contains 196 rows.



```
where CreditScore < 500 and CreditScore is not null
```

```
where CreditScore < 500 and CreditScore is not missing
```

```
where CreditScore < 500 and CreditScore ne .
```

equivalent statements

2.03 Activity – Correct Answer

- What is the default sort order? **Ascending**
- What does the DESC option do? **Changes the sort order to descending**
- Who is the first customer on the report? **Donald Leyva**
- Are the results still sorted by LastName within CreditScore? **Yes, you can sort by any columns in the input table even if they are not in the SELECT clause.**

```
select FirstName, CreditScore
  from sq.customer
 where CreditScore > 830
 order by CreditScore desc, LastName;
```

continued...

2.04 Activity – Correct Answer

Customers from Hawaii					
First Name	Last Name	State	Email Address	Estimated Income	Date of Birth
Gloria	Tisor	HI	glopetisor6918@invalid.com	\$91,955.40	18JUN1969
Grace	Wright	HI	graelwright8418@notreal.com	\$91,379.77	18OCT1984
Roberto	Robison	HI	robfrrobison8024@invalid.com	\$90,920.97	24SEP1980
Laura	Dumoulin	HI	laukadumoulin5625@invalid.com	\$88,578.71	25DEC1956
Dan	Borgen	HI	dansaborgen9012@n/a.com	\$84,554.62	12MAR1990
Howard	Gonzales	HI	howangonzales9219@voidemail.com	\$83,240.89	19FEB1992
Laura	Gardner	HI	laurogardner9020@notreal.com	\$83,230.53	20APR1990
Sheila	Anson	HI	shedoanson7131@invalid.com	\$83,116.78	31DEC1971
Taylor	Lopez	HI	tayeulopez5923@n/a.com	\$80,413.02	23AUG1959
Patricia	Nickey	HI	patmanickey642@notreal.com	\$78,487.53	02JUL1964

March 6

28
Copyright © SAS Institute Inc. All rights reserved.

To dynamically create a footnote or title using today's date, use this statement:

FOOTNOTE "%LEFT(%QSYSFUNC(TODAY(),add_date_format))";

2.04 Activity – Correct Answer

4. Change the DOLLAR16.2 format to DOLLAR7.2. What happens to the values in the **Income** column? **The values are missing commas and dollar signs, and they contain only one decimal point.**

Customers from Hawaii					
First Name	Last Name	State	Email Address	Estimated Income	Date of Birth
Gloria	Tisor	HI	glopetisor6918@invalid.com	91955.4	18JUN1969
Grace	Wright	HI	graelwright8418@notreal.com	91379.8	18OCT1984
Roberto	Robison	HI	robfrrobison8024@invalid.com	90921.0	24SEP1980
Laura	Dumoulin	HI	laukadumoulin5625@invalid.com	88578.7	25DEC1956

When the format width is not large enough, SAS preserves the number.

29
Copyright © SAS Institute Inc. All rights reserved.

2.05 Activity – Correct Answer

2. What is the name of the new column? **Without the AS keyword, the new column does not have a name.**
3. What changes? **The AS keyword specifies a column name for the new column.**
4. Did the query run successfully? **No. The WHERE clause is evaluated before the SELECT clause. Therefore, columns used in the WHERE clause must exist in the table listed in the FROM clause.**

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

2.06 Activity – Correct Answer

2. What does this query show? **It displays the unique values of N or Y in the Employed column.**

```
select distinct Employed
```

Employed
N
Y

3. What does this query show? **It displays unique combinations of Employed and Married values.**

```
select distinct Married, Employed
```

Employed	Married
N	
N	D
N	M
N	S
N	W
Y	
Y	D

2.07 Activity – Correct Answer

1. Why is the value of **MaritalStatus** different from the value of **TotalRows**?

TotalRows counts all rows in the table. **MaritalStatus** counts every nonmissing occurrence in the **Married** column.

TotalRows	Marital Status
100,004	92,850

2. What does the new report show? You can use **DISTINCT** with the **COUNT** function to return the number of distinct, nonmissing values from a column.

```
select count(*) as TotalRows format=comma10.,
       count(distinct Married) as MaritalStatus format=comma10.
  from sq.customer;
```

TotalRows	Marital Status
100,004	4

55



Copyright © SAS Institute Inc. All rights reserved.

2.08 Activity – Correct Answer

1. Which month has the highest value for **MedianSpent**? **Month 2, February**
2. What is the error? **The following columns were not found in the contributing tables: Month.**

```
select qtr(datepart(DateTime)) as Qtr,
       Median(Amount) as MedianSpent format=dollar16.
  from sq.transaction
 group by Month;
```

Replace Month
with Qtr.

3. Which quarter has the highest value for **MedianSpent**? **Quarter 1**

Qtr	MedianSpent
1	\$42
2	\$28
3	\$27
4	\$26

63



Copyright © SAS Institute Inc. All rights reserved.

continued...

2.09 Activity – Correct Answer

1. Examine and run the query. View the results.

State Name	Population Estimate
CA	39,209,127
TX	27,937,492
FL	20,629,982
NY	19,641,589
IL	12,826,895

2. Add the CREATE TABLE statement and create a table named **Top5States**. Run the query and confirm that the table was created successfully.

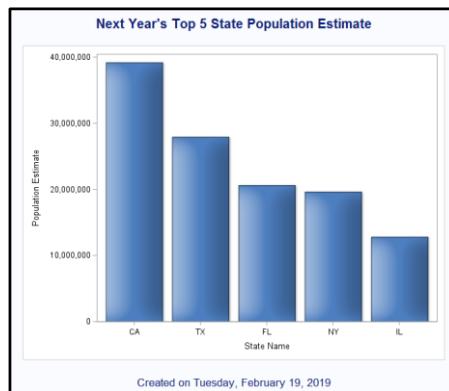
WARNING: Statement terminated early due to OUTOBS=5 option.

NOTE: Table WORK.TOP5STATES created, with 5 rows and 2 columns.

2.09 Activity – Correct Answer

3. Run the code below your SQL query. What did the code produce?

The SGLOT procedure created a visualization of the newly created table.



continued...

2.10 Activity – Correct Answer

1. There are four columns and zero rows in the new highcredit table.

NOTE: Table WORK.HIGHCREDIT created, with 0 rows and 4 columns.

2. How many rows were inserted into the table **highcredit?** 26,006 rows were inserted into work.highcredit.

```
insert into work.highcredit(FirstName,LastName,
                           UserID,CreditScore)
  select FirstName, LastName,
         UserID, CreditScore
    from sq.customer
   where CreditScore > 700;
```

86

Copyright © SAS Institute Inc. All rights reserved.



2.10 Activity – Correct Answer

3. What does the note in the log say?

NOTE: 1 row was inserted into WORK.HIGHCREDIT.

```
proc sql;
  insert into highcredit
    set FirstName="Panagiotis",
        LastName="Styliadis",
        UserID="pststyliadis",
        CreditScore=800;
quit;
```

- 4.

```
proc sql;
  drop table highcredit;
quit;
```

87

Copyright © SAS Institute Inc. All rights reserved.



continued...

2.11 Activity – Correct Answer

1.

Member Name	Data Set Label	Column Name	Column Type	Column Length	Column Position
MEMBERS	Tables, catalogs, and views	LIBNAME	char	8	0
MEMBERS	Tables, catalogs, and views	MEMNAME	char	32	8
MEMBERS	Tables, catalogs, and views	MEMTYPE	char	8	40
MEMBERS	Tables, catalogs, and views	DBMS_MEMTYPE	char	32	48
MEMBERS	Tables, catalogs, and views	ENGINE	char	8	56
MEMBERS	Tables, catalogs, and views	INDEX	char	3	64
MEMBERS	Tables, catalogs, and views	PATH	char	1024	91
TABLES	Tables and table-specific information	LIBNAME	char	8	0
TABLES	Tables and table-specific information	MEMNAME	char	32	8

2.11 Activity – Correct Answer

3. Run the query and examine all the available DICTIONARY tables in your SAS session.

Member Name	Data Set Label
CATALOGS	Catalogs and catalog-specific information
CHECK_CONSTRAINTS	Check constraints
COLUMNS	Columns from every table
CONSTRAINT_COLUMN_USAGE	Constraint column usage
CONSTRAINT_TABLE_USAGE	Constraint table usage
DATAITEMS	Information for Data Items

4. What is the data set label of the **members** DICTIONARY table?

Tables, catalogs and Views

Lesson 3 SQL Joins

3.1	Introduction to SQL Joins.....	3-3
3.2	Inner Joins.....	3-8
	Demonstration: Performing an Inner Join with PROC SQL.....	3-10
	Demonstration: Performing an Inner Join with Four Tables.....	3-16
	Practice.....	3-23
3.3	Outer Joins.....	3-28
	Demonstration: Performing a Full Join with PROC SQL.....	3-35
	Practice.....	3-40
3.4	Complex Joins.....	3-45
	Demonstration: Performing a Reflexive Join	3-47
3.5	Solutions	3-54
	Solutions to Practices	3-54
	Solutions to Activities and Questions.....	3-60

3.1 Introduction to SQL Joins

Joining Tables

smallcustomer Partial

FirstName	LastName	...	AccountID
Gary	Sienkiewicz	...	1010159565
Sergio	Lefeld	...	1010367330
John	Oliver	...	2020012887
Iva	Bower	...	3030085224

smalltransaction Partial

AccountID	DateTime	BankID	...
.	07MAY18:15:35:02
1010159565	16SEP18:14:57:08	101010101	...
1010183063	24FEB18:17:27:42	101010101	...
1010367330	15MAY18:17:54:21	101010101	...
1010367330	17OCT18:11:02:38	101010101	...

3



Copyright © SAS Institute Inc. All rights reserved.

Joining Tables

smallcustomer Partial

FirstName	LastName	...	AccountID
Gary	Sienkiewicz	...	1010159565
Sergio	Lefeld	...	1010367330
John	Oliver	...	2020012887
Iva	Bower	...	3030085224

smalltransaction Partial

AccountID	DateTime	BankID	...
.	07MAY18:15:35:02
1010159565	16SEP18:14:57:08	101010101	...
1010183063	24FEB18:17:27:42	101010101	...
1010367330	15MAY18:17:54:21	101010101	...
1010367330	17OCT18:11:02:38	101010101	...

Primary key

Foreign key

4



Copyright © SAS Institute Inc. All rights reserved.

A *primary key* is a column that contains a unique value for each row of data. A primary key cannot contain missing values.

A *foreign key* is a column in one table that refers to the primary key in another table.

Joining Tables

smallcustomer Partial

FirstName	LastName	...	AccountID
Gary	Sienkiewicz	...	1010159565
Sergio	Lefeld	...	1010367330
John	Oliver	...	2020012887
Iva	Bower	...	3030085224

smalltransaction Partial

AccountID	DateTime	BankID	...
.	07MAY18:15:35:02
1010159565	16SEP18:14:57:08	101010101	...
1010183063	24FEB18:17:27:42	101010101	...
1010367330	15MAY18:17:54:21	101010101	...
1010367330	17OCT18:11:02:38	101010101	...

FirstName	LastName	...	AccountID	DateTime	BankID	...
Gary	Sienkiewicz	...	1010159565	16SEP18:14:57:08	101010101	...
Sergio	Lefeld	...	1010367330	15MAY18:17:54:21	101010101	...
Sergio	Lefeld	...	1010367330	17OCT18:11:02:38	101010101	...

5

Copyright © SAS Institute Inc. All rights reserved.



Joins combine data horizontally from multiple source tables to produce either a report or an output table. The source tables are left intact and untouched.

SQL Join Syntax

**SELECT col-name, col-name
FROM table1, table2**

```
proc sql;
  select *
    from sq.smallcustomer, sq.smalltransaction;
  quit;
```

8 rows

12 rows

List the table names in the **FROM** clause.

6

Copyright © SAS Institute Inc. All rights reserved.



Default Join

smallcustomer Partial				smalltransaction Partial			
FirstName	LastName	...	AccountID	AccountID	DateTime	BankID	...
Gary	Sienkiewicz	...	1010159565	.	07MAY18:15:35:02
Sergio	Lefeld	...	1010367330	1010159565	16SEP18:14:57:08	101010101	...
John	Oliver	...	2020012887	1010183063	24FEB18:17:27:42	101010101	...
Iva	Bower	...	3030085224	1010367330	15MAY18:17:54:21	101010101	...
				1010367330	17OCT18:11:02:38	101010101	...

By default, SQL will join every row in the **smallcustomer** table with every row in the **smalltransaction**

8 rows x 12 rows = 96 rows

7
Copyright © SAS Institute Inc. All rights reserved.



When you run this query, the Cartesian product creates a report with a total of 96 rows. A Cartesian product is rarely the result that you want when you join tables. When working with large tables, a Cartesian product can create an unnecessarily large report or table, and even slow down system resources.

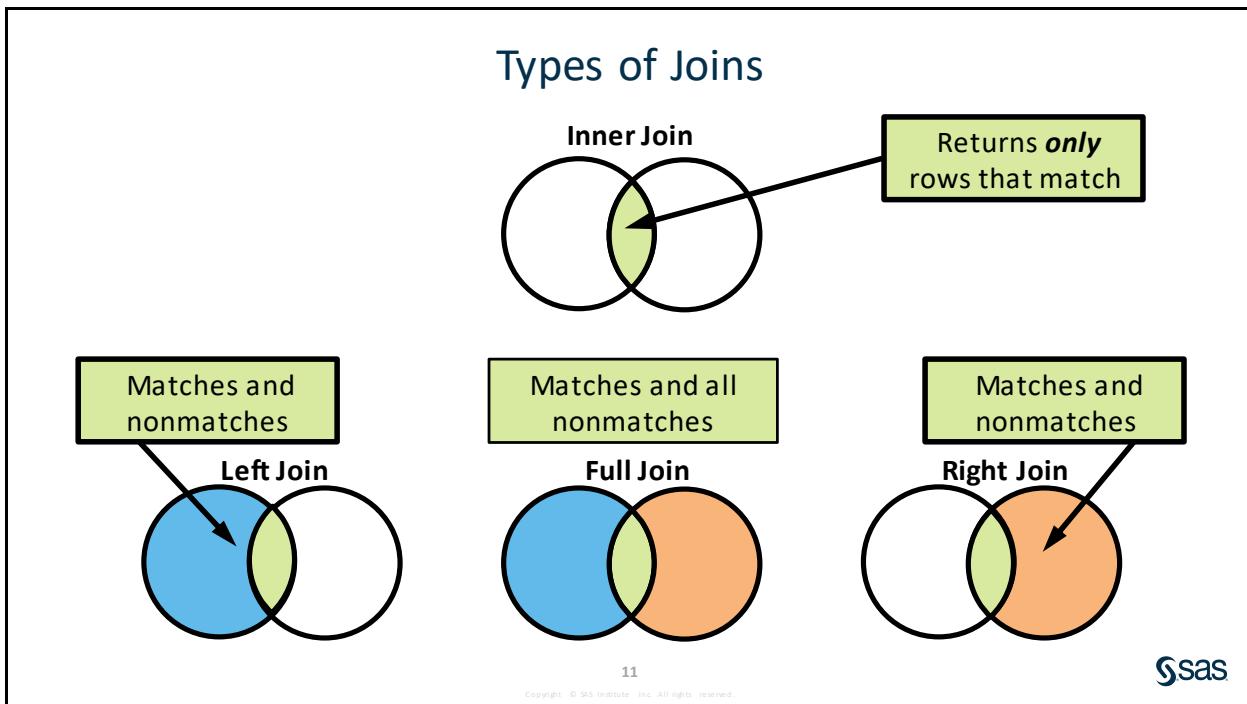
3.01 Activity

Open **s103a01.sas** from the **activities** folder and perform the following tasks to perform a default join of two tables:

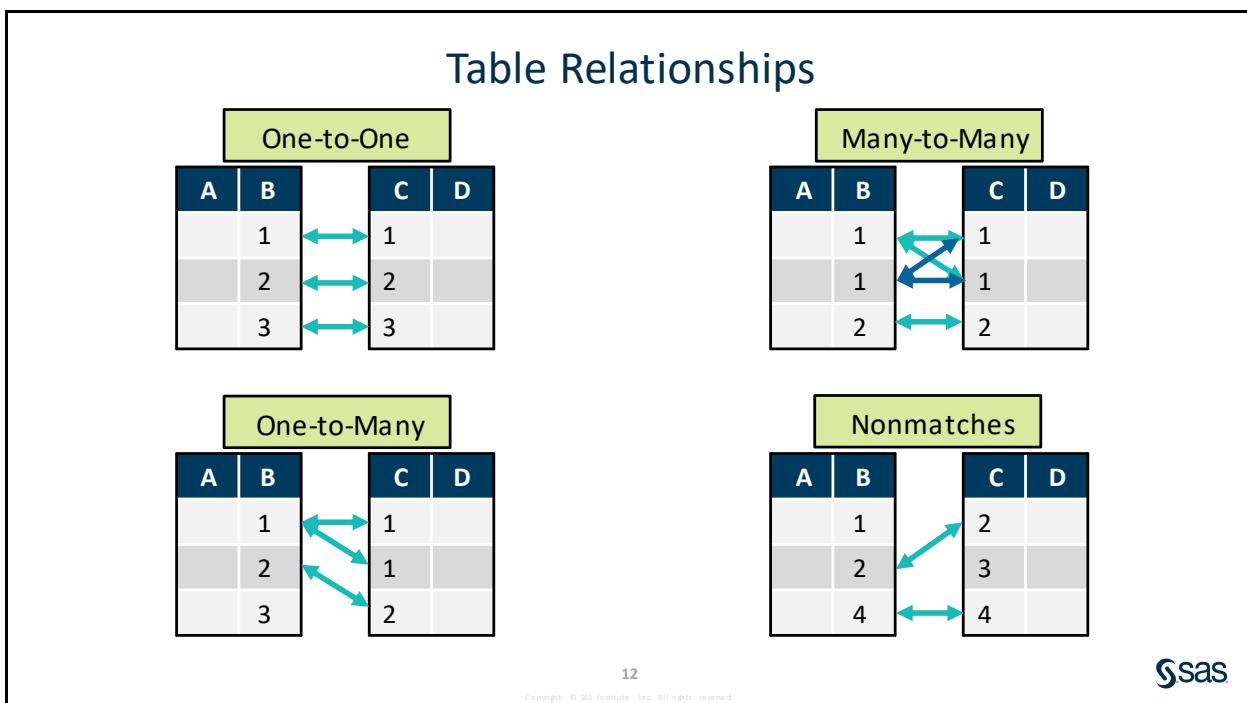
1. Examine and run the two queries to explore the **sq.smallcustomer** and **sq.smalltransaction** tables. Confirm that the **sq.smallcustomer** contains 8 rows and the **sq.smalltransaction** contains 12 rows.
2. In the next section, list the **sq.smallcustomer** and **sq.smalltransaction** table in the **FROM** clause and separate the tables by a comma. Run the query and view the log. What note do you see?
3. View the results. Name two issues with the report.

8
Copyright © SAS Institute Inc. All rights reserved.





Joining tables is a very common requirement when working with data. There are multiple methods available in SAS to join tables. The most common are SQL and the DATA step. This course focuses on SQL joins. The SAS® Programming 2 course addresses the DATA step merge.



One-to-One: A single observation in one data set is related to exactly one observation in another data set based on the values of one or more selected variables.

One-to-many: A single observation in one data set is related to more than one observation in another data set based on the values of one or more selected variables.

Many-to-Many: Occurs when multiple records in a table are associated with multiple records in another table.

Nonmatches: At least one observation in one data set is unrelated to any observation in another data set based on the values of one or more selected variables.

3.2 Inner Joins

Scenario



Report showing *only* customers with their
matching transaction



14

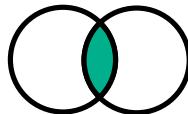
Copyright © SAS Institute Inc. All rights reserved.

SQL Inner Join Syntax

```
SELECT col-name, col-name
FROM table1 INNER JOIN table2
ON table1.column = table2.column;
```

```
proc sql;
select FirstName, LastName, State, Income, DateTime, Amount
from sq.smallcustomer inner join sq.smalltransaction
on smallcustomer.AccountID = smalltransaction.AccountID;
quit;
```

Specify the join type
in the FROM clause.



15

s103d01

Copyright © SAS Institute Inc. All rights reserved.

SQL Inner Join Syntax

```
SELECT col-name, col-name
FROM table1 INNER JOIN table2
ON table1.column = table2.column;
```

```
proc sql;
select FirstName, LastName, State, Income, DateTime, Amount
  from sq.smallcustomer inner join sq.smalltransaction
    on smallcustomer.AccountID = smalltransaction.AccountID;
quit;
```

Specify the join criteria in the ON clause. You can use other comparison operators, such as the greater than, less than, or special where operators.

16

Copyright © SAS Institute Inc. All rights reserved.

s103d01



SQL Inner Join Syntax

```
SELECT col-name, col-name
FROM table1 INNER JOIN table2
ON table1.column = table2.column;
```

```
proc sql;
select FirstName, LastName, State, Income, DateTime, Amount
  from sq.smallcustomer inner join sq.smalltransaction
    on smallcustomer.AccountID = smalltransaction.AccountID;
quit;
```

Qualify the column names to specify the location of each column.

17

Copyright © SAS Institute Inc. All rights reserved.

s103d01





Performing an Inner Join with PROC SQL

Scenario

Use PROC SQL to perform an inner join between two tables.

Files

- **s103d01.sas**
- **smallcustomer** – a SAS table that contains one row per customer
- **smalltransaction** – a SAS table that contains one row per customer transaction

Syntax

```
PROC SQL;
SELECT col-name, col-name
    FROM table1 INNER JOIN table2
        ON table1.col-name=table2.col-name;
QUIT;
```

Notes

- An SQL inner join combines matching rows between two tables.
- The two tables to be joined are listed in the FROM clause separated by INNER JOIN.
- The ON expression indicates how rows should be matched.

Demo

1. Open the **s103d01.sas** program in the **demos** folder and find the **Demo** section of the program. Run the queries in the **Explore the Tables** section to compare the columns of the **sq.smallcustomer** and **sq.smalltransaction** tables.

```
proc sql;
select *
  from sq.smallcustomer;
select *
  from sq.smalltransaction;
quit;
```

2. Find the **Perform the INNER JOIN** section and add **sq.smallcustomer** and **sq.smalltransaction** to the FROM clause to perform an inner join on **AccountID**. Qualify **AccountID** columns as *table-name.col-name* in the ON expression only. Highlight and run the query.

```
proc sql;
select FirstName, LastName, State, Income, DateTime, MerchantID,
      Amount
  from sq.smallcustomer inner join
```

```

sq.smalltransaction
on smallcustomer.AccountID = smalltransaction.AccountID;
quit;

```

3. Add the **AccountID** column to the query after **Amount**. Highlight and run the query. Examine the log. Why does the program fail?

Note: There is an ambiguous reference. The column **AccountID** is in more than one table.

```

proc sql;
select FirstName, LastName, State, Income, DateTime, MerchantID,
      Amount, AccountID
  from sq.smallcustomer inner join
       sq.smalltransaction
    on smallcustomer.AccountID = smalltransaction.AccountID;
quit;

```

4. Modify the query to qualify the **AccountID** column in the SELECT clause. Highlight the step and run the selected code.

Note: Because **AccountID** occurs in both tables, you must qualify the column with the table name to indicate which column you want to select.

```

proc sql;
select FirstName, LastName, State, Income, DateTime, MerchantID,
      Amount, smallcustomer.AccountID
  from sq.smallcustomer inner join
       sq.smalltransaction
    on smallcustomer.AccountID = smalltransaction.AccountID;
quit;

```

5. Modify the query to include a WHERE clause to subset for customers who have a **State** value of NY (New York) and an ORDER BY clause that sorts by descending **Amount**.

```

proc sql;
select FirstName, LastName, State, Income, DateTime, MerchantID,
      Amount, smallcustomer.AccountID
  from sq.smallcustomer inner join
       sq.smalltransaction
    on smallcustomer.AccountID = smalltransaction.AccountID
  where State = "NY"
  order by Amount desc;
quit;

```

FirstName	LastName	State	Income	DateTime	MerchantID	Amount	AccountID
Olga	Comstock	NY	31896.96	11MAR18:10:07:14	580881	319.95	3030165207
Gary	Sienkiewicz	NY	67210.91	16SEP18:14:57:08	568268	107.16	1010159565
Janet	Sienkiewicz	NY	50111.59	18SEP18:12:13:40	549940	37.38	3030101942
Iva	Bower	NY	67949.96	27JUL18:12:05:48	525576	26.1	3030085224

End of Demonstration

Alternative SQL Inner Join Syntax

```
SELECT col-name, col-name
  FROM table1, table2
 WHERE table1.column = table2.column;
```

```
proc sql;
select FirstName, LastName, State, Income, DateTime, Amount
  from sq.smallcustomer, sq.smalltransaction
  where smallcustomer.AccountID = smalltransaction.AccountID;
quit;
```

List the tables in the FROM clause.

List the join criteria in the WHERE clause.

Using Table Aliases

```
FROM table1 <AS> alias1, table2 <AS> alias2
```

```
proc sql;
select FirstName, LastName, State, Income, DateTime, c.AccountID
  from sq.smallcustomer as c inner join
       sq.smalltransaction as t
  on c.AccountID = t.AccountID;
quit;
```

Assign an alias (or nickname) to a table in the FROM clause by adding the keyword AS.

A table alias is a temporary alternate name for a table. You specify table aliases in the FROM clause.

3.02 Activity

Open **s103a02.sas** from the **activities** folder and perform the following tasks to perform an inner join on tables with different column names:

1. Examine and run the two queries to explore the **sq.statepopulation** and **sq.statecode** tables. What columns can you use to join the tables?
2. Specify the tables in the FROM clause and perform an inner join. Add the alias **p** for the **sq.statepopulation** table, and the alias **s** for the **sq.statecode** table.
3. Complete the ON expression to match rows where **p.Name = s.StateCode**. Highlight and run the query. How many rows are in the new report?

21



Matching Rows with a Natural Join

```
proc sql;
  select *
    from sq.smallcustomer as c natural join
         sq.smalltransaction as t;
quit;
```

**SELECT col-name, col-name
FROM table1 NATURAL JOIN table2**

A **natural join** assumes that you want to base the join on all pairs of **common columns**.



24



A *natural join* selects rows from two tables that have equal values in columns that share the same name and the same type. A natural join is requested with the syntax **NATURAL JOIN**. If like columns are not found, then a Cartesian product is performed.

Do not use an **ON** clause with a natural join. When using a natural join, an **ON** clause is implied, matching all like columns. You can use a **WHERE** clause to subset the query results.

A natural join can be an inner join or an outer join, which is requested with the syntax INNER or OUTER. If the join type specification is omitted, then an inner join is implied.

FEEDBACK Option

PROC SQL FEEDBACK;

```
proc sql feedback;
select *
  from sq.smallcustomer as c natural join
       sq.smalltransaction as t;
quit;
```

The FEEDBACK option expands a SELECT statement to the SAS log.

NOTE: Statement transforms to:

```
select COALESCE(T.AccountID, C.AccountID) as AccountID,
COALESCE(T.BankID, C.BankID) as BankID, T.DateTime...
```

25
Copyright © SAS Institute Inc. All rights reserved.



Selecting Data from More Than Two Tables

Results

FirstName	LastName	State	Income	Datetime	MerchantID	Amount	AccountID	BankID
Gary	Sienkiewicz	NY	67210.91	16SEP18:14:57:08	568268	107.16	1010159565	101010101
Sergio	Lefeld	CA	86859.07	15MAY18:17:54:21	542058	23.39	1010367330	101010101
Sergio	Lefeld	CA	86859.07	17OCT18:11:02:38	525576	21.02	1010367330	101010101
John	Oliver	CA	43623.75	23FEB18:09:25:37	525576	108.22	2020012887	202020202
Iva	Bower	NY	67949.96	27JUL18:12:05:48	525576	26.1	3030085224	303030303
Jane	Sienkiewicz	NY	50111.59	18SEP18:12:13:40	549940	37.38	3030101942	303030303
		NY	31896.96	11MAR18:10:07:14	580881	319.95	3030165207	303030303

How can I retrieve the **merchant name** and the **bank name** in the results?



Merchant Name ← Bank Name ←

26
Copyright © SAS Institute Inc. All rights reserved.



3.03 Activity

Open **s103a03.sas** from the **activities** folder and perform the following tasks to find tables that contain **BankID** and **MerchantID** columns:

1. Complete the first query by adding the **BANKID** column name in the WHERE clause. How many tables contain the **BankID** column?
2. Replace **BANKID** with **MERCHANTID**. How many tables contain the **MerchantID** column?



Performing an Inner Join with Four Tables

Scenario

Use PROC SQL to perform an inner join among four tables.

Files

- **s103d02.sas**
- **smallcustomer** – a SAS table that contains one row per customer
- **smalltransaction** – a SAS table that contains one row per customer transaction
- **bank** – a SAS table that contains bank information
- **merchant** – a SAS table that contains merchant information

Syntax

```
PROC SQL;
SELECT col-name, col-name
    FROM table1 INNER JOIN table2
        ON table1.col-name=table2.col-name INNER JOIN
            table3
        ON join-criteria INNER JOIN
            table4
        ON join-criteria;
QUIT;
```

Notes

- To join more than two tables, each join must be listed individually in the query.

Demo

1. Open the **s103d02.sas** program in the **demos** folder and find the **Demo** section of the program. Under the **Explore the Tables** section, run the queries to explore the **sq.smallcustomer**, **sq.smalltransaction**, **sq.bank**, and **sq.merchant** tables. Describe the relationships between the tables.

```
proc sql inobs=5;
title "Table: SMALLCUSTOMER";
select *
from sq.smallcustomer;
title "Table: SMALLTRANSACTION" ;
select *
from sq.smalltransaction;
title "Table: MERCHANT" ;
```

```

select *
  from sq.merchant;
title "Table: BANK";
select *
  from sq.bank;
title;
quit;

```

2. Find the **Joining Data from More Than Two Tables** section. Highlight and run the query to join **sq.smallcustomer** with **sq.transaction**. Examine the results.

```

proc sql;
select FirstName, LastName, c.State, Income, DateTime, MerchantID,
      Amount, c.AccountID, c.BankID
  from sq.smallcustomer as c inner join
       sq.smalltransaction as t
      on c.AccountID = t.AccountID;
quit;

```

3. Add a second inner join and join the **MerchantID** column from the **sq.merchant** table with the **MerchantID** column of the previous join. Replace **MerchantID** in the SELECT clause with **MerchantName**. Highlight and run the query. Examine the results.

```

proc sql;
select FirstName, LastName, c.State, Income, DateTime, MerchantID,
      MerchantName, Amount, c.AccountID, c.BankID
  from sq.smallcustomer as c inner join sq.smalltransaction as t
  on c.AccountID = t.AccountID inner join
       sq.merchant as m
  on t.MerchantID = m.MerchantID;
quit;

```

4. Add a third inner join and join the **BankID** column from the **sq.bank** table with the **BankID** column of the previous join. Replace **BankID** in the SELECT clause with the bank name. Highlight and run the query. Examine the results.

```

proc sql;
select FirstName, LastName, c.State, Income, DateTime,
      MerchantName, Amount, c.AccountID, e.BankID, b.Name
  from sq.smallcustomer as c inner join
       sq.smalltransaction as t
  on c.AccountID = t.AccountID inner join
       sq.merchant as m
  on t.MerchantID = m.MerchantID inner join
       sq.bank as b
  on t.BankID = b.BankID;
quit;

```

FirstName	LastName	State	Income	DateTime	Merchant Name	Amount	AccountID	Name
Sergio	Lefeld	CA	86859.07	17OCT18:11:02:38	Economical Superstore	21.02	1010367330	Biggest Bank, Inc.
Iva	Bower	NY	67949.96	27JUL18:12:05:48	Economical Superstore	26.1	3030085224	Wheatberry Bank, Inc.
John	Oliver	CA	43623.75	23FEB18:09:25:37	Economical Superstore	108.22	2020012887	Sailors Credit Union
Sergio	Lefeld	CA	86859.07	15MAY18:17:54:21	Happy Sour Bar & Grill	23.39	1010367330	Biggest Bank, Inc.
Janet	Sienkiewicz	NY	50111.59	18SEP18:12:13:40	Big Burgers, Inc.	37.38	3030101942	Wheatberry Bank, Inc.
Gary	Sienkiewicz	NY	67210.91	16SEP18:14:57:08	Livable Landscaping, LLC	107.16	1010159565	Biggest Bank, Inc.
Olga	Comstock	NY	31896.96	11MAR18:10:07:14	Pebble Cable, Inc.	319.95	3030165207	Wheatberry Bank, Inc.

End of Demonstration

Effects of Missing Values on Joins

smallcustomer2 Partial

First Name	Last Name	State	Bank ID	Income	Account ID
Samantha	Carney	CA	-	25476.14	-
Alejandro	Garcia	NC	-	86324.38	-
Sai	Nair	NC	-	51256.02	-
Gary	Sienkiewicz	NY	101010101	67210.91	101015956
Sergio	Lefeld	CA	101010101	86859.07	1010367330
John	Oliver	CA	202020202	43623.75	2020012887
Iva	Bower	NY	303030303	67949.96	3030085224

smalltransaction2 Partial

#	AccountID	DateTime	BankID	MerchantID	Amount	Services
		07MAY18:15:3...	.	542058	58.79	Bar
		09MAY20:12:3...	.	549940	86.36	Fast Food
		16SEP18:14:5...	101010101	568268	107.16	Lawn Care
1010183063	24FEB18:17:27...		101010101	562326	370.53	Fancy Restaurant
1010367330	15MAY18:17:5...		101010101	542058	23.39	Bar
1010367330	17OCT18:11:0...		101010101	525576	21.02	Economy
1010367364	18OCT18:17:5...		101010101	549940	37.24	Fast Food
2020012887	23FEB18:09:25...		202020202	525576	108.22	Economy

PROC SQL treats
missing values as
matches for joins.



Most database products treat missing as the absence of a value (nulls). Because they do not contain any value, they are excluded from any conditional evaluation.

PROC SQL treats missing as missing values and matches for joins. Any missing value will match with any other missing value of the same type (character or numeric) in a join. This could return unexpected results.

Effects of Missing Values on Joins

FirstName	LastName	State	BankID	Income	AccountID	AccountID	DateTime	BankID	MerchantID	Amount	Services
Samantha	Carney	CA		25476.14			07MAY18:15:35:02		542058	58.79	Bar
Sai	Nair	NC		51256.02			07MAY18:15:35:02		542058	58.79	Bar
Alejandro	Garcia	NC		86324.38			07MAY18:15:35:02		542058	58.79	Bar
Samantha	Carney	CA		25476.14			09MAY20:12:30:08		549940	86.36	Fast Food
Sai	Nair	NC		51256.02			09MAY20:12:30:08		549940	86.36	Fast Food
Alejandro	Garcia	NC		86324.38			09MAY20:12:30:08		549940	86.36	Fast Food
Samantha	Carney	CA		25476.14			16SEP18:14:57:08	101010101	568268	107.16	Lawn Care
Sai	Nair	NC		51256.02			16SEP18:14:57:08	101010101	568268	107.16	Lawn Care
Alejandro	Garcia	NC		86324.38			16SEP18:14:57:08	101010101	568268	107.16	Lawn Care
Sergio	Lefeld	CA	101010101	86859.07	1010367330	1010367330	15MAY18:17:54:21	101010101	542058	23.39	Bar
Sergio	Lefeld	CA	101010101	86859.07	1010367330	1010367330	17OCT18:11:02:38	101010101	525576	21.02	Economy

Missing values are joined, and this typically is not the desired result.

Effects of Missing Values on Joins

```
proc sql;
  select *
  from sq.smallcustomer2 as c inner join
       sq.smalltransaction2 as t
  on c.AccountID = t.AccountID and
     c.AccountID is not null;
quit;
```

FirstName	LastName	State	BankID	Income	AccountID	AccountID
Sergio	Lefeld	CA	101010101	86859.07	1010367330	1010367330
Sergio	Lefeld	CA	101010101	86859.07	1010367330	1010367330
John	Oliver	CA	202020202	43623.75	2020012887	2020012887
Iva	Bower	NY	303030303	67949.96	3030085224	3030085224
Janet	Sienkiewicz	NY	303030303	50111.59	3030101942	3030101942
Olga	Comstock	NY	303030303	31896.96	3030165207	3030165207

Adding the IS NOT NULL operator to the ON clause prevents the missing values from joining.

Copyright © SAS Institute Inc. All rights reserved.



Non-Equijoin

smallcustomer

FirstName	LastName	State	BankID	Income	AccountID
Gary	Sienkiewicz	NY	101010101	67210.91	1010159565
Sergio	Lefeld	CA	101010101	86859.07	1010367330
John	Oliver	CA	202020202	43623.75	2020012887
Iva	Bower	NY	303030303	67949.96	3030085224
Janet	Sienkiewicz	NY	303030303	50111.59	3030101942
Olga	Comstock	NY	303030303	31896.96	3030165207

What if you don't want to join by **equality**?

taxbracket

TaxBracket	LowIncome	HighIncome
10%	0	9524.99
12%	9525	38699.99
22%	38700	82499.99
24%	82500	157499.99
32%	157500	199999.99
35%	200000	499999.99

33

Copyright © SAS Institute Inc. All rights reserved.



Non-Equijoin

```
select FirstName, LastName, Income,
       TaxBracket
  from sq.smallcustomer as c inner join
       sq.taxbracket as t
 on c.Income >= t.LowIncome and
    c.Income <= t.HighIncome;
```

Use comparison operators in the
ON clause instead of equality.

FirstName	LastName	Income	TaxBracket
Olga	Comstock	31896.96	12%
Ada	Vieyra	29586.44	12%
Samantha	Carmey	25476.14	12%
Gary	Sienkiewicz	67210.91	22%
John	Oliver	43623.75	22%
Iva	Bower	67949.96	22%
Janet	Sienkiewicz	50111.59	22%
Sergio	Lefeld	86859.07	24%

34


Copyright © SAS Institute Inc. All rights reserved.

3.04 Activity

Open **s103a04.sas** from the **activities** folder and perform the following tasks to use a non-equijoin.

1. Complete the ON clause to join on rows where customer **Income** is greater than the **LowIncome** range, and less than or equal to the **HighIncome** range using the BETWEEN-AND WHERE operator.
2. What tax bracket is Olga Comstock in?
3. View your log. What note do you see?

35


Copyright © SAS Institute Inc. All rights reserved.

Syntax Summary

```
SELECT col-name, col-name  
FROM table1 INNER JOIN table2  
ON table1.column = table2.column;
```

Inner Join



```
SELECT col-name, col-name  
FROM table1 INNER JOIN table2  
ON table1.column = table2.column  
INNER JOIN table3  
ON join criteria;
```

Joining More Than Two Tables

```
ON table1.column < table2.column AND  
table1.column > table2.column;
```

Non-equijoin

PROC SQL FEEDBACK;

PROC SQL Options

38
Copyright © SAS Institute Inc. All rights reserved.





Practice

Level 1

1. Inner Join

Open **s103p01.sas** from the practices folder. Modify the program to generate a report that shows the breakdown of employment and marital status for customers in New York City.

- Add a PROC SQL step to create a table named **work.nyc** that combines **sq.customer** and **sq.maritalcode**. Follow the requirements below.
 - This table should include only **FirstName**, **LastName**, **Employed**, and **MaritalStatus**.
 - Perform an inner join on the **Married** column in the **sq.customer** table and **MaritalCode** column in the **sq.maritalcode** table.
 - Filter the **Zip** column for customers in the **10001** ZIP code.

Partial **work.nyc** Table

FirstName	LastName	Employed	MaritalStatus
Janet	Turner	Y	Single
Lawrence	Athas	Y	Married
Robert	Freydel	Y	Single
Mary	Remaklus	Y	Divorced
Lisa	Brunson	N	Married

- Execute the PROC FREQ step to generate the crosstabulation of **MaritalStatus** and **Employed**.

PROC FREQ Results

Marital Status by Employment for NYC Customers				
The FREQ Procedure				
Frequency Percent Row Pct Col Pct	Table of Marital Status by Employed			
	MaritalStatus	Employed		
		Y	N	Total
Married	1693 36.57 60.01 64.20	1128 24.37 39.99 56.63	2821 60.94	
Single	430 9.29 43.43 16.31	560 12.10 56.57 28.11	990 21.39	
Divorced	424 9.16 63.28 16.08	246 5.31 36.72 12.35	670 14.47	
Widowed	90 1.94 60.81 3.41	58 1.25 39.19 2.91	148 3.20	
	Total	2637 56.97	1992 43.03	4629 100.00

- c. Overall, are NYC customers likely to be employed or unemployed?
- d. Does this vary across marital status?

Level 2**2. Join on Inequality**

Open **s103p02.sas** from the practices folder. Modify the program to join the **sq.customer** and **sq.agegroup** tables based on a customer's year of birth.

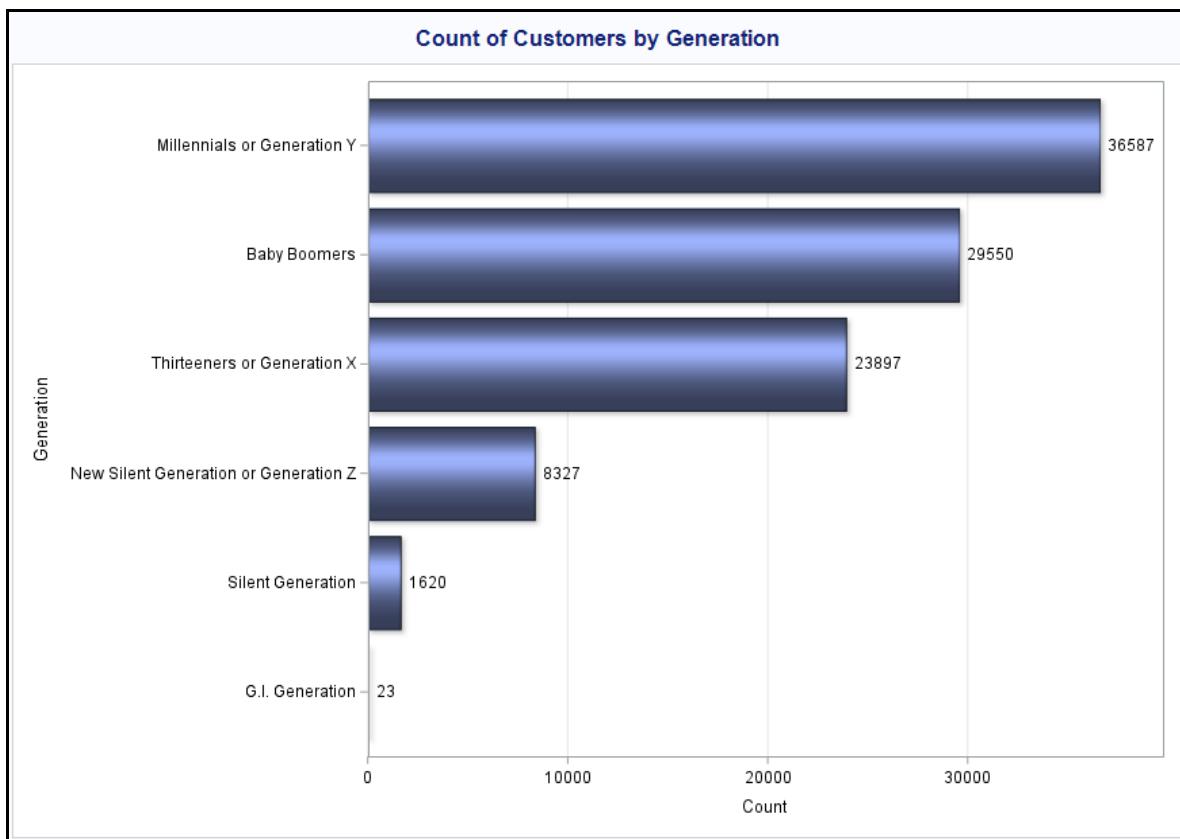
- a. Add a PROC SQL step to the top of the program to create a table named **work.generation** that combines **sq.customer** and **sq.agegroup**. Follow the requirements below.
 - 1) Select **FirstName** and **LastName**, and create a column named **Year** to determine the **DOB** year of the customer from the **sq.customer** table. Select the **Name** column from the **sq.agegroup** table.

- 2) The **StartYear** and **EndYear** columns in the **sq.agegroup** table indicate the starting and ending years for each generation. Use these columns to perform a non-equijoin using the calculated **Year** value from step 1) above.

Partial **work.generation** Table

FirstName	LastName	Year	Name
Becky	Cheers	1945	Silent Generation
Kathryn	Mathews	1940	Silent Generation
James	Frederick	1945	Silent Generation
Byron	Pray	1934	Silent Generation
Lauren	Mitchell	1934	Silent Generation
Pat	Johnson	1943	Silent Generation
Philip	Marshall	1944	Silent Generation
Stephen	Hinkle	1941	Silent Generation
James	Kline	1939	Silent Generation
Gaulo	Green	1944	Silent Generation

- b. Execute the PROC SGPLOT step below your query to generate the bar chart shown below.



- c. Which age group has the most customers?

Challenge

3. Joining More Than Two Tables

Open **s103p03.sas** from the practices folder. Modify the program to create a table named **work.births3** that will be used to create a report showing a three-year projection of births by **Region**, **Division**, and **State**. The data to create this report is stored in four tables as described below:

- a. **sq.statepopulation** contains
 - 1) **Region** (the region code)
 - 2) **Division** (the division code)
 - 3) **Name** (the state code)
 - 4) **Births3** (an indicator of state-level, three-year projected births).
- b. **sq.regioncode** contains
 - 1) **RegionCode**
 - 2) **RegionName**.
- c. **sq.divisioncode** contains
 - 1) **DivisionCode**
 - 2) **DivisionName**.
- d. **sq.statecode** contains
 - 1) **StateCode**
 - 2) **StateName**.
- e. Using all inner joins, combine the four tables above such that the descriptive **Region**, **Division**, and **State** names are combined with the **Births3** data.

Partial **work.Births3** Data

	RegionName	DivisionName	StateName	Births3
1	South Region	East South Central	Alabama	57216
2	West Region	Pacific	Alaska	10693
3	West Region	Mountain	Arizona	83550
4	South Region	West South Central	Arkansas	37191
5	West Region	Pacific	California	477145
6	West Region	Mountain	Colorado	66249
7	Northeast Regi...	New England	Connecticut	35048
8	South Region	South Atlantic	Delaware	10709
9	South Region	South Atlantic	District of Columbia	9788
10	South Region	South Atlantic	Florida	221488

- f. Execute the PROC TABULATE step to generate the required report and answer the following questions:

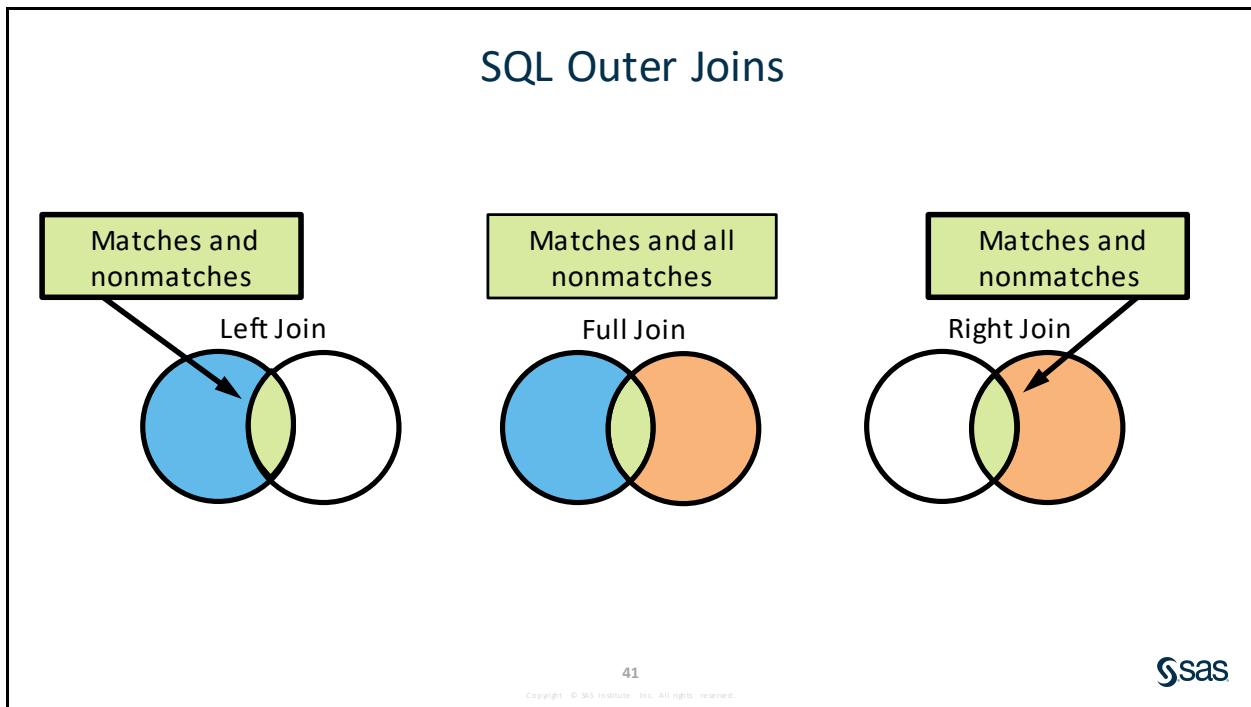
- 1) Which division has the highest projected three-year births, and which has the lowest?
- 2) Which region has the highest projected three-year births, and which has the lowest?

Partial PROC TABULATE Output

Projected 3-Year Births by Region/Division/State				
			Projected 3-Year Births	
Region	Division	State		
Midwest Region	East North Central	Illinois	148,117	
		Indiana	81,075	
		Michigan	110,676	
		Ohio	134,734	
		Wisconsin	64,939	
		Division Total	539,541	
	West North Central	State		
		Iowa	38,417	
		Kansas	36,637	
		Minnesota	68,617	
		Missouri	72,508	
		Nebraska	25,899	
		North Dakota	10,869	
		South Dakota	11,943	
Northeast Region	Division	Division Total	264,890	
		Region Total	804,431	
Northeast Region	Division	State		

End of Practices

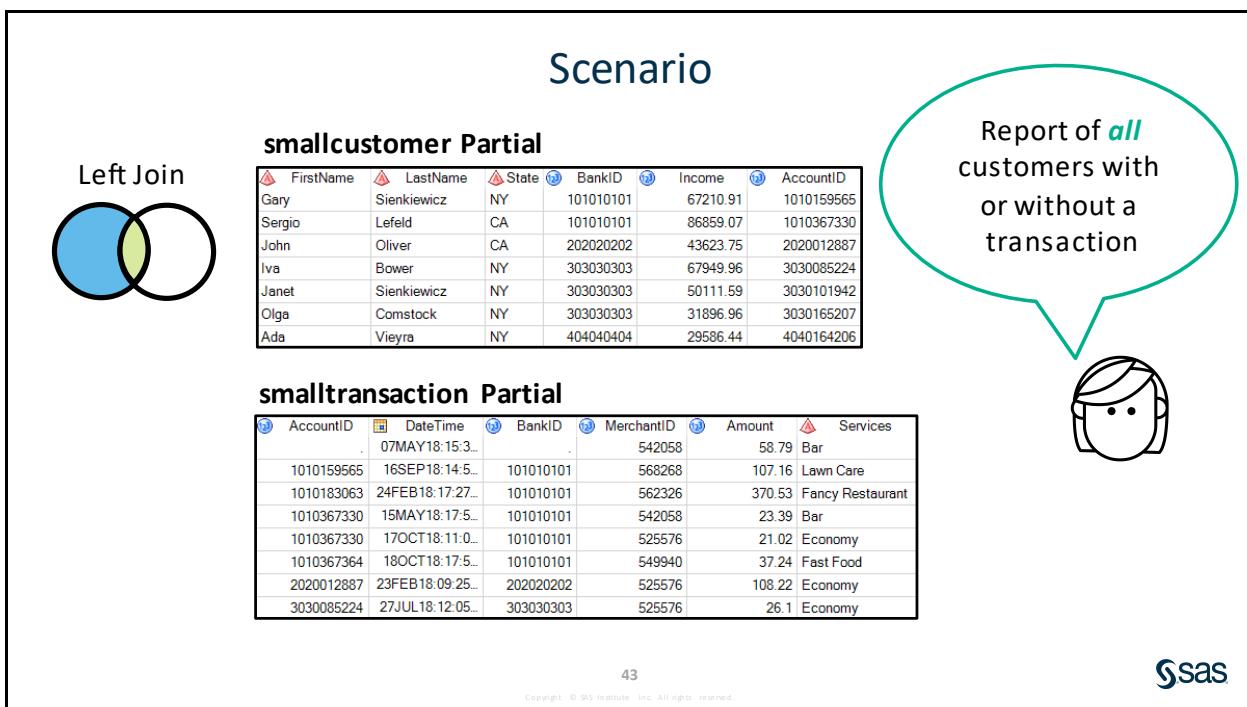
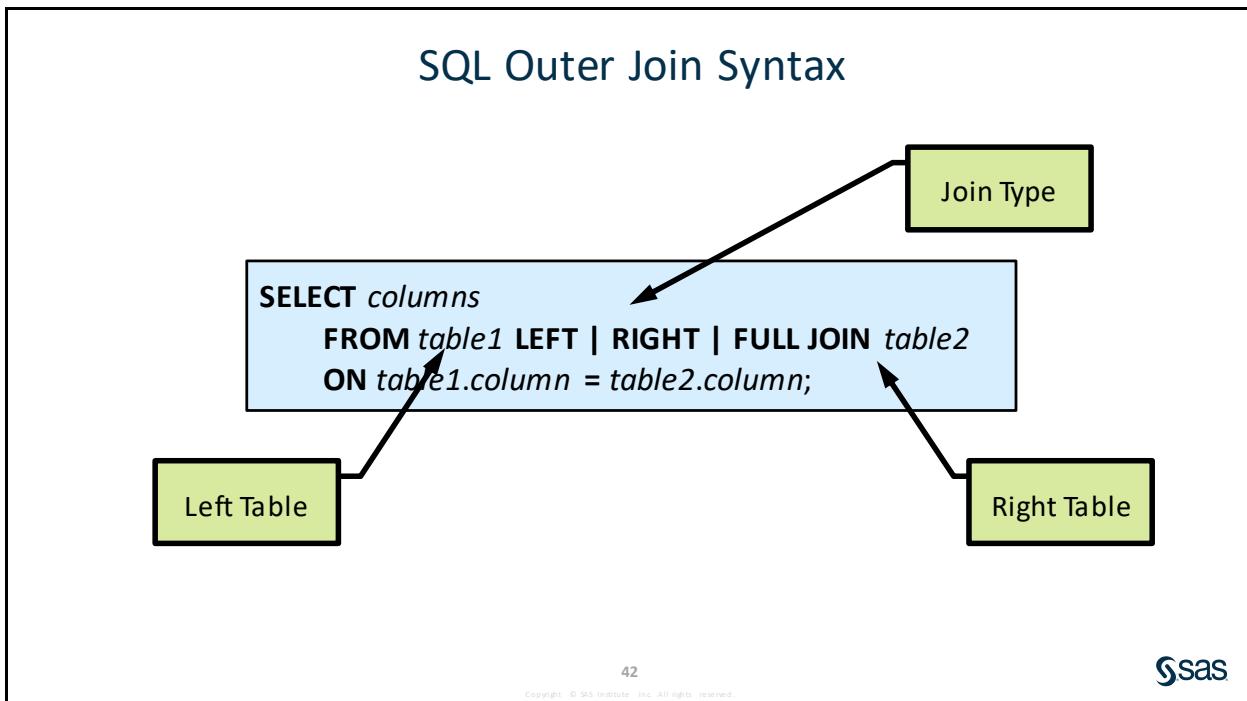
3.3 Outer Joins



Left joins return all rows from the left table and matching rows from the right table.

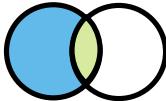
Right joins return all rows from the right table and the matching rows from the left.

Full joins return all matching and nonmatching rows from both tables.



SQL Left Join Syntax

Left Join



```
select *
  from sq.smallcustomer as c left join
       sq.smalltransaction as t
  on c.AccountID = t.AccountID;
```

FirstName	LastName	State	BankID	Income	AccountID	AccountID	DateTime	BankID
Gary	Sienkiewicz	NY	101010101	67210.91	1010159565	1010159565	16SEP18:14:57:08	10101010
Sergio	Lefeld	CA	101010101	86859.07	1010367330	1010367330	17OCT18:11:02:38	10101010
Sergio	Lefeld	CA	101010101	86859.07	1010367330	1010367330	15MAY18:10:07:14	10101010
John	Oliver	CA	202020202	43623.75	2020012887	2020012887	23FEB18:11:02:38	10101010
Iva	Bower	NY	303030303	67949.96	3030085224	3030085224	27JUL18:12:13:40	30303030
Janet	Sienkiewicz	NY	303030303	50111.59	3030101942	3030101942	18SEP18:12:13:40	30303030
Olga	Comstock	NY	303030303	31896.96	3030165207	3030165207	11MAR18:10:07:14	30303030
Ada	Vieyra	NY	404040404	29586.44	4040164206	4040164206	.	.

Report of *all* customers with or without transactions

44



A left outer join lists matching rows and rows from the left-hand table (the first table listed in the FROM clause) that do not match any row in the right-hand table. A left join is specified with the keywords LEFT JOIN and ON.

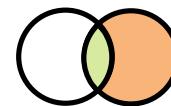
Scenario

Report of *all* transactions with or without a customer

smallcustomer Partial

FirstName	LastName	State	BankID	Income	AccountID
Gary	Sienkiewicz	NY	101010101	67210.91	1010159565
Sergio	Lefeld	CA	101010101	86859.07	1010367330
John	Oliver	CA	202020202	43623.75	2020012887
Iva	Bower	NY	303030303	67949.96	3030085224
Janet	Sienkiewicz	NY	303030303	50111.59	3030101942
Olga	Comstock	NY	303030303	31896.96	3030165207
Ada	Vieyra	NY	404040404	29586.44	4040164206

Right Join



smalltransaction Partial

AccountID	DateTime	BankID	MerchantID	Amount	Services
1010159565	07MAY18:15:3...	.	542058	58.79	Bar
1010159565	16SEP18:14:5...	101010101	568268	107.16	Lawn Care
1010183063	24FEB18:17:27...	101010101	562326	370.53	Fancy Restaurant
1010367330	15MAY18:17:5...	101010101	542058	23.39	Bar
1010367330	17OCT18:11:0...	101010101	525576	21.02	Economy
1010367364	18OCT18:17:5...	101010101	549940	37.24	Fast Food
2020012887	23FEB18:09:25...	202020202	525576	108.22	Economy
3030085224	27JUL18:12:05...	303030303	525576	26.1	Economy

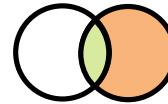
45



SQL Right Join Syntax

```
select *
  from sq.smallcustomer as c right join
       sq.smalltransaction as t
  on c.AccountID = t.AccountID;
```

Right Join



FirstName	LastName	State	BankID	Income	AccountID	AccountID	DateTime	BankID	MerchantID	Amount
Gary	Sienkiewicz	NY	101010101	67210.91	1010159565	1010159565	07MAY18:15:35:02		542058	58.75
Sergio	Lefeld	CA	101010101	86859.07	1010367330	1010367330	16SEP18:14:57:08	101010101	568268	107.10
Sergio	Lefeld	CA	101010101	86859.07	1010367330	1010367330	24FEB18:17:27:42	101010101	562326	370.50
John	Oliver	CA	202020202	43623.75	2020012887	2020012887	23FEB18:09:25:37	202020202	525576	108.25

Report of *all* transactions with or without a customer

46



A right join, specified with the keywords RIGHT JOIN and ON, is the opposite of a left join: nonmatching rows from the right-hand table (the second table listed in the FROM clause) are included with all matching rows in the output. This example reverses the join of the last example. It uses a right join to select all the transactions from the **smalltransaction** table and displays the customers only if the customer matches.

Selecting Columns in Inner Joins

```
proc sql;
select FirstName, LastName, Income, AccountID,
       DateTime, MerchantID, Amount
  from sq.smallcustomer as c inner join
       sq.smalltransaction as t
  on c.AccountID = t.AccountID;
quit;
```

c.AccountID

t.AccountID



With an inner join,
you can select either
AccountID column.



47

Copyright © SAS Institute Inc. All rights reserved.

Selecting Columns in Outer Joins

```

proc sql;
select FirstName, LastName, Income, AccountID,
      DateTime, MerchantID, Amount
from sq.smallcustomer as c left join
      sq.smalltransaction as t
on c.AccountID = t.AccountID;
quit;

```

Depending on which **AccountID** column we choose, our results differ.

48

Copyright © SAS Institute Inc. All rights reserved.



3.05 Activity

Open **s103a05.sas** from the **activities** folder and perform the following tasks to perform a left join:

1. Run the query to create a left join between the **sq.smallcustomer** and **sq.smalltransaction** tables. Notice the difference within the **AccountID** columns in rows 8 and 9.
2. Remove the column **t.AccountID** in the SELECT clause. Run the query and examine the results. How many missing **AccountID** values are in the results?
3. Replace **c.AccountID** with **t.AccountID**. Replace the **c** in the column label with a **t**. How many missing **AccountID** values are in the results?

49

Copyright © SAS Institute Inc. All rights reserved.

Scenario

Report of *all* customers and transactions.

smallcustomer Partial

FirstName	LastName	State	BankID	Income	AccountID
Gary	Sienkiewicz	NY	101010101	67210.91	1010159565
Sergio	Lefeld	CA	101010101	86859.07	1010367330
John	Oliver	CA	202020202	43623.75	2020012887
Iva	Bower	NY	303030303	67949.96	3030085224
Janet	Sienkiewicz	NY	303030303	50111.59	3030101942
Olga	Comstock	NY	303030303	31896.96	3030165207
Ada	Vieyra	NY	404040404	29586.44	4040164206

Full Join

smalltransaction Partial

AccountID	DateTime	BankID	MerchantID	Amount	Services
1010159565	07MAY18:15:3...	101010101	542058	58.79	Bar
1010183063	16SEP18:14:5...	101010101	568268	107.16	Lawn Care
1010367330	24FEB18:17:27...	101010101	562326	370.53	Fancy Restaurant
1010367330	15MAY18:17:5...	101010101	542058	23.39	Bar
1010367364	17OCT18:11:0...	101010101	525576	21.02	Economy
1010367364	18OCT18:17:5...	101010101	549940	37.24	Fast Food
2020012887	23FEB18:09:25...	202020202	525576	108.22	Economy
3030085224	27JUL18:12:05...	303030303	525576	26.1	Economy

52

SQL Full Join Syntax

```
select *
  from sq.smallcustomer as c full join
       sq.smalltransaction as t
  on c.AccountID = t.AccountID;
```

Full Join

FirstName	LastName	State	BankID	Income	AccountID	AccountID	DateTime	BankID	Mercha
							07MAY18:15:35:02		54
Gary	Sienkiewicz	NY	101010101	67210.91	1010159565	1010159565	16SEP18:14:57:08	101010101	56
							1010183063	24FEB18:	
Sergio	Lefeld	CA	101010101	86859.07	1010367330	1010367330	17OCT18:		
Sergio	Lefeld	CA	101010101	86859.07	1010367330	1010367330	15MAY18:17:54:21	101010101	54
John	Oliver	CA	202020202	43623.75	2020012887	2020012887	23FEB18:09:25:27	202020202	52

Report of *all* customers and transactions

53

SQL Full Join Syntax

FirstName	LastName	State	BankID	Income	AccountID	AccountID	DateTime	BankID	Merch
			07MAY18:15:35:02	.	54
Gary	Sienkiewicz	NY	101010101	67210.91	1010159565	1010159565	16SEP18:14:57:08	101010101	56
			.	.	.	1010183063	24FEB18:17:27:42	101010101	56
Sergio	Lefeld	CA	101010101	86859.07	1010367330	1010367330	17OCT18:11:02:38	101010101	52
Sergio	Lefeld	CA	101010101	86859.07	1010367330	1010367330	15MAY18:17:54:21	101010101	54
			.	.	.	1010367364	18OCT18:17:52:51	101010101	54
John	Oliver	CA	202020202	12622.75	2020010007	2020010007	22FEB18:00:25:27	202020202	50

Notice that the **AccountID** values from each table are not always equal.

COALESCE Function

COALESCE(argument-1, argument-2<, ...argument-n>)

```
select coalesce(c.AccountID, t.AccountID) as AccountID
```

c.AccountID	t.AccountID
.	1010183063
4040164206	.
3030165207	3030165207

The COALESCE function returns the value of the **first nonmissing argument**.

You can use the *COALESCE function* to overlay columns.



Performing a Full Join with PROC SQL

Scenario

Use PROC SQL to perform a full join between two tables.

Files

- **s103d03.sas**
- **smallcustomer** – a SAS table that contains one row per customer
- **smalltransaction** – a SAS table that contains customer transaction data

Syntax

```
PROC SQL;
SELECT col-name, col-name
    FROM table1 FULL JOIN table2
        ON table1.col-name=table2.col-name;
QUIT;

COALESCE(argument-1, argument-2<, ...argument-n>)
```

Notes

- The two tables to be joined are listed in the FROM clause separated by FULL JOIN.
- A full join lists all values in both tables, with or without a match.
- The COALESCE function returns the value of the first nonmissing argument.

Demo

1. Open the **s103d03.sas** program in the **demos** folder and find the **Demo** section of the program. Highlight and run the query. Examine the results. Discuss the values in both **AccountID** columns.

```
proc sql;
select FirstName, LastName, Income, c.AccountID, t.AccountID,
       DateTime, MerchantID, Amount
   from sq.smallcustomer as c full join sq.smalltransaction as t
     on c.AccountID = t.AccountID;
quit;
```

2. Modify the SELECT clause and remove the **t.AccountID** column. Highlight and run the query. Examine the results. Discuss the missing values in the **c.AccountID** column.

```
proc sql;
select FirstName, LastName, Income, c.AccountID, t.AccountID,
       DateTime, MerchantID, Amount
  from sq.smallcustomer as c full join
       sq.smalltransaction as t
  on c.AccountID = t.AccountID;
quit;
```

3. Modify the SELECT clause and replace **c.AccountID** with **t.AccountID**. Highlight and run the query. Examine the results. Discuss the missing values in the **t.AccountID** column.

```
proc sql;
select FirstName, LastName, Income, t.AccountID,
       DateTime, MerchantID, Amount
  from sq.smallcustomer as c full join
       sq.smalltransaction as t
  on c.AccountID = t.AccountID;
quit;
```

4. Modify the SELECT clause, use the COALESCE function, and add **c.AccountID** and **t.AccountID** as arguments. Add the alias **AccountID** and the **FORMAT=10.** column modifier to the newly created column. Highlight and run the query. Examine the results.

```
proc sql;
select FirstName, LastName, Income,
       coalesce(c.AccountID,t.AccountID) as AccountID format=10.,
       DateTime, MerchantID, Amount
  from sq.smallcustomer as c full join
       sq.smalltransaction as t
  on c.AccountID = t.AccountID;
quit;
```

FirstName	LastName	Income	AccountID	DateTime	MerchantID	Amount
		.	.	07MAY18:15:35:02	542058	58.79
Gary	Sienkiewicz	67210.91	1010159565	16SEP18:14:57:08	568268	107.16
		.	1010183063	24FEB18:17:27:42	562326	370.53
Sergio	Lefeld	86859.07	1010367330	17OCT18:11:02:38	525576	21.02
Sergio	Lefeld	86859.07	1010367330	15MAY18:17:54:21	542058	23.39
		.	1010367364	18OCT18:17:52:51	549940	37.24
John	Oliver	43623.75	2020012887	23FEB18:09:25:37	525576	108.22
Iva	Bower	67949.96	3030085224	27JUL18:12:05:48	525576	26.1
Janet	Sienkiewicz	50111.59	3030101942	18SEP18:12:13:40	549940	37.38
Olga	Comstock	31896.96	3030165207	11MAR18:10:07:14	580881	319.95
		.	3030231909	13FEB18:16:48:05	513178	115.48
		.	3030231909	27JAN18:13:52:35	536123	1.76
Ada	Vieyra	29586.44	4040164206	.	.	.
Samantha	Carney	25476.14	5540174271	.	.	.

End of Demonstration

Identifying Nonmatching Rows

```
select FirstName, LastName, Income,
       c.AccountID "c.AccountID",
       t.AccountID "t.AccountID",
       DateTime, MerchantID
  from sq.smallcustomer as c left join
       sq.smalltransaction as t
     on c.AccountID = t.AccountID;
```

FirstName	LastName	Income	c.AccountID	t.AccountID	DateTime	MerchantID
Gary	Sienkiewicz	67210.91	1010159565	1010159565	16SEP18:14:57:08	568268
Sergio	Lefeld	86859.07	1010367330	1010367330	17OCT18:11:02:38	525576
Sergio	Lefeld	86859.07	1010367330	1010367330	15MAY18:17:54:21	542058
John	Oliver	43623.75	2020012887	2020012887	23FEB18:09:25:37	525576
Iva	Bower	67949.96	3030085224	3030085224	27JUL18:12:05:48	525576
Janet	Sienkiewicz	50111.59	3030101942	3030101942	18SEP18:12:13:40	549940
Olga	Comstock	31896.96	3030165207	3030165207	11MAR18:10:07:14	580881
Ada	Vieyra	29586.44	4040164206	.	.	.
Samantha	Carney	25476.14	5540174271	.	.	.

Produce a list of
customers who **don't**
have a transaction.



sas

Identifying Nonmatching Rows

```
select FirstName, LastName, Income,
       c.AccountID "c.AccountID",
       t.AccountID "t.AccountID",
       DateTime, MerchantID
  from sq.smallcustomer as c left join
       sq.smalltransaction as t
     on c.AccountID = t.AccountID;
```

FirstName	LastName	Income	c.AccountID	t.AccountID	DateTime	MerchantID
Gary	Sienkiewicz	67210.91	1010159565	1010159565	16SEP18:14:57:08	568268
Sergio	Lefeld	86859.07	1010367330	1010367330	17OCT18:11:02:38	525576
Sergio	Lefeld	86859.07	1010367330	1010367330	15MAY18:17:54:21	542058
John	Oliver	43623.75	2020012887	2020012887	23FEB18:09:25:37	525576
Iva	Bower	67949.96	3030085224	3030085224	27JUL18:12:05:48	525576
Janet	Sienkiewicz	50111.59	3030101942	3030101942	18SEP18:12:13:40	549940
Olga	Comstock	31896.96	3030165207	3030165207	11MAR18:10:07:14	580881
Ada	Vieyra	29586.44	4040164206	.	.	.
Samantha	Carney	25476.14	5540174271	.	.	.

customers who do not have
a transaction

Copyright © SAS Institute Inc. All rights reserved.

sas

Identifying Nonmatching Rows

```
select FirstName, LastName, Income,
       c.AccountID "c.AccountID",
       t.AccountID "t.AccountID",
       DateTime, MerchantID
  from sq.smallcustomer as c left join
       sq.smalltransaction as t
      on c.AccountID = t.AccountID
   where t.AccountID is null;
```

The WHERE clause filters for all customers with a missing transaction AccountID.

FirstName	LastName	Income	c.AccountID	t.AccountID	DateTime	MerchantID
Ada	Vieyra	29586.44	4040164206	.	.	.
Samantha	Carney	25476.14	5540174271	.	.	.

3.06 Activity

Open **s103a06.sas** from the **activities** folder and perform the following tasks to find all transactions not associated with a documented customer:

1. Run the query to create a left join between the **sq.smalltransaction2** and **sq.smallcustomer2** tables. Examine the report. Notice that the rows with missing values in **AccountID** have been joined.
2. In the ON clause, add the expression **AND t.AccountID is not null**. Run the query. Confirm that missing values were not joined.
3. Add a WHERE clause with the expression **c.AccountID is NULL** to filter for all transactions without a documented customer. Run the query and examine the report. How many transactions do not have a customer associated with them?

Syntax Summary



```
SELECT columns
      FROM table1 <LEFT | RIGHT | FULL > JOIN table2
           ON table1.column = table2.column
```

Outer Join

```
COALESCE(argument-1, argument-2, ...argument-n)
```

COALESCE Function



Practice

Level 1

4. Using Outer Joins to Find Nonmatches

Join the **sq.globalpop** and **sq.globalmetadata** tables to create the **work.meta** table. Use the **work.meta** table to generate a report showing the country codes for countries in the **sq.globalpop** table that do not have any country metadata in the **sq.globalmetadata** table.

- Create a table from the join. Name the table **work.meta**.
- Select the **CountryCode**, **SeriesName**, **EstYear1**, and **EstYear3** columns from the **sq.globalpop** table and the **ShortName** and **IncomeGroup** columns from the **sq.globalmetadata** table.
- Perform a left join on the **sq.globalpop** and **sq.globalmetadata** tables.
- Use the **CountryCode** column in both tables for the join criteria.

Partial **work.meta** Table

CountryCode	SeriesName	EstYear1	EstYear3	ShortName	IncomeGroup
ABW	Population 70-74	3179	3641		
ABW	Population 65-69	4638	5339		
ABW	Population 60-64	6247	6983		
ABW	Population 00-04	5810	5394		
ABW	Population 10-14	7231	7060		
ABW	Population 40-44	7645	7236		
ABW	Population 80+	2048	2355		
ABW	Population 45-49	8485	7899		
ABW	Population 30-34	5857	5399		
ABW	Population 50-54	9264	8999		
ABW	Population 25-29	5127	6063		
ABW	Population 20-24	6979	7679		
ABW	Population 35-39	6821	6325		
ABW	Population 75-79	2290	2435		
ABW	Population 55-59	7783	8610		
ABW	Population 05-09	6813	6396		
ABW	Population 15-19	7581	7443		
AFG	Population 25-29	2412180	2699261	Afghanistan	Low income

- Create a report showing the unique country codes for which there is no global metadata using the **work.meta** table.
 - Select the **CountryCode** column from the **work.meta** table and eliminate duplicate values.

- 2) Filter for rows where the **ShortName** column is missing. The **ShortName** column contains values from the **sq.globalmetadata** table. If the results are missing, then the row did not retrieve information from **sq.globalmetadata**.
- 3) Order the results by **CountryCode**.
- f. What is the last **CountryCode** value in your results?

Partial Results

CountryCode
ABW
ATG
BHS
BRB
BRN
CER

Level 2

5. Using Outer Joins to Summarize Data

Generate a report showing the count of customer marital status descriptions for each primary bank. The **sq.customer** table contains a marital code (**Married**) and a primary bank ID. The final results should contain **BankID**, **MaritalStatus**, **Name** (name of bank), and **Count**.

- a. Write a PROC SQL step to join the **sq.customer** table and the **sq.maritalcode** table.
- 1) Select **BankID** from the **sq.customer** table and **MaritalStatus** value from the **sq.maritalcode** table. Create a new column named **Count** to count the number of customers. Format the new column using commas.
 - 2) Use a left join to select all customers from the **sq.customer** table, with or without matches in the **sq.maritalcode** table.
 - 3) Use the **Married** column in the **sq.customer** table and the **MaritalCode** column in the **sq.maritalcode** table as the join criteria.
 - 4) Filter rows where the customer's bank ID is not missing.
 - 5) Group the data by **BankID** and **MaritalStatus**.
 - 6) Order the table by descending **Count**.
 - 7) As a checkpoint, run the query.

Partial Results

BankID	MaritalStatus	Count
101010101	Married	22,933
202020202	Married	16,997
303030303	Married	14,258
101010101	Single	8,793
202020202	Single	6,589

- b. In the same PROC SQL step, add the descriptive bank **Name** column to the results.
- 1) After the **MaritalStatus** column in the SELECT clause, add **Name** from the **sq.bank** table based on matching **BankID** values, again using a left join.
 - 2) Add **Name** in the GROUP BY clause after the **MaritalStatus** column.
 - 3) Correctly reference the **BankID** columns in the query.
 - 4) Add an appropriate title.

Partial Results

Count of Marital Status by Bank			
BankID	MaritalStatus	Name	Count
101010101	Married	Biggest Bank, Inc.	22,933
202020202	Married	Sailors Credit Union	16,997
303030303	Married	Wheatberry Bank, Inc.	14,258
101010101	Single	Biggest Bank, Inc.	8,793
202020202	Single	Sailors Credit Union	6,589
303030303	Single	Wheatberry Bank, Inc.	5,458
101010101	Divorced	Biggest Bank, Inc.	4,896

- c. Which combination of **MaritalStatus** and **Name** had the lowest count of customers?

Challenge**6. Combining Inner and Outer Joins**

- a. Some of our New York City merchants are having difficulty with sales. Your job is to identify merchants who have had no recent sales, capture what industry they are in, and then generate a list of customers in their area who are transacting with other vendors to help them market the appropriate customers to generate sales.

- 1) Use a left outer join to combine the **sq.merchant** and **sq.transaction** tables to identify New York City merchants (**Zip=10001**) with no transactions. From the **sq.merchant** table, select **MerchantName**, **MerchantID**, **Type**, and **Zip**. Label **Type** as **Merchant Type** and **Zip** as **Merchant Zipcode**. Merchants who do not have any transactions will be those who have no entries in the **sq.transaction** table.
- 2) As a checkpoint, the NYC merchants with no transactions are shown below.

Results

NYC Merchants with no Transactions			
Merchant Name	Merchant ID	Merchant Type	Merchant Zipcode
Good Service Auto Repair	502136	Auto	10001
Gifting Hands	517039	Charity	10001
Miasma Mitigation, Inc.	518339	Charity	10001
13th Street Grocery	565543	Groceries	10001
Gerivest Services	585048	Retirement	10001

- b. The next step is to find the customers in NYC who have similar **Type** transactions with other vendors. The problem is that the **sq.transactionfull** table (where transaction information is combined with customer information) does not contain a column for the customer's ZIP code for us to join on, only the full address. Therefore, we need to extract the ZIP code from the **customer.Address** column to perform the final join.

- 1) In the same PROC SQL step, use an inner join to combine the above list with the **sq.transactionfull** table by **Type** and **Zip** to generate a report with customer contact information for all NYC customers with similar **Type** transactions.

Hint: Assign the **sq.transactionfull** table an alias of **c** and use the following expression to extract **Zip** from the **c.address** column to perform this join:

```
input (scan (c.address, -1) ,5 .)
```

- 2) Include **CustomerID**, **CustomerName**, and **Address** from the **sq.transactionfull** table. Label **Address** as **Customer Address**.
 - 3) Ensure that the report shows unique rows of data and is ordered by **merchantID** and **CustomerID**. Add an appropriate title to the report.

Partial PROC SQL Output

Customers with similar Type Transactions as NYC Merchants with no Sales						
Merchant Name	Merchant ID	Merchant Type	Merchant Zipcode	CustomerID	CustomerName	Customer Address
Good Service Auto Repair	502136	Auto		10001	Pennacchio, Joan Lynn	28 Crest Bend New York NY 10001
Good Service Auto Repair	502136	Auto		10001	Comstock, Olga Cathy	653 County Line Court New York NY 10001
Good Service Auto Repair	502136	Auto		10001	Alexander, Ruth Helen	14 16th Street New York NY 10001
Good Service Auto Repair	502136	Auto		10001	Bowers, Margaret Katie	173 Lakeshore Terrace New York NY 10001
Good Service Auto Repair	502136	Auto		10001	Bower, Omar Randy	367 Lake Lane New York NY 10001
Good Service Auto Repair	502136	Auto		10001	Sienkiewicz, Gary Roman	301 Wood Drive New York NY 10001
Gifting Hands	517039	Charity		10001	Pennacchio, Joan Lynn	28 Crest Bend New York NY 10001
Gifting Hands	517039	Charity		10001	Comstock, Olga Cathy	653 County Line Court New York NY 10001
Giving Hands	517039	Charity		10001	Alexander, Ruth Helen	14 16th Street New York NY 10001
Giving Hands	517039	Charity		10001	Bowers, Margaret Katie	173 Lakeshore Terrace New York NY 10001
Giving Hands	517039	Charity		10001	Bower, Omar Randy	367 Lake Lane New York NY 10001
Giving Hands	517039	Charity		10001	Sienkiewicz, Gary Roman	301 Wood Drive New York NY 10001

- c. How many potential new customers can the merchant Miasma Mitigation, Inc. now market?

End of Practices

3.4 Complex Joins

Reflexive Join

employee			
EmployeeID	EmployeeName	ManagerID	ManagerName
121044	Abbott, Ray	121144	
120145	Aisbitt, Sandy	120103	
120761	Akinfolarin, Tameaka	120746	
121144	Capachietti, Renee	121142	

The **employee** table includes a list of all employees.

Find **ManagerName** for each employee.



sas

67

Copyright © SAS Institute Inc. All rights reserved.

Reflexive Join

EmployeeID	EmployeeName	ManagerID	ManagerName
121044	Abbott, Ray	121144	
120145	Aisbitt, Sandy	120103	
120761	Akinfolarin, Tameaka	120746	
121144	Capachietti, Renee	121142	

Self-join on the **employee** table to retrieve manager names.



sas

68

Copyright © SAS Institute Inc. All rights reserved.

Required Table Aliases

```
FROM table1 <AS> alias1 JOIN-TYPE  
table1 <AS> alias2
```

```
select e.EmployeeID, e.EmployeeName,  
       e.StartDate format=date9.,  
       e.ManagerID,  
       m.EmployeeName as ManagerName  
  from sq.employee as e inner join  
       sq.employee as m  
  on e.ManagerID = m.EmployeeID;
```

To read the same table *twice*, list it *twice* in the FROM clause.



Sas



Performing a Reflexive Join

Scenario

Use PROC SQL to perform a reflexive join using a single table.

Files

- **s103d04.sas**
- **employee** – a SAS table that contains one row per employee

Syntax

```
PROC SQL;
  SELECT col-name, col-name
    FROM table1 INNER JOIN table1
      ON table1.col-name=table1.col-name;
QUIT;
```

Notes

- A reflexive join or self-join is used to join a table to itself as if the table were two tables.
- The ON expression indicates how rows should be matched. The column names must be qualified as *table-name.col-name*.

Demo

1. Open the **s103d04.sas** program in the **demos** folder and find the **Demo** section of the program. Highlight and run the query. Examine the results.

```
proc sql;
  select e.EmployeeID, e.EmployeeName, e.StartDate format=date9.,
         e.ManagerID
    from sq.employee as e;
quit;
```

2. Modify the query to create a reflexive join. In the FROM clause, add an inner join followed by the **sq.employee** table again. Add the alias **m** to the second **sq.employee** table. Add the ON clause and set **e.ManagerID** equal to **m.EmployeeID**.

```
proc sql;
  select e.EmployeeID, e.EmployeeName, e.StartDate format=date9.,
         e.ManagerID
    from sq.employee as e inner join
         sq.employee as m
      on e.ManagerID = m.EmployeeID;
quit;
```

3. Add the **EmployeeName** column in the SELECT clause. Qualify the new **EmployeeName** column with the table alias **m**. Highlight and run the query. Examine the results.

```
proc sql;
select e.EmployeeID, e.EmployeeName, e.StartDate format=date9.,
       e.ManagerID, m.EmployeeName
  from sq.employee as e inner join
       sq.employee as m
  on e.ManagerID = m.EmployeeID;
quit;
```

4. Add the column alias **ManagerName** to the **m.EmployeeName** column and an ORDER BY clause to sort by **ManagerName**. Highlight and run the query. Examine the results.

```
proc sql;
select e.EmployeeID, e.EmployeeName, e.StartDate format=date9.,
       e.ManagerID, m.EmployeeName as ManagerName
  from sq.employee as e inner join
       sq.employee as m
  on e.ManagerID = m.EmployeeID
  order by ManagerName;
quit;
```

EmployeeID	EmployeeName	StartDate	ManagerID	ManagerName
120803	Droste, Victor	01JAN1990	120798	Ardskin, Elizabeth
120804	Zied, Ahmed	01JAN1986	120798	Ardskin, Elizabeth
120808	Dupree, Marcel	01JUN1996	120798	Ardskin, Elizabeth
120791	Chiseloff, Richard	01OCT1998	120798	Ardskin, Elizabeth
120805	Walker, Robert	01APR2012	120798	Ardskin, Elizabeth
120806	Ousley, Lorna	01FEB2004	120798	Ardskin, Elizabeth
120801	Kennedy, Kathryn	01JUL2011	120798	Ardskin, Elizabeth

End of Demonstration

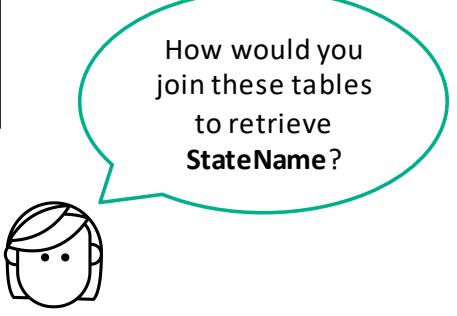
Scenario

transactionfull

StateID	CustomerName
CA37492351	Caberto, Glen Daniel
CA53344918	Lefeld, Sergio Vance
CA95831948	Lefeld, Linda Erica
NY67246023	Bowers, Margaret Katie
CA57669199	Kennedy, Lisa Diane
CA95831948	Lefeld, Linda Erica
NY14984651	Balo, Cynthia Patricia

statecode

StateCode	StateName
AL	Alabama
AK	Alaska
AZ	Arizona
AR	Arkansas
CA	California
CO	Colorado
CT	Connecticut
DE	Delaware



How would you join these tables to retrieve StateName?

71

Copyright © SAS Institute Inc. All rights reserved.

Sas

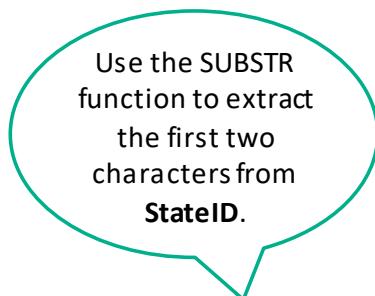
Using Functions to Join Tables

transactionfull

StateID	CustomerName
CA37492351	Caberto, Glen Daniel
CA53344918	Lefeld, Sergio Vance
CA95831948	Lefeld, Linda Erica
NY67246023	Bowers, Margaret Katie
CA57669199	Kennedy, Lisa Diane
CA95831948	Lefeld, Linda Erica
NY14984651	Balo, Cynthia Patricia

statecode

StateCode	StateName
AL	Alabama
AK	Alaska
AZ	Arizona
AR	Arkansas
CA	California
CO	Colorado
CT	Connecticut
DE	Delaware



Use the SUBSTR function to extract the first two characters from StateID.

```
select StateID, CustomerName, StateName
  from sq.transactionfull as t inner join
       sq.statecode as s
  on substr(t.StateID,1,2) = s.StateCode;
```

72

Copyright © SAS Institute Inc. All rights reserved.

Sas

3.07 Activity

Open **s103a07.sas** from the **activities** folder and perform the following tasks to join two tables using the SUBSTR function:

1. Examine and run the query. Did you receive a syntax error?
2. In the ON clause, use the SUBSTR function on **t.StateID** to extract the **first two characters**. Run the query. Which **StateName** is *Caberto*, *Glen Daniel* from?

Using Functions to Join Tables

customerzip

CustomerID	ZipCode	Gender	Employed
1	14580	M	Y
2	04429	M	Y
3	50101	M	Y
4	27510	M	N

Character ZipCode

sashelp.zipcode

ZIP	x	y	CITY	STATECODE
501	-73.046388	40.813078	Holtsville	NY
544	-73.049288	40.813223	Holtsville	NY
601	-66.723627	18.16595	Adjuntas	PR
602	-73.106552	40.902005	Ayey	PR

Numeric ZIP

Can the columns you use to join tables have a different **column type**?



3.08 Activity

Open **s103a08.sas** from the **activities** folder and perform the following tasks to join a **numeric** column with a **character** column:

1. Run the queries in the **Create a Table and Insert Values** section. View the newly created table.
2. Run the query in the **Join Different Column Types** section. What syntax error was generated when you joined columns of different types?
3. Leave the program open for the next activity.

Converting Column Value Functions



Function	What it does
<code>INPUT(source, informat)</code>	Converts character values to numeric values using a specified informat
<code>PUT(source, format)</code>	Converts numeric or character values to character values using a specified format

For more information about the PUT function, see "PUT Function" in the *SAS® 9.4 Functions and CALL Routines: Reference, Fifth Edition* documentation. You can also find the direct link in the Course Links section on the ELP.

For more information about the INPUT function, see "INPUT Function" in the *SAS® 9.4 Functions and CALL Routines: Reference, Fifth Edition* documentation. You can also find the direct link in the Course Links section on the ELP.

Converting Numeric to Character Values

`put(z.Zip,z5.)`

The **Z format** writes standard numeric data with leading 0s.

The PUT function with the **Z format** converts the numeric ZIP code 4429 to 04429.



3.09 Activity

Use **s103a08.sas** from the previous activity and perform the following tasks to join a **numeric** column with a **character** column by using the PUT function.

1. If you have not run the queries in the **Create a Table and Insert Values** section, run those now.
2. Use the PUT function to convert **z.Zip** in the ON clause to a character value using the z5 format. Run the query.
3. What city does the **ZipCode** value 14216 represent?

Beyond SQL Essentials

What if you want to ...

... learn more about merging using the DATA step?

- Take the [SAS Programming 2 course](#) to learn more about DATA step match-merges.
- Visit the **SQL and the DATA Step** section on the ELP for additional resources about comparisons of the DATA step and PROC SQL.

... download the SQL Join Summary cheat sheet?

- Visit the **Course Handouts** section on the ELP and download the **SQL Join Summary** PDF.

... learn more about functions in PROC SQL?

- Read the SAS paper [Top 10 Most Powerful Functions for PROC SQL](#).

3.5 Solutions

Solutions to Practices

1. Inner Join

```
/*s103s01.sas*/

proc sql;
create table work.NYC as
select c.FirstName, c.LastName, c.Employed, m.MaritalStatus
  from sq.customer as c inner join
       sq.maritalcode as m
  on c.married=m.MaritalCode
  where zip=10001;
quit;

/*Alternate Solution*/
proc sql;
create table work.NYC as
select c.FirstName, c.LastName, c.Employed, m.MaritalStatus
  from sq.customer as c, sq.maritalcode as m
  where c.married=m.MaritalCode and
        zip=10001;
quit;

title 'Marital Status by Employment for NYC Customers';
proc freq data=work.NYC order=Freq;
  tables MaritalStatus*employed;
run;
title;
```

Overall, are NYC customers likely to be employed or unemployed? **Employed**

Does this vary across marital status? **Yes, NYC customers whose marital status is *Single* are more likely to be unemployed.**

2. Join on Inequality

```
/*s103s02.sas*/

/*Solution*/
proc sql;
create table work.generation as
select c.FirstName,
       c.LastName,
       year(c.DOB) as Year,
       a.Name
  from sq.Customer as c inner join
       sq.AgeGroup as a
```

```

    on year(c.DOB) between a.StartYear and a.EndYear;
quit;

/*Alternate Solution*/
proc sql;
create table work.generation as
select c.FirstName,
       c.LastName,
       year(c.DOB) as Year,
       a.Name
  from sq.Customer as c, sq.AgeGroup as a
 where calculated Year between a.StartYear and a.EndYear;
quit;

/*Bar Chart*/
title 'Count of Customers by Generation';
proc sgplot data=work.generation noautolegend;
  hbar Name /
    stat=freq
    dataskin=sheen
    categoryorder=respdesc
    datalabel
    datalabelattrs=(size=9pt)
    FILLATTRS=(color=cx6f7eb3);
    yaxis label='Generation';
    xaxis grid label='Count';
run;
title;

```

Which age group has the most customers? **Millennials or Generation Y**

3. Joining More Than Two Tables

```

/*s103s03.sas*/

proc sql;
create table Births3 as
select c.RegionName, d.DivisionName, b.Statename, a.Births3
  from sq.statepopulation as a inner join
       sq.statecode as b
  on a.name=b.statecode inner join
       sq.RegionCode as c
  on a.region=c.regionCode inner join
       sq.divisioncode as d
  on a.Division=d.divisioncode;
quit;

/*Alternate Solution*/
proc sql;
create table Births3 as

```

```

select c.RegionName, d.DivisionName, b.Statename, a.Births3
  from sq.statepopulation as a,
       sq.statecode as b,
       sq.RegionCode as c,
       sq.divisioncode as d
 where a.name=b.statecode and
       a.region=c.regionCode and
       a.Division=d.divisioncode;
quit;

title 'Projected 3-Year Births by Region/Division/State';
proc tabulate data=WORK.BIRTHS3;
  var Births3;
  class RegionName / order=unformatted missing;
  class DivisionName / order=unformatted missing;
  class StateName / order=unformatted missing;
  table
    Sum={label=""}*(  

      RegionName={label= 'Region'}*(  

        DivisionName={label= 'Division'}*(  

          StateName={label="State"}  

          all={label= 'Division Total'}*{style={BACKGROUND=#FFFF99}})  

          all={label= 'Region Total'}*{style={BACKGROUND=#CCFFFF}})  

          all={label= 'Overall Total'}*{style={BACKGROUND=#CCFFCC}}),  

        Births3={label= 'Projected 3-Year Births'}*F=comma12.;
run;

```

Which division has the highest projected three-year births, and which has the lowest?

South Atlantic has the highest. New England has the lowest.

Which region has the highest projected three-year births, and which has the lowest?

South region has the highest. Northeast region has the lowest.

4. Using Outer Joins to Find Nonmatches

```

/*s103s04.sas*/
*****;
*Solution with Temporary Table      *;
*****;
proc sql;
create table work.meta as
select p.CountryCode, p.SeriesName, p.EstYear1, p.EstYear3,
       m.ShortName, m.IncomeGroup
  from sq.globalpop as p left join
       sq.globalmetadata as m
  on p.countrycode= m.countrycode;
quit;

title 'Countries with no Metadata';
proc sql;

```

```

select distinct CountryCode
  from work.meta
  where ShortName is null
  order by CountryCode;
quit;
title;

*****;
*Solution with No Temporary Table  *;
*****;
title 'Countries with no Metadata';
title2 'No Temporary Table';
proc sql;
select distinct p.CountryCode
  from sq.globalpop as p left join
       sq.globalmetadata as m
  on p.CountryCode= m.CountryCode
  where ShortName is null;
quit;
title;

```

What is the last **CountryCode** value in your results? **WSM**

5. Using Outer Joins to Summarize Data

```

/*s103s05.sas*/
/*a*/
proc sql;
select c.BankID,
      m.MaritalStatus,
      count(*) as Count format=comma10.
  from sq.customer as c left join
       sq.maritalcode as m
  on c.Married=m.MaritalCode
  where c.BankID is not null
  group by c.BankID, m.MaritalStatus
  order by Count desc;
quit;

/*b*/
title 'Count of Marital Status by Bank';
proc sql;
select c.BankID,
      m.MaritalStatus,
      b.Name,
      count(*) as Count format=comma10.
  from sq.customer as c left join
       sq.maritalcode as m
  on c.Married=m.MaritalCode left join
       sq.bank as b

```

```

    on c.BankID=b.BankID
    where c.BankID is not null
    group by c.BankID, m.MaritalStatus, b.Name
    order by Count desc;
quit;
title;

```

Which combination of **MaritalStatus** and **Name** had the lowest count of customers?
Widowed and Wheatberry Bank, Inc. had a total count of 362.

6. Combining Inner and Outer Joins

```

/*s103s06.sas*/

/*a*/
title 'NYC Merchants with no Transactions';
proc sql;
select a.MerchantName,
       a.MerchantID,
       a.Type 'Merchant Type',
       a.Zip 'Merchant Zipcode'
  from sq.merchant as a left join
       sq.transaction as b
  on a.MerchantID=b.MerchantID
  where a.Zip=10001 and b.MerchantID is null;
quit;
title;

/*b*/
proc sql;
title 'Customers with similar Type Transactions';
title2 'as NYC Merchants with no Sales';
select distinct a.MerchantName,
       a.MerchantID,
       a.type 'Merchant Type',
       a.Zip 'Merchant Zipcode',
       c.CustomerID,
       c.Customername,
       c.Address 'Customer Address'
/*Join merchants with transactions. Notice where clause below
that determines which rows are returned.*/
  from sq.merchant as a left join
       sq.transaction as b
  on a.MerchantID=b.MerchantID
/*Combine the above list with all customers with similar type
transactions.*/
  inner join sq.transactionfull as c
  on a.type=c.type and input(scan(c.address,-1),5.)=a.zip
/*the where filter selects NYC merchants who do not exist in
the transaction table=no sales*/
  where a.zip=10001 and b.MerchantID is null

```

```
order by 2,5;  
quit;  
title;
```

How many potential new customers can the merchant Miasma Mitigation, Inc. now market?
Six potential new customers

End of Solutions

Solutions to Activities and Questions

continued...

3.01 Activity – Correct Answer

2. What note do you see?

NOTE: The execution of this query involves performing one or more Cartesian product joins that can not be optimized.

The default JOIN combines every row in each table. This is called the *Cartesian product*. Typically, the Cartesian product is not the desired result.

9



3.01 Activity – Correct Answer

Partial

FirstName	LastName	State	BankID	Income	AccountID	AccountID	Datetime	BankID	MerchantID	Amount	Services
Gary	Sienkiewicz	NY	101010101	67210.91	1010159565		07MAY18:15:35:02		549940	50.70	Bar
Gary	Sienkiewicz	NY	101010101	67210.91	1010159565	1010159565	16SEP18:14:57:08	101010101	562920	570.55	Fancy Restaurant
Gary	Sienkiewicz	NY	101010101	67210.91	1010159565	1010183063	24FEB18:17:27:42	101010101	542058	23.39	Bar
Gary	Sienkiewicz	NY	101010101	67210.91	1010159565	1010367330	15MAY18:17:54:21	101010101	525576	21.02	Economy
Gary	Sienkiewicz	NY	101010101	67210.91	1010159565	1010367330	17OCT18:11:02:38	101010101	525576	21.02	Economy
Gary	Sienkiewicz	NY	101010101	67210.91	1010159565	1010367364	18OCT18:17:52:51	101010101	549940	37.24	Fast Food
Gary	Sienkiewicz	NY	101010101	67210.91	1010159565	2020012887	23FEB18:09:25:37	202020202	525576	108.22	Economy
Gary	Sienkiewicz	NY	101010101	67210.91	1010159565	3030085224	27JUL18:12:05:48	303030303	525576	26.1	Economy
Gary	Sienkiewicz	NY	101010101	67210.91	1010159565	3030101942	18SEP18:12:13:40	303030303	549940	37.38	Fast Food

Redundant columns

Nonmatching IDs

NOTE: The execution of this query involves performing one or more Cartesian product joins that can not be optimized.

10



continued...

3.02 Activity – Correct Answer

1. What columns can you use to join the tables? **The Name column from the statepopulation table and StateCode column from the statecode table**



Name	PopEstimate1	PopEstimate1	PopEstimate1
AL	4864745	4864745	4864745
AK	741504	741504	741504
AZ	6945452	6945452	6945452
AR	2990410	2990410	2990410
CA	39209127	39209127	39209127



StateCode	StateName
AL	Alabama
AK	Alaska
AZ	Arizona
AR	Arkansas
CA	California

22



3.02 Activity – Correct Answer

3. Complete the ON expression to match rows when **p.Name = s.StateCode**. Highlight and run the query. How many rows are in the new report? **52**

```
select Name, StateName, PopEstimate1, PopEstimate2,
       PopEstimate3
  from sq.statepopulation as p inner join
       sq.statecode as s
  on p.Name = s.StateCode
 order by StateName;
```

Name	StateName	PopEstimate1	PopEstimate2	PopEstimate3
AL	Alabama	4864745	4875120	4887871
AK	Alaska	741504	739786	737438
AZ	Arizona	6945452	7048876	7171646
AR	Arkansas	2990410	2990907	2912902

In SQL, join condition columns do not need to have the same names.



Copyright © SAS Institute Inc. All rights reserved.

23

3.03 Activity – Correct Answer

1. How many tables contain the **BankID** column? **Eight**

Member Name	Column Name
BANK	BankID
CUSTOMER	BankID
SMALLCUSTOMER	BankID
SMALLCUSTOMER2	BankID

contains the full **BANK** information

2. How many tables contain the **MerchantID** column? **Five**

Member Name	Column Name
MERCHANT	MerchantID
SMALLTRANSACTION	MerchantID
SMALLTRANSACTION2	MerchantID
TRANSACTION	MerchantID
TRANSACTIONFULL	MerchantID

contains the full **MERCHANT** information

continued...

3.04 Activity – Correct Answer

- 1.

```
select FirstName, LastName,
       Income format=dollar16., TaxBracket
  from sq.smallcustomer as c inner join
       sq.taxbracket as t
 on c.Income between t.LowIncome and t.HighIncome
 order by TaxBracket desc, Income desc;
```

Use the BETWEEN-AND special where operator.

2. What tax bracket is Olga Comstock in? **12%**

FirstName	LastName	Income	TaxBracket
Sergio	Lefeld	\$86,859	24%
Iva	Bower	\$67,950	22%
Gary	Sienkiewicz	\$67,211	22%
Janet	Sienkiewicz	\$50,112	22%
John	Oliver	\$43,624	22%
Olga	Comstock	\$31,897	12%

3.04 Activity – Correct Answer

3. What note do you see?

```
select FirstName, LastName,
       Income format=dollar16., TaxBracket
  from sq.smallcustomer as c inner join
       sq.taxbracket as t
 where c.Income between t.LowIncome and t.HighIncome
 order by TaxBracket desc, Income desc;
```

NOTE: The execution of this query involves performing one or more Cartesian product joins that can not be optimized.

37



continued...

3.05 Activity – Correct Answer

1. Notice the difference within the **AccountID** columns in rows 8 and 9.

Row	FirstName	LastName	Income	c.AccountID	t.AccountID	DateTime	MerchantID	Amount
1	Gary	Sienkiewicz	67210.91	1010159565	1010159565	16SEP18:14:57:08	568268	107.16
2	Sergio	Lefeld	86859.07	1010367330	1010367330	17OCT18:11:02:38	525576	21.02
3	Sergio	Lefeld	86859.07	1010367330	1010367330	15MAY18:17:54:21	542058	23.39
4	John	Oliver	43623.75	2020012887	2020012887	23FEB18:09:25:37	525576	108.22
5	Iva	Bower	67949.96	3030085224	3030085224	27JUL18:12:05:48	525576	26.1
6	Janet	Sienkiewicz	50111.59	3030101942	3030101942	18SEP18:12:13:40	549940	37.38
7	Olga	Comstock	31896.96	3030165207	3030165207	11MAR18:10:07:14	580881	319.95
8	Ada	Vieyra	29586.44	4040164206	-	-	-	-
9	Samantha	Carney	25476.14	5540174271	-	-	-	-

The left join selects all customers with or without a transaction.

50



3.05 Activity – Correct Answer

No missing AccountID values

Row	FirstName	LastName	Income	c.AccountID	t.AccountID
1	Gary	Sienkiewicz	67210.91	1010159565	1010159565
2	Sergio	Lefeld	86859.07	1010367330	1010367330
3	Sergio	Lefeld	86859.07	1010367330	1010367330
4	John	Oliver	43623.75	2020012887	2020012887
5	Iva	Bower	67949.96	3030085224	3030085224
6	Janet	Sienkiewicz	50111.59	3030101942	3030101942
7	Olga	Comstock	31896.96	3030165207	3030165207
8	Ada	Vieyra	29586.44	4040164206	
9	Samantha	Carney	25476.14	5540174271	

selects AccountID values from the smallcustomer table

Two missing AccountID values

Row	FirstName	LastName	Income	c.AccountID	t.AccountID
1	Gary	Sienkiewicz	67210.91	1010159565	1010159565
2	Sergio	Lefeld	86859.07	1010367330	1010367330
3	Sergio	Lefeld	86859.07	1010367330	1010367330
4	John	Oliver	43623.75	2020012887	2020012887
5	Iva	Bower	67949.96	3030085224	3030085224
6	Janet	Sienkiewicz	50111.59	3030101942	3030101942
7	Olga	Comstock	31896.96	3030165207	3030165207
8	Ada	Vieyra	29586.44		
9	Samantha	Carney	25476.14		

selects AccountID values from the smalltransaction table

51



Copyright © SAS Institute Inc. All rights reserved.

continued...

3.06 Activity – Correct Answer

- Notice that the rows with missing values in **AccountID** have been joined.

FirstName	LastName	Income	c.AccountID	t.AccountID	DateTime	MerchantID
Alejandro	Garcia	86324.38			07MAY18:15:35:02	542058
Alejandro	Garcia	86324.38			09MAY20:12:30:08	549940
Alejandro	Garcia	86324.38			16SEP18:14:57:08	568268
Sai	Nair	51256.02			07MAY18:15:35:02	542058
Sai	Nair	51256.02			09MAY20:12:30:08	549940
Sai	Nair	51256.02			16SEP18:14:57:08	568268
Samantha	Carney	25476.14			07MAY18:15:35:02	542058
Samantha	Carney	25476.14			09MAY20:12:30:08	549940
Samantha	Carney	25476.14			16SEP18:14:57:08	568268
			1010183063	24FEB18:17:27:42		562326

Missing values rows were joined.

61



Copyright © SAS Institute Inc. All rights reserved.

continued...

3.06 Activity – Correct Answer

2. Confirm that missing values were not joined.

```
on c.AccountID = t.AccountID and t.AccountID is not null;
```

FirstName	LastName	Income	c.AccountID	t.AccountID	DateTime	MerchantID
		.	.	.	07MAY18:15:35:02	542058
		.	.	.	09MAY20:12:30:08	549940
		.	.	.	16SEP18:14:57:08	568268
		.	.	1010183063	24FEB18:17:27:42	562326
Sergio	Lefeld	86859.07	1010367330	1010367330	17OCT18:11:02:38	525576
Sergio	Lefeld	86859.07	1010367330	1010367330	15MAY18:17:54:21	542058
		.	.	1010367364	18OCT18:17:52:51	549940
John	Oliver	43623.75	2020012887	2020012887	23FEB18:09:25:37	525576

Do not join missing values.

62



3.06 Activity – Correct Answer

3. How many transactions do not have a customer associated with them?
seven transactions

```
on c.AccountID = t.AccountID and t.AccountID is not missing
where c.AccountID is null;
```

FirstName	LastName	Income	c.AccountID	t.AccountID	DateTime	MerchantID
		.	.	.	07MAY18:15:35:02	542058
		.	.	.	09MAY20:12:30:08	549940
		.	.	.	16SEP18:14:57:08	568268
		.	.	1010183063	24FEB18:17:27:42	562326
		.	.	1010367364	18OCT18:17:52:51	549940
		.	.	3030231909	13FEB18:16:48:05	513178
		.	.	3030231909	27JAN18:13:52:35	536123

Find all transactions without a documented customer.

63



3.07 Activity – Correct Answer

- Did you receive a syntax error?

No syntax error, but no rows were returned from the query.

NOTE: No rows were selected.

- Which **StateName** is *Caberto, Glen Daniel* from? **California**

```
select StateID, CustomerName, StateName
      from sq.transactionfull as t inner join
           sq.statecode as s
      on substr(t.StateID,1,2) = s.StateCode;
```



3.08 Activity – Correct Answer

- What syntax error was generated when you joined columns of different types?

```
select c.CustomerID, c.ZipCode, c.Gender,
       z.Zip, z.City, z.StateCode
  from customerzip as c inner join
       sashelp.zipcode as z
  on c.ZipCode = z.Zip;
```

Character ZIP code

Numeric ZIP code

ERROR: Expression using equals (=) has components that are of different data types.



3.09 Activity – Correct Answer

3. What city does the **ZipCode** value 14216 represent? **Buffalo**

```
select c.CustomerID, c.ZipCode, c.Gender,
       z.Zip, z.City, z.StateCode
  from customerzip as c inner join
       sashelp.zipcode as z
      on c.ZipCode = put(z.Zip,z5.);
quit;
```

CustomerID	ZipCode	Gender	The 5-digit ZIP Code	Name of city/org	Two-letter abbrev. for state name.
2	04429	M	04429	Holden	ME
5	14216	M	14216	Buffalo	NY
1	14580	M	14580	Webster	NY
4	27510	M	27510	Cary	NC

The **Zip** column converts to character in the ON clause.



Sas

Lesson 4 Subqueries

4.1 Subquery in WHERE and HAVING Clauses	4-3
Demonstration: Subquery That Returns a Single Value.....	4-7
Demonstration: Subquery That Returns Multiple Values.....	4-12
Practice.....	4-19
4.2 In-Line Views (Query in the FROM Clause).....	4-25
Demonstration: Using an In-Line View	4-29
Practice.....	4-39
4.3 Subquery in the SELECT Clause	4-43
Demonstration: Remerging Summary Statistics.....	4-46
Practice.....	4-51
4.4 Solutions	4-54
Solutions to Practices	4-54
Solutions to Activities and Questions.....	4-61

4.1 Subquery in WHERE and HAVING Clauses

Subquery in the WHERE and HAVING Clauses

Outer Query

```
SELECT col-name, col-name
FROM input-table
WHERE column operator (SELECT col-name
                        FROM input-table...);
```

Subquery

```
(select avg(PopEstimate1)
   from sq.statepopulation
  where ... ) ...
```



Sas

A subquery in the WHERE and HAVING clauses

- returns values to be used in the outer query's WHERE or HAVING clause
- must return only a single column
- can return multiple values or a single value.

Noncorrelated Subqueries

Outer Query

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

Independent

Subquery

```
(select avg(PopEstimate1)
  from sq.statepopulation
 where ... ) ...
```

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

4

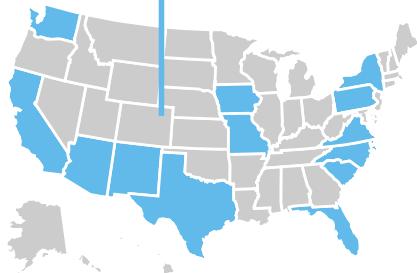


This course focuses on noncorrelated subqueries and refers to them as simply *subqueries* for ease.

Scenario

State PopEstimate1

Total avg(PopEstimate1)



Create a report that displays states with **PopEstimate1** values greater than the *average* **PopEstimate1** value of all states.



5



Subqueries Steps

1

Calculate the subquery's value.

2

Use the value from the subquery in the outer query.

3

Combine the subquery with the outer query.

statepopulation

Name	PopEstimate1
AL	4864745
AK	741504
AZ	6945452
AR	2990410
CA	39209127
CO	5540921

6



Solution without a Subquery

```
1 proc sql;
select avg(PopEstimate1) as Average
  from sq.statepopulation;
quit;
```

Average
6278420

```
2 proc sql;
select Name, PopEstimate1
  from sq.statepopulation
 where PopEstimate1 > 6278420;
quit;
```

What happens if
PopEstimate1
changes in the data?



7



Subquery That Returns a Single Value

```
3
select Name, PopEstimate1
      from sq.statepopulation
     where PopEstimate1 > (select avg(PopEstimate1)
                               from sq.statepopulation);
```

The subquery is evaluated first.

The subquery executes *independently* of the outer query.



Sas

8

Copyright © SAS Institute Inc. All rights reserved.

Subquery That Returns a Single Value

```
3
select Name, PopEstimate1
      from sq.statepopulation
     where PopEstimate1 > (6278420);
```

The outer query uses the value returned by the subquery.

Name	PopEstimate1
AZ	6945452
CA	39209127
FL	20629982
GA	10304763
IL	12826895
IN	6633344

9

Copyright © SAS Institute Inc. All rights reserved.

Sas



Subquery That Returns a Single Value

Scenario

Use a noncorrelated subquery that returns a single value.

Files

- **s104d01.sas**
- **statepopulation** – a SAS table that contains estimated state populations for the next three years

Syntax

```
PROC SQL;
  SELECT col-name, col-name
    FROM input-table
    WHERE column operator (SELECT col-name
                           FROM input-table...)
      ORDER BY col-name <DESC>;
QUIT;
```

Notes

- A subquery is a query expression that is nested as part of another query expression.
- A single-value subquery returns a single row and column.

Demo

1. Open the **s104d01.sas** program in the **demos** folder and find the **Demo** section. Run the query to explore the **sq.statepopulation** table.
2. Run the query in the **Future Subquery** section to find the average of **PopEstimate1** in the **sq.statepopulation** table. View the results.

```
proc sql;
  select avg(PopEstimate1)
    from sq.statepopulation;
quit;
```

3. Complete the outer query to find all states that have a higher **PopEstimate1** value than the result of the previous query's value of 6278420. Sort the results by descending **PopEstimate1**.

```
proc sql;
  select Name, PopEstimate1
    from sq.statepopulation
    where PopEstimate1 > 6278420
    order by PopEstimate1 desc;
quit;
```

4. Remove the value 6278420 in the WHERE clause, replace it with the query that found the average of **PopEstimate1**, and enclose the inner query in parentheses.

Note: Remember to remove the semicolon from the subquery when copying and pasting.

```
proc sql;
select Name, PopEstimate1
  from sq.statepopulation
 where PopEstimate1 > (select avg(PopEstimate1)
                           from sq.statepopulation)
   order by PopEstimate1 desc;
quit;
```

5. Replace the subquery with a new query. Find the average of **Population_2010** from the **sashelp.us_data** table.

Note: The table in a subquery can be a different table than the outer query.

```
proc sql;
select Name, PopEstimate1
  from sq.statepopulation
 where PopEstimate1 > (select avg(Population_2010)
                           from sashelp.us_data)
   order by PopEstimate1 desc;
quit;
```

Name	PopEstimate1
CA	39209127
TX	27937492
FL	20629982
NY	19641589
IL	12826895
PA	12792520

End of Demonstration

4.01 Activity

Open **s104a01.sas** from the **activities** folder and perform the following tasks to use a **subquery** to return two columns in the WHERE clause:

1. Examine and run the first query. Confirm that the results contain one row and two columns.
2. Add the first query as a subquery in the second query to find all states with **PopEstimate1** higher than the average estimated state population.
3. Run the query. What is the syntax error in the log?

Subquery in the HAVING Clause

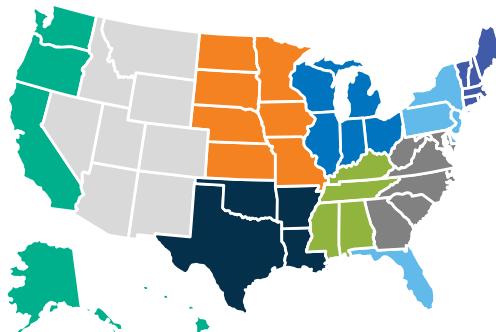
Division `avg(PopEstimate1)`



Total `avg(PopEstimate1)`

sq.statepopulation

Division	PopEstimate1
6	4864745
9	741504
8	6945452
7	2990410
9	39209127
8	5540921
1	3578674
5	949216
5	686575
5	20630000



Subquery in the HAVING Clause

1

```
proc sql;
  select avg(PopEstimate1) as Average
    from sq.statepopulation;
quit;
```

Average
6278420

2

```
proc sql;
  select Division, avg(PopEstimate1) as avgDivisionPop
    from sq.statepopulation
   group by Division
   having avgDivisionPop > 6278420;
quit;
```

15



Copyright © SAS Institute Inc. All rights reserved.

4.02 Activity

Open **s104a02.sas** from the **activities** folder and perform the following tasks to perform a subquery in the HAVING clause:

1. Examine and run the first query. View the results.
2. Modify the second query. Copy the value returned by the first query into the subquery against the HAVING clause to return divisions with an average **PopEstimate1** value greater than the total average of **PopEstimate1**.
3. Remove the static value and add the subquery in the HAVING clause.
4. How many divisions have a higher average **PopEstimate1** than the average **PopEstimate1** of all the states?

16



Copyright © SAS Institute Inc. All rights reserved.

Subquery That Returns Multiple Values

statepopulation Results

Division	Name
3	IL
3	IN
3	MI
3	OH
3	WI

sq.customer

FirstName	MiddleName	LastName	State
Rodney	Matthew	Joyner	WI
Jeanne	Carol	Ballenger	WA
Brian	Dallas	Harper	WI
Thomas	Eric	Henderson	WA
Becky	Danna	Cheers	WI
Alberto	Daryl	Texter	WI
Peter	Douglas	Schmand	WA
Danielle	Julie	Bell	WI
Robert	Javier	Brousseau	WI
Chloe	Leanne	Howell	WA

Create a table of all customers who reside in a state in Division 3.



Sas

18

Copyright © SAS Institute Inc. All rights reserved.

Subquery That Returns Multiple Values

1

```
proc sql;
select Name
  from sq.statepopulation
 where Division = '3';
quit;
```

2

```
proc sql;
create table division3 as
select *
  from sq.customer
 where State in ("IL", "IN", "MI", "OH", "WI");
quit;
```

Name
IL
IN
MI
OH
WI

19

Copyright © SAS Institute Inc. All rights reserved.

Sas



Subquery That Returns Multiple Values

Scenario

Use a noncorrelated subquery that returns multiple values.

Files

- **s104d02.sas**
- **customer** – a SAS table that contains one row per customer
- **statepopulation** – a SAS table that contains estimated state populations for the next three years

Syntax

```
PROC SQL;
  SELECT col-name, col-name
    FROM input-table
    WHERE column IN (SELECT column
      FROM input-table...)
    ORDER BY col-name <DESC>;
QUIT;
```

Notes

- A multiple-value subquery can return more than one value from a column.
- A multiple-value subquery can be used in a WHERE or HAVING expression that contains an IN operator.

Demo

1. Open the **s104d02.sas** program in the **demos** folder and find the **Demo** section.
2. Examine the **Future Subquery** section and run the query to find the states that reside in Division 3. View the results.

```
proc sql;
  select Division, Name
    from sq.statepopulation
    where Division = '3';
quit;
```

3. Using the results from the **Future Subquery** section, modify the outer query by entering the **State** abbreviations ("IL", "IN", "MI", "OH", "WI") in the WHERE clause. Run the query and view the results.

```
proc sql;
  create table Division3 as
  select *
    from sq.customer
    where State in ("IL", "IN", "MI", "OH", "WI");
quit;
```

4. Replace the values in parentheses with the query from step 2 by copying and pasting the query inside the parentheses. The query is now the subquery. Be sure to remove the semicolon from inside the parentheses. Highlight and run the query. Discuss the error in the log.

Note: This program will return a syntax error.

```
...
where State in (select Division, Name
                 from sq.statepopulation
                 where Division = '3');
quit;
```

ERROR: A subquery cannot select more than one column.
 ERROR: A Composite expression (usually a subquery) is used incorrectly in an expression.

5. Remove the **Division** column from the subquery. Highlight and run the query. Discuss the results.

```
...
where State in (select Name
                 from sq.statepopulation
                 where Division = '3');
quit;
```

NOTE: Table WORK.DIVISION3 created, with 16022 rows and 22 columns.

6. Change the **Division** value from **3** to **6** in the subquery. Replace the **3** at the end of the new table name in the CREATE TABLE statement to a **6**. Highlight and run the query. Discuss the results.

```
proc sql;
create table division6 as
select *
  from sq.customer
  where State in (select Name
                  from sq.statepopulation
                  where Division = '3');
quit;
```

NOTE: Table WORK.DIVISION6 created, with 4994 rows and 22 columns.

End of Demonstration

ANY Keyword

WHERE column = ANY(subquery values)

```
proc sql;
create table Division3 as
select *
  from sq.customer
  where State = any(select Name
                     from sq.statepopulation
                     where Division = '3');
quit;
```

Equivalent to
using the **IN**
operator

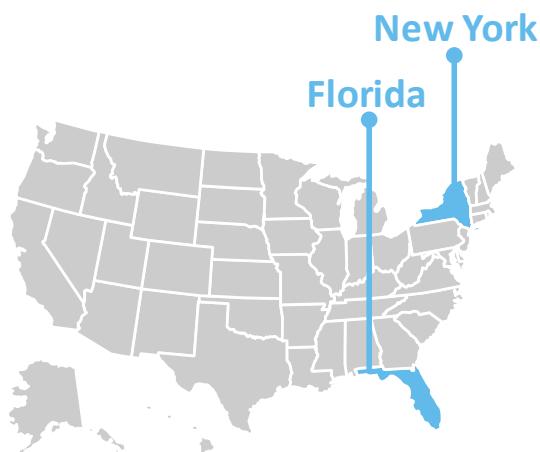


21

Copyright © SAS Institute Inc. All rights reserved.

Similar to the IN operator, you can also use the ANY expression for all returned values from the subquery.

Scenario



Which states have a
PopEstimate1 value
that is **greater** than
New York or Florida?



22

Copyright © SAS Institute Inc. All rights reserved.

ANY Keyword

```
select Name, PopEstimate1
from sq.statepopulation
where PopEstimate1 > any(select PopEstimate1
                           from sq.statepopulation
                           where Name in ("NY", "FL"));
```

Name	PopEstimate1
CA	39209127
FL	20629982
TX	27937492

20629982, 19641589

The ANY keyword is true when the value of the specified column is greater than **any** of the values returned by the subquery.

23



MIN Function

```
select Name, PopEstimate1
from sq.statepopulation
where PopEstimate1 > (select min(PopEstimate1)
                           from sq.statepopulation
                           where Name in ("NY", "FL"));
```

Name	PopEstimate1
CA	39209127
FL	20629982
TX	27937492

19641589

You can also use the MIN function inside the subquery to return the minimum PopEstimate1.

24



You can also use > or < to return values greater than any of the values or less than any of the values.

In this example, you can also use the MIN function inside the subquery to return the minimum value of PopEstimate1 and use a comparison operator to compare the values.

4.03 Activity

Open **s104a03.sas** from the **activities** folder and perform the following tasks to find all states with a **PopEstimate1** value that is lower than the value for New York or Florida:

1. Complete the query using the ANY keyword or MAX statistic.
2. Run the query. How many states have estimated populations lower than New York or Florida?

Correlated Subqueries

Outer Query

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

Dependent

Correlated

```
(SELECT ...
FROM ...
<WHERE ...>)
```

Correlated
subqueries are
resource intensive.



The previous subqueries have been noncorrelated subqueries that are self-contained and that execute independently of the outer query.

A correlated subquery is dependent on the outer query. A correlated subquery requires one or more values to be passed to it by the outer query before the subquery can be resolved. This means that PROC SQL must process the correlated subquery multiple times: once for each table row that the outer query processes. Correlated subqueries tend to use more resources than noncorrelated subqueries. You typically want to avoid a correlated subquery, and we will briefly discuss why.

Correlated Subqueries

```
select count(*) as TotalCustomer
  from sq.customer as c
 where '1' = (select Division
              from sq.statepopulation as a
             where a.Name = c.State);
```

The inner query needs information from the **outer query**.

How many customers are from a **state** in **Division 1**?



Sas

28

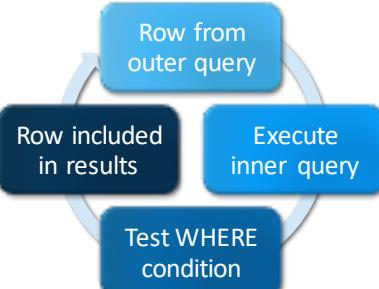
Copyright © SAS Institute Inc. All rights reserved.

A correlated subquery is not stand-alone. It needs additional information from the main query.

Correlated Subqueries

```
select count(*) as TotalCustomer
  from sq.customer as c
 where '1' = (select Division
              from sq.statepopulation as s
             where s.Name = c.State);
```

Executing a correlated subquery can be resource intensive.



Each subquery is executed once for **every row** of the outer query.



Sas

29

Copyright © SAS Institute Inc. All rights reserved.

Conceptually, correlated subqueries are pretty tricky and resource intensive. Using correlated subqueries reduces performance in your system. A better method is joining the tables and using a WHERE clause to obtain the necessary rows.

Using Joins

```
select count(*) as TotalCustomer
  from sq.customer as c
 where '1' = (select Division
               from sq.statepopulation as s
              where s.Name = c.State);
```

TotalCustomer
3292

```
select count(*) as TotalCustomer
  from sq.customer as c inner join
       sq.statepopulation as p
      on p.Name = c.State
 where Division = '1';
```

TotalCustomer
3292

A more concise method
is to use a join.

Copyright © SAS Institute Inc. All rights reserved.



Subqueries can sometimes be replaced with joins. In this example, instead of the tricky and resource-intensive correlated subquery, you can use a join to retrieve the results.

The join joins the **sq.customer** and **sq.statepopulation** tables on **State** abbreviations. Then the query filters for rows where **Division** equals 1.

Syntax Summary

```
SELECT col-name, col-name
  FROM input-table
 WHERE column operator (SELECT col-name
                         FROM input-table...)
 GROUP BY col-name
 HAVING column operator (SELECT col-name
                          FROM input-table...);
```

WHERE and HAVING Subqueries



```
WHERE column IN (subquery values)
WHERE column = ANY(subquery values)
WHERE column > ANY(subquery values)
```

Multiple Values Returned by a Subquery

Copyright © SAS Institute Inc. All rights reserved.





Practice

Level 1

1. Subquery That Returns a Single Value

The **sq.statepopulation** table contains estimated population statistics for every state in the US and its territories. Which states have an estimated three-year population growth greater than the average for all states?

- Write a query that displays the average three-year population growth for all states. Use the **nPopChg3** column to calculate the average from the **sq.statepopulation** table.

Results

36355.1

- Use the query from step a to display the states that have a projected three-year population growth greater than the overall average.
 - Include **Name** and **nPopChg3** in the results.
 - Label **nPopChg3** as **Estimated Growth** and format the values with commas.
 - Use the query from step a to subset the table.
 - Order the results by descending **nPopChg3**.
 - Add an appropriate title to the report.

Partial Results

States with an estimated 3-Year Population Growth Greater than the Overall Average

Name	Estimated Growth
TX	379,128
FL	322,513
CA	157,696
AZ	122,770
NC	112,820
WA	110,159
GA	106,420
CO	79,662
SC	62,002

- Which state has the lowest estimated growth your results?

Level 2**2. Subquery with Multiple Functions**

Determine which Texas customers have higher than average credit scores.

- Using the **sq.customer** table, write a query to define the high-credit threshold as 2 standard deviations greater than the mean of **CreditScore** for TX customers.
 - Use the following expression to create the column **HighScore**. This will be your future subquery result.

```
sum(avg(creditscore), (2*std(creditscore)))
```

- Run the query and compare your results.

Results

HighScore
765.0846

- Write a second query to create a report showing Texas customers whose **CreditScore** value is greater than the results from the previous query. Use the query from part a as a subquery.
 - Select **CustomerID**, **FirstName**, **LastName**, and **CreditScore**.
 - Subset the table for customers who are from the state of Texas and who have a greater **CreditScore** value than the value from the query in part a.
 - Order the report by descending **CreditScore**.
 - Add an appropriate title.

Partial Results

Texas Customers with Higher than Average Credit Scores			
Customer ID	First Name	Last Name	CreditScore
1915019345	Christopher	Miras	848
1921584392	Christopher	Kiddy	845
1965260689	Morgan	Tamanaha	843
1940349233	Ruth	Poloskey	835
1988972911	Bea	Holzwarth	832
1955883210	Victor	Galway	831
1966934182	Joseph	Malacara	827
101E02AE1E	Tripp	Dolland	824

- In your report, what is the lowest value for **CreditScore**?

3. Subquery Using Different Tables

Create a report that lists all countries and their forecasted percentage of *Borrowed for health or medical purposes (% age 15+)* that reside in Europe or Central Asia and are considered high income.

- Using the **sq.globalmetadata** table, return all the country codes for the region of Europe & Central Asia that are in the *High Income* group.

Partial Report

CountryCode
AUT
BEL
CHE
CYP
CZE

- Write the second query using the **sq.globalindex** table and view the estimated percentage of population who borrowed for health or medical purposes (% age 15+) for the region of Europe & Central Asia.
 - Select **CountryCode** and create two new columns that convert **EstYear1** and **EstYear3** to percentages.
 - Calculate the columns by dividing **EstYear1** and **EstYear3** by 100. Name the new columns **EstPct1** and **EstPct3** respectively and format them using the percent format.
 - Subset the table where **IndicatorName** is equal to *Borrowed for health or medical purposes (% age 15+)*, and use the previous query to subset the data by **CountryCode** values in Europe and Central Asia.

Hint: You can use the **UPCASE** function to standardize character casing.

- Order the report by descending **EstYear1**.
- Add an appropriate title.

Partial Report

**Estimated Forecasted Percentages
Countries in Europe & Central Asia with High Incomes
Borrowed Money for Health or Medical Purposes**

CountryCode	EstPct1	EstPct3
LTU	8.10%	15.0%
CYP	7.11%	9.44%
LUX	6.49%	5.91%
LVA	6.39%	4.34%
PRT	5.91%	3.88%
ITA	5.82%	1.34%
AUT	5.24%	5.50%
ESP	5.08%	4.59%
IDI	4.86%	7.46%

- c. Which country in your report has the lowest **EstPct1** value for next year?

4. Using a Subquery with Summarized Data

Determine which customers spend less on average utility bills than the overall utility bill average?

- a. Use the **sq.transactionfull** table to create a report showing **CustomerID** and the average utility payment amount, named **UtilityAmt**, for customers whose average utility payment is less than the overall average utility payment for all customers.

Hint: Use the **Type** column to determine which transactions are utility payments.

- 1) Order by descending **UtilityAmt**.
- 2) Add an appropriate title to the report.

Results

Customers with Lower than Average Utility Payments

CustomerID	UtilityAmt
1990559364	163.8138
1950882733	158.0746
1929444655	132.0638
1978669535	122.6131
1970082571	33.01538

- b. How many customers in the **sq.transaction** table have lower than average utility payments?

Challenge

5. Using Nested Subqueries

Create a report that lists all countries that have a higher population than the average of all countries and where the **EstYear1** forecast for *Outstanding housing loan (% age 15+)* increases in **EstYear3**.

- Open **s104p05.sas** from the **practices** folder. Run the query to create the table **CountryEstPop**. The table contains total estimated population for ages 15+ for each country.
- Write a query to find all countries in the **CountryEstPop** table with a higher population than the mean of all countries.

Partial Results

CountryCode
BGD
BRA
CHN
COD
COL
DEU
EGY

- Use the previous query as a subquery. Select all columns from the **sq.globalindex** table where **IndicatorName** is *Outstanding housing loan (% age 15+)*, the **EstYear1** forecast increases in three years, **EstYear1** and **EstYear3** are not null, and the country has a population estimate greater than the mean of all countries.
 - Convert **EstYear1** and **EstYear3** to a percent and format the values. Create a new column to determine the percent increase by subtracting **EstYear3** by **EstYear1**. Name the column **PctIncrease** and format the values using a percent.
 - Order the results by descending **PctIncrease**. Add an appropriate title.

Results

Countries with a Higher Population Estimate than the Mean Number of Outstanding Housing Loan Increases from Year1 to Year3				
CountryCode	IndicatorName	EstYear1	EstYear3	PctIncrease
BGD	Outstanding housing loan (% age 15+)	2.26%	10.2%	7.90%
KOR	Outstanding housing loan (% age 15+)	18.9%	26.1%	7.21%
IRN	Outstanding housing loan (% age 15+)	21.2%	26.7%	5.54%
ITA	Outstanding housing loan (% age 15+)	11.2%	16.7%	5.43%
CHN	Outstanding housing loan (% age 15+)	8.49%	12.4%	3.92%
IND	Outstanding housing loan (% age 15+)	25.0%	20.5%	2.50%

- d. Which country had the largest value for **PctIncrease**?

End of Practices

4.2 In-Line Views (Query in the FROM Clause)

In-Line View

Outer Query

```
SELECT ...
  FROM (SELECT col-name
           FROM ...
         <WHERE ...>
<WHERE ...>
<GROUP BY ...>
<HAVING ...>
<ORDER BY ...>;
```

An in-line view acts as a *virtual table*.



34

In-Line View

```
proc sql;
select *
  from (select CustomerID, State, Income
           from sq.customer
         where Income > 100000);
quit;
```

Creates a virtualtable to use in the outer query.

You *cannot* use an ORDER BY clause in an in-line view.



35

Here are some characteristics of in-line views:

- An in-line view is not assigned a permanent name, although it can take an alias.
- An in-line view can be referred to only in the query in which it is defined. It cannot be referenced in another query.
- You cannot use an ORDER BY clause in an in-line view.
- The names of columns in an in-line view can be assigned in the column list of that in-line view. This syntax can be useful for renaming columns.
- In order to visually separate an in-line view from the rest of the query, you can enclose the in-line view in any number of pairs of parentheses. If you specify an alias for the in-line view, then the alias specification must appear outside the outermost pair of parentheses for that in-line view.

Scenario

State	TotalCustomer	EstimateBase	PctCustomer
VT	47	625744	.008%
WV	217	1853001	.012%
ME	159	1328369	.012%
DE	110	897934	.012%
MD	768	5773798	.013%
SC	745	4625381	.016%
NH	234	1316464	.018%
PA	2405	12702873	.019%
HI	263	1360307	.019%
GA	1912	9688709	.020%

Create a report that shows the percentage of customers in each state based on each state's estimated population.



Using Temporary Tables

```
create table totalcustomer as
select State, count(*) as TotalCustomer
from sq.customer
group by State;
```

Count the number of customers in each state.

1

State	TotalCustomer
AK	289
AL	1337
AR	760
AZ	3185
CA	18134
CO	1920
CT	755

37



Using Temporary Tables

```
select c.State,
       c.TotalCustomer,
       s.EstimateBase,
       c.TotalCustomer/s.EstimateBase as
          PctCustomer format=percent7.3
  from totalcustomer as c inner join
       sq.statepopulation as s
  on c.State = s.Name
  order by PctCustomer;
```

Join with the sq.statepopulation table.

2

State	TotalCustomer
AK	289
AL	1337
AR	760
AZ	3185
CA	18134
CO	1920
CT	755

Name	EstimateBase
AL	4780138
AK	710249
AZ	6392288
AR	2916028
CA	37254523
CO	5029316
CT	3574147
DE	897934
DC	601766

38



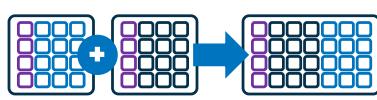
Using Temporary Tables

1



Create the
totalcustomer table.

2



Join **totalcustomer** with
sq.statepopulation.

What if the **customer** table is updated daily
and we always want the
most **recent results**?



Sas



Using an In-Line View

Scenario

Use an in-line view to create a virtual table.

Files

- **s104d03.sas**
- **customer** – a SAS table that contains one row per customer
- **statepopulation** – a SAS table that contains estimated state populations for the next three years

Syntax

```
PROC SQL;
  SELECT col-name, col-name
    FROM (SELECT column, ...
           FROM input-table...)
      WHERE expression
      ORDER BY col-name <DESC>;
QUIT;
```

Notes

- An in-line view is a query in the FROM clause.
- An in-line view produces a virtual table that the outer query uses to select data.
- An in-line view can be referenced only in the query in which it is defined.

Demo

1. Open the **s104d03.sas** program in the **demos** folder and find the **Demo** section. Run the first query to explore the **statepopulation** and **customer** tables.
2. Move to the next section, **Temporary Table Solution**. Discuss both queries in the section.
 - a. Run the first query to create the **totalcustomer** temporary table. View the results.
 - b. Run the second query to join the **totalcustomer** and **sq.statepopulation** tables and calculate the new column **PctCustomer** that calculates the percentage of customers in each state based on the current year's estimated population. View the results.
3. Copy only the query that is used to create the **totalcustomer** table. Move to the **Using an In-Line View** section of the program. Paste the query (not including the CREATE TABLE statement) in the FROM clause to create an in-line view. Be sure to remove the semicolon. Highlight and run the query. View the syntax error in the log.

```

...
from (select State, count(*) as TotalCustomer
      from sq.customer
      group by State
      order by TotalCustomer desc) as c inner join
      sq.statepopulation as s
      on c.State = s.Name
...

```

```

49          order by TotalCustomer desc;
22
76
ERROR 22-322: Syntax error, expecting one of the following: !, !!, &, (, ), *, **, +, ',', -, '.', /, <, <=, <>, =, >, >=, ?, AND,
BETWEEN, CONTAINS, EQ, EQT, EXCEPT, GE, GET, GT, GTT, HAVING, IN, INTERSECT, IS, LE, LET, LIKE, LT, LTT, NE, NET,
NOT, NOTIN, OR, OUTER, UNION, ^, ^=, |, ||, ~, ~=.

ERROR 76-322: Syntax error, statement will be ignored.

NOTE: PROC SQL set option NOEXEC and will continue to check the syntax of statements.
49          ) as c inner join
180
ERROR 180-322: Statement is not valid or it is used out of proper order.

```

4. Remove the ORDER BY clause in the subquery. Run the query and view the results.

```

...
from (select State, count(*) as TotalCustomer
      from sq.customer
      group by State
      order by TotalCustomer desc) as c inner join
      sq.statepopulation as s
      on c.State = s.Name
...

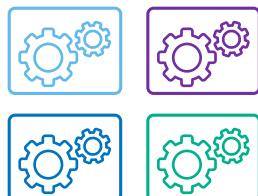
```

State	TotalCustomer	EstimateBase	PctCustomer
VT	47	625744	.008%
WV	217	1853001	.012%
ME	159	1328369	.012%
DE	110	897934	.012%
MD	768	5773798	.013%
SC	745	4625381	.016%
MI	224	1216164	.019%

End of Demonstration

Storing an In-Line View

```
...
from (select State, count(*) as TotalCustomer
      from sq.customer
      group by State)
...
;
```



How can you **store** the in-line view to use in other queries?



Sas

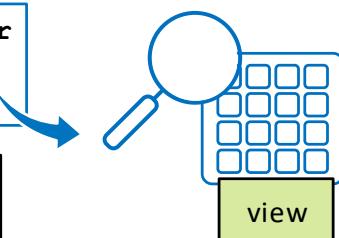
41

Copyright © SAS Institute Inc. All rights reserved.

Creating a View

```
select State, count(*) as TotalCustomer
      from sq.customer
      group by State);
```

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



42

Copyright © SAS Institute Inc. All rights reserved.

Sas

CREATE VIEW Statement

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

```
proc sql;
create view sq.totalcustomer as
select State, count(*) as TotalCustomer
from sq.customer
group by State;
quit;
```

The **query** is stored as a permanent view in the **sq** library.

43

Copyright © SAS Institute Inc. All rights reserved.

For sake of efficiency, it is recommended that you avoid using the ORDER BY clause in a query that defines a view. Using the ORDER BY clause in a view definition forces PROC SQL to sort the data every time that the view is referenced. Instead, you can use an ORDER BY clause in queries that reference the view.

Using a VIEW

sq.totalcustomer

State	TotalCustomer
AK	289
AL	1337
AR	760
AZ	3185

```
select c.State, c.TotalCustomer, s.EstimateBase,
c.TotalCustomer/s.EstimateBase as
PctCustomer format=percent7.3
from (select State, count(*) as TotalCustomer
from sq.customer
group by State) as c inner join
sq.statepopulation as s
on c.State = s.Name
order by PctCustomer;
```

You can use a view **in place** of the in-line view.

44

Copyright © SAS Institute Inc. All rights reserved.

Using a VIEW

sq.totalcustomer

State	TotalCustomer
AK	289
AL	1337
AR	60
AZ	

```
select c.State, c.TotalCustomer, s.EstimateBase,
       c.TotalCustomer/s.EstimateBase as
           PctCustomer format=percent7.3
  from sq.totalcustomer as c inner join
       sq.statepopulation as s
      on c.State = s.Name
  order by PctCustomer;
```

The view **executes** the stored query and extracts the **most current data**.



4.04 Activity

Open **s104a04.sas** from the **activities** folder and perform the following tasks to create and use a view:

1. Create a view named **VWtotalcustomer** from the query. Run the query and examine the log.
2. Run the code in the section **Use the View in the PROCS Below**. Which state has the highest number of customers?

Location of a PROC SQL View

```
proc sql;
create view sq.totalcustomer as
select State, count(*) as TotalCustomer
from customer
group by State;
quit;
```

PROC SQL expects the view to reside in the **same** SAS library as the contributing table or tables.


S: \workshop\data

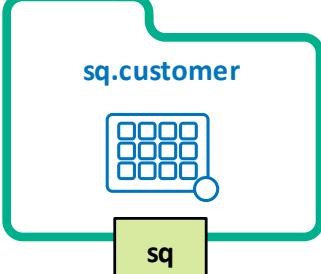

sq

48

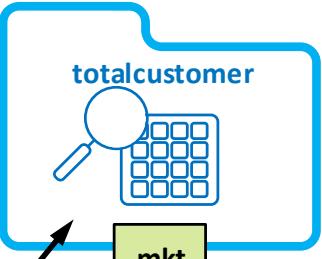
Copyright © SAS Institute Inc. All rights reserved.



Scenario

S: \workshop\data

sq



S: \workshop

mkt

Members of the marketing team moved the view from the **sq** library to their working location.



49

Copyright © SAS Institute Inc. All rights reserved.



Scenario

```
libname mkt "s:/workshop";

proc sql;
select *
  from mkt.totalcustomer
quit;
```

ERROR: File MKT.CUSTOMER.DATA does not exist.

Marketing executes a query using the view but receives this error. Why?

50

Copyright © SAS Institute Inc. All rights reserved.

Sas

Exploring the Problem

```
proc sql;
create view sq.totalcustomer as
select State, count(*) as TotalCustomer
  from customer
 group by State;
quit;
```

The view assumes that the **customer** data is in the **s:\workshop** library.

51

Copyright © SAS Institute Inc. All rights reserved.

Sas

This violated the one-level naming convention in the CREATE VIEW query. When marketing used the view, it retrieved the stored query. The stored query that we are using uses the one-level naming convention. When the view was created, PROC SQL assumed that the customer data was in the **sq** library. When marketing moved the view and executed a query using the view, the stored query assumed that the **customer** table is now in the **mkt** library in **s:\workshop**. However, the **customer** table is not in that location.

Making a View Portable



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

```
libname mkt "s:/workshop";
proc sql;
create view mkt.totalcustomer as
select State, count(*) as TotalCustomer
from sq.customer
group by State
using libname sq 's:/workshop/data';
quit;
```

The scope of the libref is local to the view and does not conflict with any identically named librefs in the SAS session.

52

Copyright © SAS Institute Inc. All rights reserved.



By using a SAS enhancement, you can create a usable view that is stored in a different physical location than its source tables. In other words, you can make the view portable.

You can embed a SAS LIBNAME statement or a SAS/ACCESS LIBNAME statement in a view by using the USING LIBNAME clause. When PROC SQL executes the view, the stored query assigns the libref. For SAS/ACCESS librefs, PROC SQL establishes a connection to a DBMS. The scope of the libref is local to the view and does not conflict with any identically named librefs in the SAS session. When the query finishes, the libref is disassociated. The connection to the DBMS is terminated, and all data in the library becomes unavailable.

Views

Advantages

- avoid storing copies of large tables
- avoid a frequent refresh of table copies; when the underlying data changes, a view surfaces the most current data
- combine data from multiple database tables and multiple libraries or databases
- simplify complex queries
- prevent other users from inadvertently altering the query code

53

Copyright © SAS Institute Inc. All rights reserved.

Views

Disadvantages

- Views might produce different results each time they are accessed if the data in the underlying data sources changes.
- Views can require significant resources each time that they execute. With a view, you save disk storage space at the cost of extra CPU and memory usage.

54

Copyright © SAS Institute Inc. All rights reserved.

Syntax Summary

```
SELECT ...
  FROM (SELECT col-name
        FROM ...
       <WHERE ...>);
```

In-Line View



```
CREATE VIEW table-name AS query
```

CREATE VIEW

```
CREATE VIEW ...
  USING LIBNAME libref engine "path";
```

USING Clause

Copyright © SAS Institute Inc. All rights reserved.





Practice

Level 1

6. In-Line View Summarizing CreditScore

Determine which customers have an extremely high credit score relative to other customers in their ZIP code (**Zip**). Similar to the practice in the previous section, extremely high credit is defined as greater than 2 standard deviations above the mean of **CreditScore**. However, rather than use an overall high-credit threshold for all customers, we need to calculate the threshold for each value of **Zip**. This can be accomplished using an in-line view.

- Open **s104p06.sas** from the **practices** folder. Run the query to summarize the **HighZipCredit** threshold for each **Zip** value for the first 1000 rows.

Partial Results

High Credit Threshold by Zipcode	
Zip	HighZipCredit
6050	753.6741
6101	727.6
6320	640
6340	637
6360	783.7965
6418	690

- Use the query from step a as an in-line view to join with the **sq.customer** table.
 - Select **c.CustomerID**, **c.Zip**, and **c.CreditScore** from the **sq.customer** table, and select **s.HighZipCredit** from the in-line view. Format the **c.Zip** column using the Z5. format.
 - Perform an inner join with the **sq.customer** table and the in-line view from step a. Give the **sq.customer** table the alias **c** and the in-line view the alias **s**. Remove the INOBS= option from the in-line view.
 - Use **c.Zip = s.Zip** as the join criteria.
 - Filter rows where the customer's **c.CreditScore** value is greater than the **s.HighZipCredit** value.
 - Order the results by **Zip** and descending **CreditScore**.
 - Add an appropriate title to the report.

Partial Results

Customers with Extremely High Credit for their Zipcode			
Customer ID	Zip	CreditScore	HighZipCredit
1942378776	00501	812	780.2733
1974119755	00501	793	780.2733
1991947460	01027	798	786.9846
1666368801	01085	845	791.1248
1916975658	01101	847	794.4782
1992140250	01101	813	794.4782
1934252655	01101	797	794.4782
1001755040	01750	000	702.7740

- c. What is the last **Zip** value and the corresponding **CreditScore** value in your final report?

Level 2**7. Building a Complex Query with In-Line Views**

Determine which employees have the highest salary for their job title in every state. Using the **sq.employee** table, write a query to calculate the maximum **Salary** value for each value of **JobTitle** within each state.

- a. Using the **sq.employee** table, write a query to calculate the maximum salary for each job title within each state.
- 1) Convert the values of the **State** column to uppercase and name the column **State** to standardize the state code values. Select **JobTitle** and calculate the maximum salary. Name the column **MaxJobSalary**.
 - 2) Filter rows where the **State** is not null.
 - 3) Group the results by **State** and **JobTitle**.
 - 4) Order the results by **State**.
 - 5) Add an appropriate title.
 - 6) Run the query and compare your results.

Partial Results

Maximum Salary for Each Job in every State		
State	JobTitle	MaxJobSalary
CA	BI Specialist II	44425
CA	Temp. Sales Rep.	26885
CA	Concession Consultant II	32195
CA	Sales Rep. I	27410
CA	Warehouse Assistant II	26690
CA	Training Manager	48335
CA	Auditor II	45100

- b. Using the query in step a as an in-line view, create a report by joining that result with the **sq.employee** table to do the following:
- 1) Display **EmployeeID**, **EmployeeName**, **State**, **JobTitle**, and **Salary** for the highest paid employee for each value of **JobTitle** in every state.
 - 2) Perform an inner join with the **sq.employee** table and the in-line view from step a. Give the **sq.employee** table the alias **detail** and the in-line view the alias **summary**.
- Hint: Remove the ORDER BY clause when using an in-line view.
- 3) Use **JobTitle**, **State**, and **Salary** equal to **MaxJobSalary** as the join criteria.
 - 4) Order the report by **State** and **JobTitle**.
 - 5) Add an appropriate title and format the **Salary** values with a dollar sign and comma.

Partial Results

Employees with Highest Salary for their Job in every State				
EmployeeID	EmployeeName	State	JobTitle	Salary
120759	Apr, Nishan	CA	Accountant II	\$36,230
120757	Knopfmacher, Paul	CA	Accountant III	\$38,545
120803	Droste, Victor	CA	Applications Developer I	\$43,630
120764	Worton, Steven	CA	Auditor I	\$40,450
120763	Capps, Ramond	CA	Auditor II	\$45,100
120808	Dunbar, Marcell	CA	PLS Specialist II	\$44,125

- c. Who is the last employee in the final report?

Challenge

8. Building a Complex Query Using a Join and Subquery

Generate a report of the total estimated number of individuals next year who made or received digital payments in the past year (% age 15+) in South Asia to determine the best country to promote a digital payment app.

- a. Calculate the total estimated population for ages 15+ for countries in South Asia in the **sq.globalpop** table.
 - 1) Select the **CountryCode**, sum of **EstYear1**. Name the new column **EstYear1Pop** and format using commas.
 - 2) Filter the **SeriesName** column for estimated population greater than 15 years of age. Use a subquery from the **sq.globalmetadata** table to include only **CountryCode** in South Asia.

Hint: Use the LIKE operator in the WHERE clause to include rows for ages 15+.

 - 3) Group by **CountryCode**.
 - 4) Run the query and compare your results.

Results

Estimated Population for Next Year in South Asia Ages 15+	
CountryCode	EstYear1Pop
AFG	17,922,774
BGD	111,653,576
BTN	559,366
IND	917,227,653
LKA	15,621,067
NPL	18,861,095
PAK	120,225,516

- b. Use the previous query as an in-line view to join by **CountryCode** and multiply the estimated percentage of individuals who made or received digital payments in the past year (% age 15+) by the estimated population for next year.
- 1) Select the **CountryCode** and **IndicatorName** columns from the **sq.globalindex** table. Convert the **EstYear1** value from **sq.globalindex** to a percentage by dividing by 100 and then multiply by the **EstYear1Pop** value from the in-line view. Name the column **Estimate** and format using commas.
- Note:** Multiplying the **EstYear1** percentage of individuals by the **EstYear1Pop** total population for the country returns an estimated number of individuals who used digital payments.
- 2) Perform an inner join of **sq.globalindex** and the in-line view from step a. Assign **sq.globalindex** the alias **f**, and assign the in-line view the alias **pop**. Use **CountryCode** as the join criteria.
 - 3) Filter the rows by the **IndicatorName** value of *Made or received digital payments in the past year (% age 15+)*.
 - 4) Order the results by **Estimate** descending. Add an appropriate title.

Results

Estimated Population who Made or Received Digital Payments in the Past Year South Asia Ages 15+		
CountryCode	IndicatorName	Estimate
IND	Made or received digital payments in the past year (% age 15+)	177,123,495
PAK	Made or received digital payments in the past year (% age 15+)	9,323,844
BGD	Made or received digital payments in the past year (% age 15+)	8,275,363
LKA	Made or received digital payments in the past year (% age 15+)	3,254,101
NPL	Made or received digital payments in the past year (% age 15+)	1,777,532
AFG	Made or received digital payments in the past year (% age 15+)	998,380
BTN	Made or received digital payments in the past year (% age 15+)	96,220

- c. Which value of **CountryCode** has the highest estimated use of digital payments?

End of Practices

4.3 Subquery in the SELECT Clause

Subquery in the SELECT Clause

Outer Query

```
SELECT col-name, (SELECT col-name
                   FROM ...
                   <WHERE ...>
                   FROM ...
                   <WHERE ...>
                   <GROUP BY ...>
                   <HAVING ...>
                   <ORDER BY ...>;
```

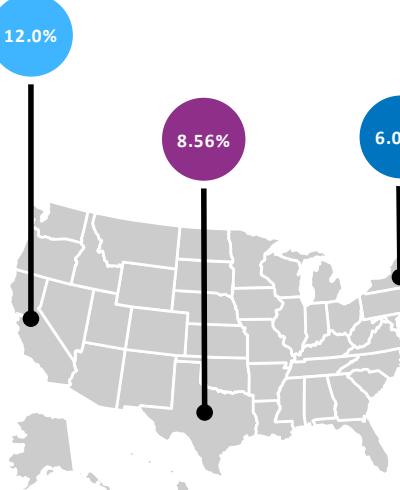
A subquery in the SELECT clause can return a *single value* to the outer query.



58

Copyright © SAS Institute Inc. All rights reserved.

Scenario



State
PopEstimate1

Total
sum(PopEstimate1)

Create results that show next year's estimated percentage of population of each state.



59

Copyright © SAS Institute Inc. All rights reserved.

Subquery in the SELECT Clause



```
select Name, PopEstimate1 / (select sum(PopEstimate1)
                               from sq.statepopulation)
                                as PctPop format=percent7.2
   from sq.statepopulation
  order by PctPop desc;
```

Name	PctPop
CA	12.0%
TX	8.56%
FL	6.32%
NY	6.02%
"	3.02%

State
PopEstimate1



326477837

PopEstimate1 from
each state

total estimated
population for the US

60



Copyright © SAS Institute Inc. All rights reserved.

When using a subquery in the SELECT clause, you do not have to use a value from the table in the outer query. You can retrieve values from other tables.

Remerging Summary Statistics

selects each name and
PopEstimate1 value from
each row

summarizes the *entire*
column



The remerge
feature of SAS
makes **two
passes** through a
table.



```
select Name, PopEstimate1 / sum(PopEstimate1)
      as PctPop format=percent7.2
   from sq.statepopulation
  order by PctPop desc;
```

Name	PctPop
CA	12.0%
TX	8.56%
FL	6.32%
NY	6.02%
"	3.02%

61



Copyright © SAS Institute Inc. All rights reserved.

Aggregate functions, such as the SUM function, can cause the same calculation to repeat for every row. This occurs whenever PROC SQL remerges data. Remerging occurs whenever any of the following conditions exist:

- The SELECT clause references a column that contains an aggregate function and other columns that are not listed in the GROUP BY clause.
- The ORDER BY clause references a column that is not referenced by the SELECT clause.

When a query remerges data, PROC SQL displays a note in the log to indicate that data remerging has occurred. PROC SQL runs an internal query to find the sum and then runs another internal query to divide each state's population by the sum.



Remerging Summary Statistics

Scenario

Remerge summary statistics in SAS to find the percentage population of each state.

Files

- **s104d04.sas**
- **statepopulation** – a SAS table that contains estimated state populations for the next three years

Syntax

```
PROC SQL;
  SELECT col-name, summary function(column)
    FROM table;
  QUIT;
```

Notes

- The SELECT clause references a column that contains an aggregate function and other columns that are not listed in the GROUP BY clause.

Demo

1. Open the **s104d04.sas** program in the **demos** folder and find the **Demo** section. Run the query to select the **Name** and **PopEstimate1** columns from the **sq.statepopulation** table.

```
proc sql;
  select Name, PopEstimate1
    from sq.statepopulation;
quit;
```

2. In the SELECT clause, add the SUM function to sum **PopEstimate1**. Format the column using the COMMA12 format. Run the query and examine log and results.

Note: All values are the sum of the entire **PopEstimate1** column, whereas the **Name** and **PopEstimate1** columns have individual rows from the input table.

```
proc sql;
  select Name, PopEstimate1, sum(PopEstimate1) format=comma12.
    from sq.statepopulation;
quit;
```

NOTE: The query requires remerging summary statistics back with the original data.

3. Modify the SELECT clause by dividing **PopEstimate1** and **sum(PopEstimate1)**. Replace the COMMA12. format with the PERCENT7.2 format. Name the new column **PctPop**. Run the query and examine the log and results.

```
proc sql;
  select Name,
    PopEstimate1/sum(PopEstimate1) as PctPop format=percent7.2
```

```
from sq.statepopulation;
quit;
```

4. Add an ORDER BY clause and sort the results by descending PctPop. Run the query and examine the results.

```
proc sql;
select Name,
       PopEstimate1/sum(PopEstimate1) as PctPop format=percent7.2
  from sq.statepopulation
 order by PctPop desc;
quit;
```

Name	PctPop
CA	12.0%
TX	8.56%
FL	6.32%
NY	6.02%
IL	3.93%
PA	3.02%

End of Demonstration

Remerging Summary Statistics

```
select Region,
       sum(PopEstimate1) as TotalRegion format=comma14.
  from sq.statepopulation;
```

Specifying a column and a summary function
remerges the data and does **not** return the
desired result.

Region	TotalRegion
3	326,477,837
4	326,477,837
4	326,477,837
3	326,477,837
4	326,477,837
4	326,477,837
4	326,477,837



Remerging can be a powerful tool. However, it does not always produce the desired result. The most common example is when you forget the GROUP BY clause in a query and specify a grouping column and a summary function. Always check your log to help avoid this error and others.

Disabling the Remerging of Summary Statistics

```
PROC SQL NOR MERGE;
```

```
proc sql noremerge;
select Region,
       sum(PopEstimate1) as TotalRegion format=comma14 .
     from sq.statepopulation;
quit;
```

ERROR: The query requires remerging summary statistics back with the original data. This is disallowed due to the NOR MERGE proc option or NOSQLREMERGE system option.

64



Resubmitting the query with the NOR MERGE option in the PROC SQL statement produces no output and results in an error message in the SAS log.

4.05 Activity

Open **s104a05.sas** from the **activities** folder and perform the following tasks to disable remerging of summary statistics:

1. Examine and run the query. Examine the log and the results. What note do you see in the log?
2. Add the PROC SQL option NOR MERGE. Run the query. Did it run successfully? What was the error in the log?
3. Add a GROUP BY clause after the FROM clause and group by **Region**. Run the query. Did it run successfully?

65



Scenario

Name	PopEstimate1	TotalRegionEst	PctRegion	Region
NY	19641589	56,058,789	35.0%	1
PA	12783538	56,058,789	22.8%	1
NJ	8874516	56,058,789	15.8%	1
MA	6826022	56,058,789	12.2%	1
CT	3578674	56,058,789	6.4%	1
NH	1342373	56,058,789	2.4%	1
ME	1331370	56,058,789	2.4%	1
RI	1057063	56,058,789	1.9%	1
VT	623644	56,058,789	1.1%	1
IL	12826895	67,996,917	18.9%	2
OH	11635003	67,996,917	17.1%	2
MI	9951890	67,996,917	14.6%	2
IN	6633344	67,996,917	9.8%	2
MO	6087203	67,996,917	9.0%	2

How can we calculate the percentage of population in each *region*?



Remerging GROUP BY Summary Statistics

```

proc sql;
select Name, PopEstimate1,
       sum(PopEstimate1) as TotalRegionEst format=comma14.,
       PopEstimate1/calculated TotalRegionEst as
                           PctRegion format=percent7.1,
       Region
  from sq.statepopulation
 group by Region
 order by Region, PctRegion desc;
quit;

```

SAS remerges the sum of each region.

Name	PopEstimate1	TotalRegionEst	PctRegion	Region
NY	19641589	56,058,789	35.0%	1
PA	12783538	56,058,789	22.8%	1
NJ	8874516	56,058,789	15.8%	1
MA	6826022	56,058,789	12.2%	1
CT	3578674	56,058,789	6.4%	1
NH	1342373	56,058,789	2.4%	1
ME	1331370	56,058,789	2.4%	1
RI	1057063	56,058,789	1.9%	1
VT	623644	56,058,789	1.1%	1
IL	12826895	67,996,917	18.9%	2
OH	11635003	67,996,917	17.1%	2
MI	9951890	67,996,917	14.6%	2
IN	6633344	67,996,917	9.8%	2
MO	6087203	67,996,917	9.0%	2

Syntax Summary

```
SELECT col-name, (SELECT col-name  
FROM ...  
<WHERE ...>)  
FROM table1;
```

Subquery in the SELECT Clause



PROC SQL NOREMERGE;

PROC SQL Option

```
SELECT col-name, summary function(column)  
FROM table1;
```

Remerging Summary Statistics

Beyond SQL Essentials

What if you want to ...

... learn about using a reflexive join with a subquery?

- View the SAS paper [Nifty Uses of SQL Reflexive Join and Sub-query in SAS.](#)

... learn about building subqueries in SAS Enterprise Guide?

- Read the SAS blog [Building an SQL subquery in SAS Enterprise Guide.](#)

... review examples using subqueries to select data?

- Visit [Using Subqueries to Select Data](#) in the SAS documentation.



Practice

Level 1

9. Remerging Summary Statistics

Determine which states have the most estimated births for next year. Include a column showing each state's births as the percent of national births.

- Select the **Name** and **Births1** columns. Create a new column named **PctBirth** by dividing **Births1** for each state by the sum of **Births1** for all states. Format the new column using the PERCENT format.
- Order the results by descending **PctBirth**.
- Add an appropriate title.

Partial Results

Estimate Percentage of Births by Each State		
Name	Births1	PctBirth
CA	490384	12.3%
TX	400968	10.0%
NY	235826	5.91%
FL	224643	5.63%
IL	156299	3.91%
PA	140256	3.51%

- Which state has the highest percentage of estimated births for next year?

Level 2

10. Subquery in the SELECT Clause with an In-Line View

Find the top 10 countries that have the highest percentage of estimated global population using the population estimates of ages 15+ in the **sq.globalfull** table.

- Use the **sq.globalfull** table to write a query to sum the **EstYear1Pop** of all countries.
 - Use an in-line view to select the distinct **CountryCode** and **EstYear1Pop** values from the **sq.globalfull** table.

Note: You must find the **distinct** estimated population of each country because the data contains the estimated population for each country multiple times.
 - Sum the **EstYear1Pop** column and name the column new column **EstPct**. Format the column using the COMMA format.
 - Run the query and compare your results.

Results

Estimated Population of Ages 15+ for Next Year	
EstPct	
5,271,101,653	

- b. Create a new query using the query from step a in the SELECT clause to determine the estimated global population percentage of each country.
- 1) Select the distinct **CountryCode** and **ShortName** values. Determine the percentage population by creating a new column. Divide the **EstYear1Pop** value of each country by the total value calculated in step a. Name the new column **PctPop** and format using the PERCENT format.
 - 2) Use the **sq.globalfull** table.
 - 3) Order the results by **PctPop** descending.
 - 4) Limit the results to the **top 10 countries**.
 - 5) Add an appropriate title.

Results

Top 10 Countries by Estimate Population Ages 15+		
CountryCode	ShortName	PctPop
CHN	China	21.3%
IND	India	17.4%
USA	United States	4.87%
IDN	Indonesia	3.48%
BRA	Brazil	2.98%
PAK	Pakistan	2.28%
RUS	Russia	2.28%
BGD	Bangladesh	2.12%
JPN	Japan	2.10%
NGA	Nigeria	1.87%

- c. Which country has the highest estimated population of individuals 15 or over?

Challenge

11. Remerging GROUP BY Summary Statistics

In the **sq.statepopulation** table, states in the United States are categorized by divisions. Calculate the percentage of births for next year by each state for its division.

- a. Use the **sq.statepopulation** table to write a query to sum the values of **Births1** for each division and divide the summarized value by the **Births1** values of each state.

- 1) Select the **Name**, **Division**, and **Births1** columns. Create two new calculated columns.
 - a) The first column should sum all **Births1** values. Name the column **TotalDivisionEst**, and format it using the COMMA format.
 - b) The second column should divide **Births1** by the new column, **TotalDivisionEst**. Name the column **PctDivision**, and format it using the PERCENT format.
- 2) Group the query by **Division**.
- 3) Order the results by **Division** and **PctDivision** descending.
- 4) Format the **Births1** column using the COMMA format.
- 5) Add an appropriate title.

Partial Results

Percentage of Estimated Births by Each Division				
Name	Division	Births1	TotalDivisionEst	PctDivision
MA	1	71,418	149,056	47.9%
CT	1	35,892	149,056	24.1%
ME	1	12,652	149,056	8.5%
NH	1	12,358	149,056	8.3%
RI	1	10,902	149,056	7.3%
VT	1	5,834	149,056	3.9%
NY	2	235,826	478,983	49.2%
PA	2	140,256	478,983	29.3%
NJ	2	102,901	478,983	21.5%

- b. Which state has the highest percentage of estimated births in Division 8?

End of Practices

4.4 Solutions

Solutions to Practices

1. Subquery That Returns a Single Value

```
/*s104s01.sas*/
/*a*/
proc sql;
select mean(nPopChg3)
  from sq.statepopulation;
quit;

/*b*/
title "States with an estimated 3-Year Population Growth";
title2 "Greater than the Overall Average";
proc sql;
select Name, nPopChg3 label="Estimated Growth" format=commal6.
  from sq.statepopulation
  where nPopChg3 > (select mean(nPopChg3)
                      from sq.statepopulation)
        order by nPopChg3 desc;
quit;
title;
```

Which state has the lowest estimated growth in your results? MA, with an estimated growth of 38,903

2. Subquery with Multiple Functions

```
/*s104s02.sas*/
/*a*/
proc sql;
select sum(avg(CreditScore), (2*std(CreditScore))) as HighScore
  from sq.customer
  where state='TX';
quit;

/*b*/
title 'Texas Customers with Higher than Average Credit Scores';
proc sql;
select CustomerID, FirstName, LastName, CreditScore
  from sq.customer
  where state="TX" and
        CreditScore >
          (select sum(avg(CreditScore), (2*std(CreditScore)))
           as HighScore
            from sq.customer
            where state='TX')
```

```

    order by CreditScore desc;
quit;
title;

```

In your report, what is the lowest value for **CreditScore**? 766

3. Subquery Using Different Tables

```

/*s104s03.sas*/

/*a*/
proc sql;
select CountryCode
  from sq.globalmetadata
 where upcase(Region)='EUROPE & CENTRAL ASIA' and
       upcase(IncomeGroup)='HIGH INCOME';
quit;

/*b*/
title 'Estimated Forecasted Percentages';
title2 'Countries in Europe & Central Asia with High Incomes';
title3 'Borrowed Money for Health or Medical Purposes';
proc sql;
select CountryCode,
       EstYear1/100 as EstPct1 format=percent7.2,
       EstYear3/100 as EstPct3 format=percent7.2
  from sq.globalindex
 where IndicatorName='Borrowed for health or medical purposes
      (% age 15+)' and
       CountryCode in
          (select CountryCode
             from sq.globalmetadata
            where upcase(Region)='EUROPE & CENTRAL ASIA'
                  and upcase(IncomeGroup)='HIGH INCOME')
       order by EstYear1 desc;
quit;
title;

```

Which country in your report has the lowest **EstPct1** value for next year? FIN

4. Using a Subquery with Summarized Data

```

/*s104s04.sas*/

title "Customers with Lower than Average Utility Payments";
proc sql;
select CustomerID, avg(Amount) as UtilityAmt
  from sq.transactionfull
 where Type="Utilities"
 group by 1
 having UtilityAmt <
        (select avg(Amount)
           from sq.transactionfull

```

```

        where Type="Utilities")
    order by UtilityAmt desc;
quit;
title;

```

How many customers in the **sq.transaction** table have lower than average utility payments? **Five**

5. Using Nested Subqueries

```

title "Countries with a Higher Population Estimate than the Mean";
title2 "Number of Outstanding Housing Loan Increases from Year1 to
      Year3";

/*b*/
proc sql;
select CountryCode
  from CountryEstPop
 where EstPop > (select mean(EstPop)
                  from CountryEstPop);
quit;

/*c*/
proc sql;
select CountryCode, IndicatorName,
       EstYear1/100 as EstYear1 format=percent7.2,
       EstYear3/100 as EstYear3 format=percent7.2,
       calculated EstYear3 - calculated EstYear1
              as PctIncrease format=percent7.2
  from sq.globalindex
 where IndicatorName = "Outstanding housing loan (% age 15+)"
   and EstYear1 is not null
   and EstYear3 is not null
   and EstYear1 < EstYear3
   and CountryCode in
      (select CountryCode
       from CountryEstPop
       where EstPop > (select mean(EstPop)
                         from CountryEstPop))
 order by PctIncrease desc;
quit;
title;

```

Which country had the largest value for **PctIncrease**? **BGD**

6. In-Line View Summarizing CreditScore

```
/*s104s06.sas*/

title 'Customers with Extremely High Credit for their Zipcode';
proc sql;
select c.CustomerID, c.Zip format=z5., c.CreditScore,
      s.HighZipCredit
  from sq.customer as c inner join
       (select Zip,
              sum(avg(CreditScore), (2*std(CreditScore)))
                  as HighZipCredit
         from sq.customer
        where CreditScore is not null
        group by Zip) as s
  on c.Zip=s.Zip
  where c.CreditScore>s.HighZipCredit
  order by Zip, CreditScore desc;
quit;
title;
```

What is the last **Zip** value and the corresponding **CreditScore** value in your final report?
The last Zip value is 99701 with a CreditScore value of 815.

7. Building a Complex Query with In-Line Views

```
/*s104s06.sas*/

/*a*/
title 'Maximum Salary for Each Job in every State';
proc sql;
select upcase(State) as State, JobTitle,
      max(Salary) as MaxJobSalary
  from sq.employee
  where State is not null
  group by State, JobTitle
  order by State;
quit;
title;

/*b*/
title 'Employees with Highest Salary for their Job in every State';
proc sql;
select detail.EmployeeID, detail.EmployeeName, detail.State,
      detail.JobTitle, detail.Salary format=dollar12.
  from sq.employee as detail inner join
       (select upcase(State) as State,
              JobTitle, max(Salary) as MaxJobSalary
         from sq.employee
        where State is not null
        group by State, JobTitle) as summary
  on detail.JobTitle=summary.JobTitle and
```

```

      detail.State=Summary.State and
      detail.Salary=Summary.MaxJobSalary
      order by detail.State, detail.JobTitle;
quit;
title;

```

Who is the last employee in the final report? The last employee is Pongor, Katherine.

8. Building a Complex Query Using a Join and Subquery

```

/*s104s07.sas*/
/*a*/
title 'Estimated Population for Next Year in South Asia';
title2 'Ages 15+';
proc sql;
select CountryCode,
       sum(EstYear1) as EstYear1Pop format=comma14.
  from sq.globalpop
 where SeriesName not like '%00-04' and
       SeriesName not like '%05-09' and
       SeriesName not like '%10-14' and
       CountryCode in (select CountryCode
                         from sq.globalmetadata
                         where Region = 'South Asia')
 group by CountryCode;
quit;
title;

/*b*/
title 'Estimated Population who Made or Received Digital Payments
      in the Past Year';
title2 'South Asia Ages 15+';
proc sql;
select f.CountryCode, f.IndicatorName,
       ((f.EstYear1/100) * pop.EstYear1Pop)
       as Estimate format=comma14.
  from sq.globalindex as f inner join
       (select CountryCode, sum(EstYear1)
            as EstYear1Pop format=comma14.
  from sq.globalpop
 where SeriesName not like '%00-04' and
       SeriesName not like '%05-09' and
       SeriesName not like '%10-14' and
       CountryCode in (select CountryCode
                         from sq.globalmetadata
                         where Region = 'South Asia')
 group by CountryCode) as pop
 on f.CountryCode = pop.CountryCode
 where IndicatorName = 'Made or received digital payments in
      the past year (% age 15+)'

```

```
order by Estimate desc;
quit;
```

Which value of **CountryCode** has the highest estimated use of digital payments?
IND has an estimated digital payment use of 177,123,495 people.

9. Remerging Summary Statistics

```
title 'Estimate Percentage of Births by Each State';
proc sql;
select Name, Births1,
       Births1/sum(Births1) as PctBirth format=percent7.2
  from sq.statepopulation
  order by PctBirth desc;
quit;
title;

/*Alternate Solution*/
title 'Estimate Percentage of Births by Each State';
proc sql;
select Name, Births1,
       Births1/(select sum(Births1)
                  from sq.statepopulation)
              as PctBirth format=percent7.2
  from sq.statepopulation
  order by PctBirth desc;
quit;
title;
```

Which state has the highest percentage of estimated births for next year? **CA (California)**

10. Subquery in the SELECT Clause with an In-Line View

```
/*s104s10.sas*/
/*a*/
title 'Estimated Population of Ages 15+ for Next Year';
proc sql;
select sum(EstYear1Pop) as EstPct format=comma16.
      from (select distinct CountryCode, EstYear1Pop
            from sq.globalfull);
quit;
title;

/*b*/
title 'Top 10 Countries by Estimate Population';
title2 'Ages 15+';
proc sql outobs=10;
select distinct CountryCode, ShortName,
      EstYear1Pop/
      (select sum(EstYear1Pop) as EstPct format=comma16.
       from (select distinct CountryCode, EstYear1Pop
             from sq.globalfull))
```

```
as PctPop format=percent7.2
from sq.globalfull
order by PctPop desc;
quit;
title;
```

Which country has the highest estimated population of individuals 15 or over? **China**

11. Remerging GROUP BY Summary Statistics

```
/*s104s11.sas*/
title "Percentage of Estimated Births by Each Division";
proc sql;
select Name, Division, Births1 format=comma14.,
       sum(Births1) as TotalDivisionEst format=comma14.,
       Births1/calculated TotalDivisionEst
              as PctDivision format=percent7.1
from sq.statepopulation
group by Division
order by Division, PctDivision desc;
quit;
title;
```

Which state has the highest percentage of estimated births in Division 8?

AZ with 27.7% of estimated births

End of Solutions

Solutions to Activities and Questions

continued...

4.01 Activity – Correct Answer

2.

```
...
where PopEstimate1 > (select avg(PopEstimate1),
                           "Average Estimated Population"
                        from sq.statepopulation);
```

3. Run the query. What is the syntax error in the log?

ERROR: A subquery cannot select more than one column.
 ERROR: A Composite expression (usually a subquery) is used incorrectly in an expression.

12



Copyright © SAS Institute Inc. All rights reserved.

4.01 Activity – Correct Answer

```
...
where PopEstimate1 > (select avg(PopEstimate1),
                           'Average Population'
                        from sq.statepopulation);
```

A subquery must return values in a **single column**.



13

6278420	Average Population
---------	--------------------



Copyright © SAS Institute Inc. All rights reserved.

4.02 Activity – Correct Answer

4. How many divisions have a higher average **PopEstimate1** than the average **PopEstimate1** of all the states? **Five divisions**

```
select Division, avg(PopEstimate1) as avgDivisionPop
  from sq.statepopulation
 group by Division
 having avgDivisionPop > (select avg(PopEstimate1)
                            from sq.statepopulation);
```

Division	avgDivisionPop
2	13766548
3	9364018
5	7103557
7	9883222
9	10552964

Subquery

17



Copyright © SAS Institute Inc. All rights reserved.

4.03 Activity – Correct Answer

```
select Name, PopEstimate1
  from sq.statepopulation
 where PopEstimate1 < any (select PopEstimate1
                            from sq.statepopulation
                           where Name in ("NY", "FL"));
```

```
select Name, PopEstimate1
  from sq.statepopulation
 where PopEstimate1 < (select max(PopEstimate1)
                            from sq.statepopulation
                           where Name in ("NY", "FL"));
```

49 states

26

46	WV	1830929
47	WI	5772958
48	WY	584290
49	PR	3406495



Copyright © SAS Institute Inc. All rights reserved.

4.04 Activity – Correct Answer

1. Create a view named **VWtotalcustomer** from the query. Run the query and examine the log.

```
create view VWtotalcustomer as
select State, count(*) as TotalCustomer
from sq.customer
group by State;
```

NOTE: SQL view WORK.VWTOTALCUSTOMER has been defined.

2. Run the code in the section **Use the View in the PROCS Below**. Which state has the highest number of customers? **California**

47



continued...

4.05 Activity – Correct Answer

1. What note do you see in the log?

NOTE: The query requires remerging summary statistics back with the original data.

2. Did it run successfully? What was the error in the log? **No. The NOREMERGE option disables remerging of summary statistics.**

ERROR: The query requires remerging summary statistics back with the original data. This is disallowed due to the NOREMERGE proc option or NOSQLREMERGE system option.

66



4.05 Activity – Correct Answer

3. Add a GROUP BY clause after the FROM clause and group by **Region**. Run the query. Did it run successfully? **Yes. The data is not remerged when you use the GROUP BY clause.**

```
proc sql noremerge;
  select Region,
    sum(PopEstimate1) as TotalRegion format=comma14.
  from sq.statepopulation
  group by region;
quit;
```

Region	TotalRegion
1	56,058,789
2	67,996,917
3	122,401,186
4	76,614,450
X	3,406,495

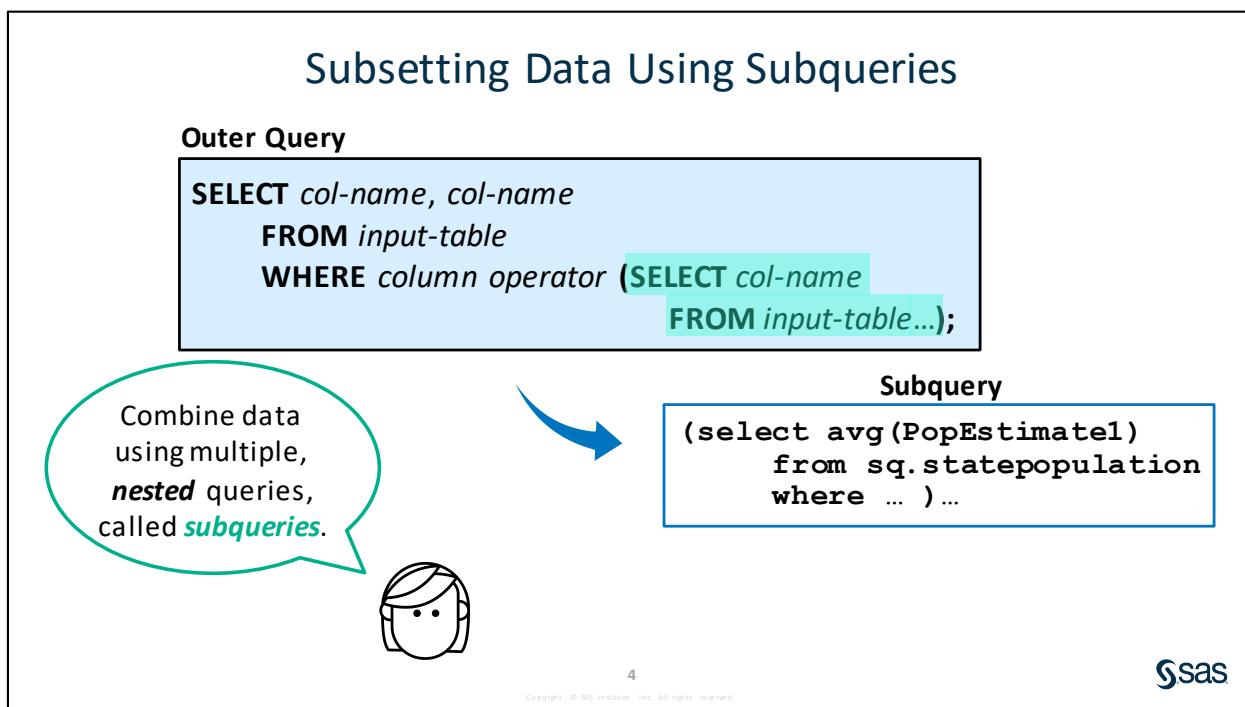
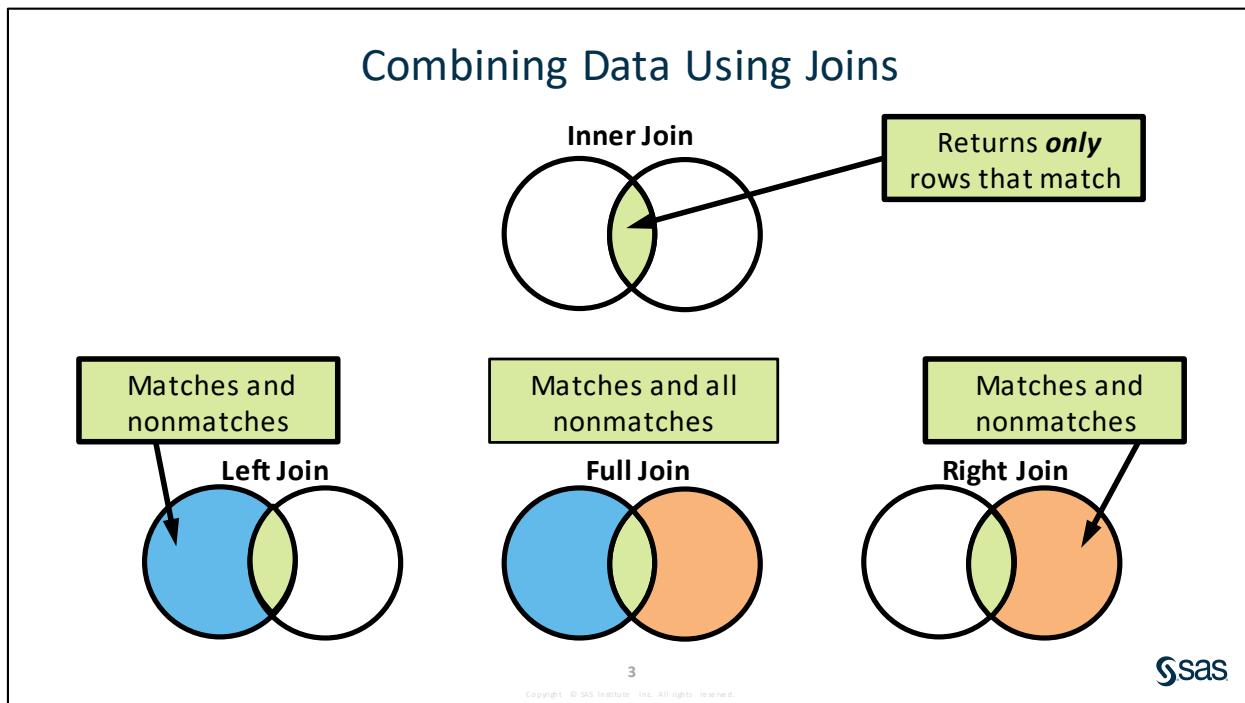
67
Copyright © SAS Institute Inc. All rights reserved.

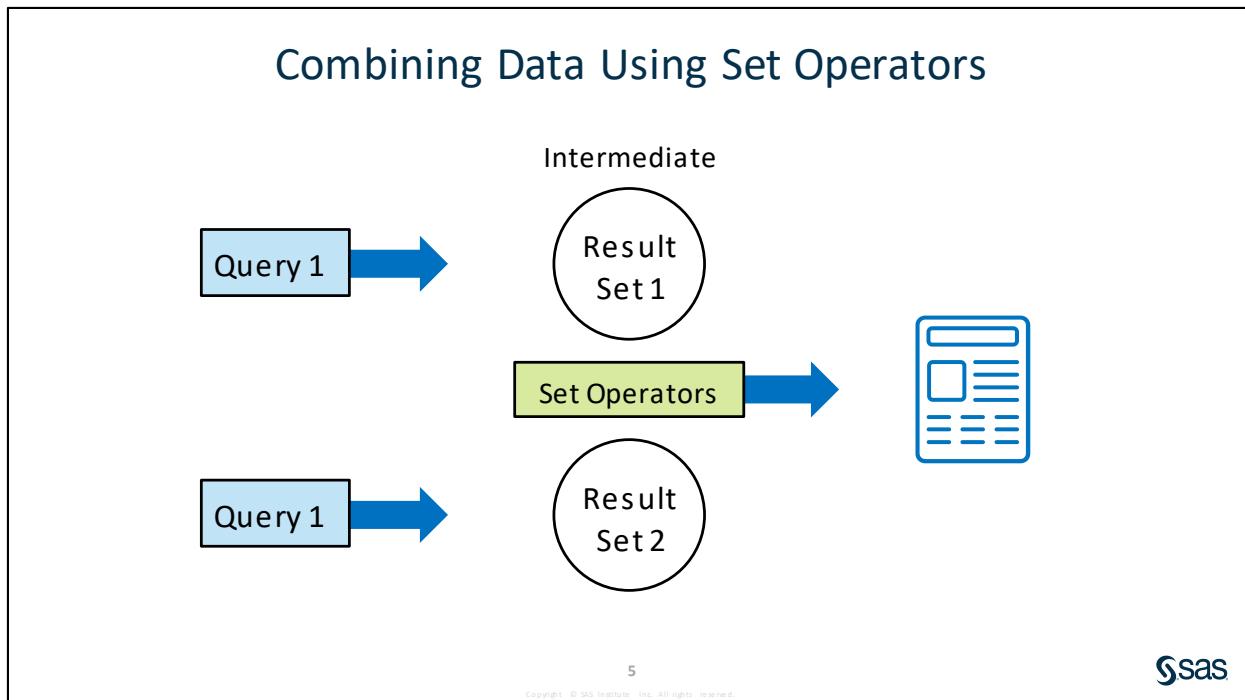


Lesson 5 Set Operators

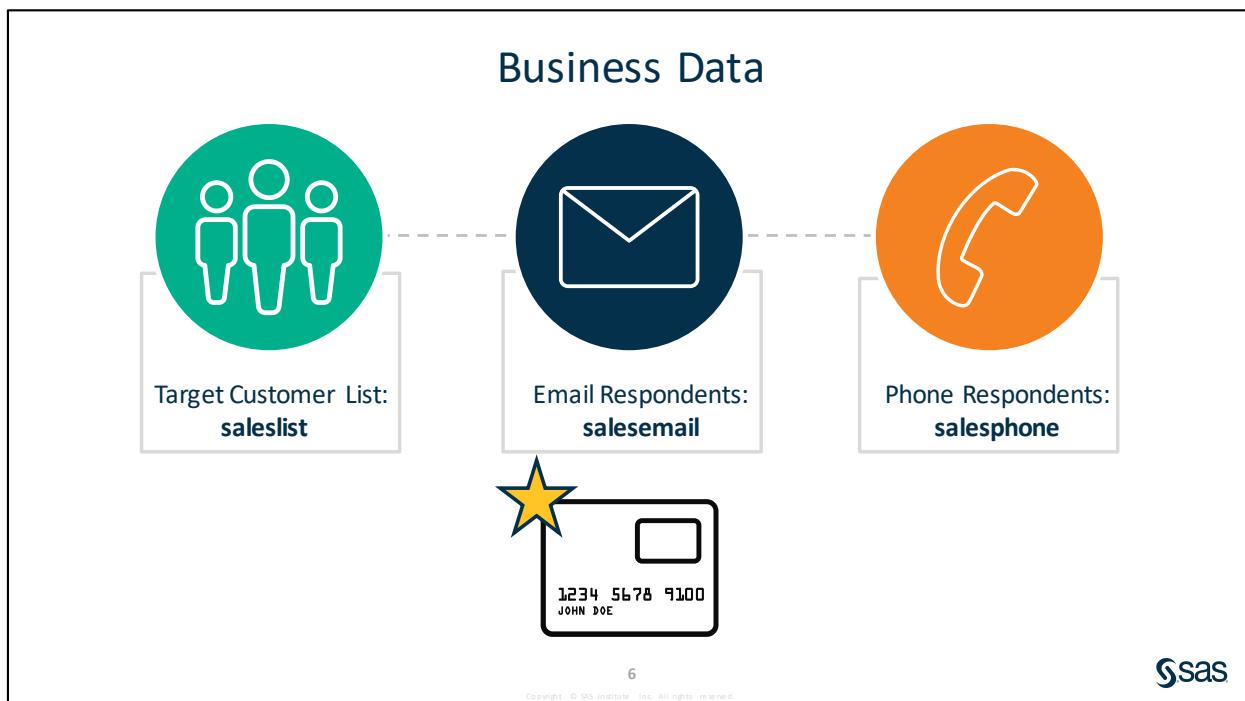
5.1	Introduction to Set Operators	5-3
5.2	INTERSECT, EXCEPT, and UNION	5-9
	Demonstration: Using the UNION Set Operator.....	5-18
	Practice.....	5-24
5.3	OUTER UNION	5-28
	Demonstration: Using the OUTER UNION Set Operator.....	5-30
	Practice.....	5-34
5.4	Solutions	5-36
	Solutions to Practices	5-36
	Solutions to Activities and Questions.....	5-39

5.1 Introduction to Set Operators





A set operator vertically combines the intermediate result sets from two queries to produce a final result set.



Scenario

Which customers have responded to both **email** and **phone** requests?

Which customers have not responded to the **email** request?

Which customers responded to either **phone** or **email**?

Complete list of all customer **responses**

7

Copyright © SAS Institute Inc. All rights reserved.



Target Customer List Table

saleslist

User ID	Customer ID	Cell Phone	Home Phone
marremartinez6531@ismissing.com	1939774314	(630)7734430	(630)1642888
kimlihuffman843@fakeemail.com	1958716829	(212)2023592	
heacamoredock827@invalid.com	1999302252	(212)6871017	(212)1618048
heljaboone8613@invalid.com	1963960449		(212)7089847
jambeerskin7521@fakeemail.com	1987175132	(337)2330976	(337)0712345
aranihale6320@ismissing.com	1970095442	(805)2412974	(805)1234567
terlowhite687@invalid.com	1908694347	(904)2003480	(904)1234567
barmablanton521@n/a.com	1918638906	(860)8721645	(860)1234567
... 8 ...	1908694347	(860)8721645	(860)1234567

List of **target customers** and their contact information.

8

Copyright © SAS Institute Inc. All rights reserved.



Response Tables



Email Respondents



Phone Respondents



Customer responses by *email* and *phone*.

salesemail	
CustomerID	EmailResp
1939774314	Accepted
1999302252	Declined
1963960449	Declined
1908694347	Accepted
1000011449	Accepted

salesphone		
CustomerID	SalesRep	PhoneResp
1939774314	121038	Declined
1999302252	120145	Call Back
1999302252	120145	Accepted
1963960449	120145	Declined
1000011449	120145	Declined

9

Copyright © SAS Institute Inc. All rights reserved.



Scenario

saleslist			
UserID	CustomerID	CellPhone	HomePhone
marremartinez6531@ismissing.com	1939774314	(630)7734430	(630)1642888
kimlihoffman843@fakeemail.com	1958716829	(212)2023592	
heacamoredock827@invalid.com	1999302252	(212)6871017	(212)1618048
heljabone8613@invalid.com	1963960449	(212)7089847	
jambeerskin7521@fakeemail.com	1987175132	(337)2330976	(337)0369173
aranihale6320@ismissing.com	1970095442	(805)2412974	(805)0046155
terlowwhite687@invalid.com	1908694347	(904)2003480	(904)4874841
barmablantron521@n/a.com	1918638906	(860)8721645	(860)7883034
... 8 more rows ...	1000011449	(210)10000000	

All tables contain **CustomerID**.

salesemail		
CustomerID	EmailResp	
1939774314	Accepted	
1999302252	Declined	
1963960449	Declined	
1908694347	Accepted	
1000011449	Accepted	

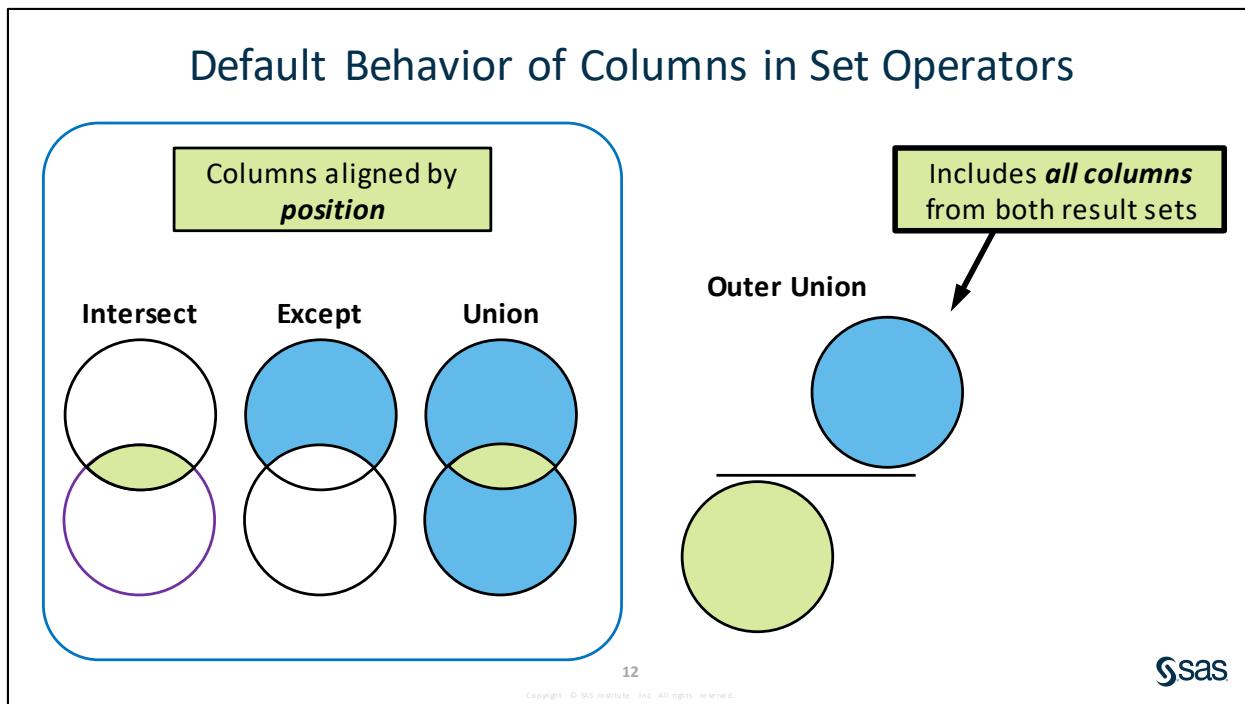
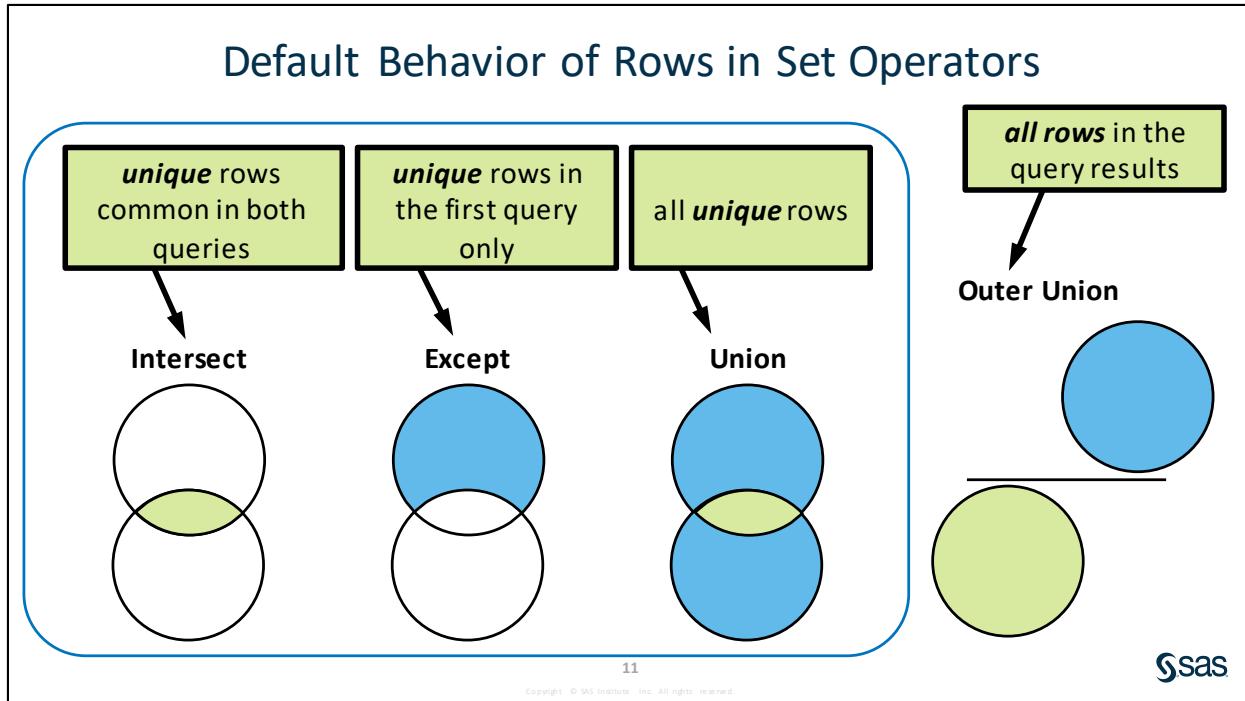
salesphone		
CustomerID	SalesRep	PhoneResp
1939774314	121038	Declined
1999302252	120145	Call Back
1999302252	120145	Accepted
1963960449	120145	Declined
1000011449	120145	Declined

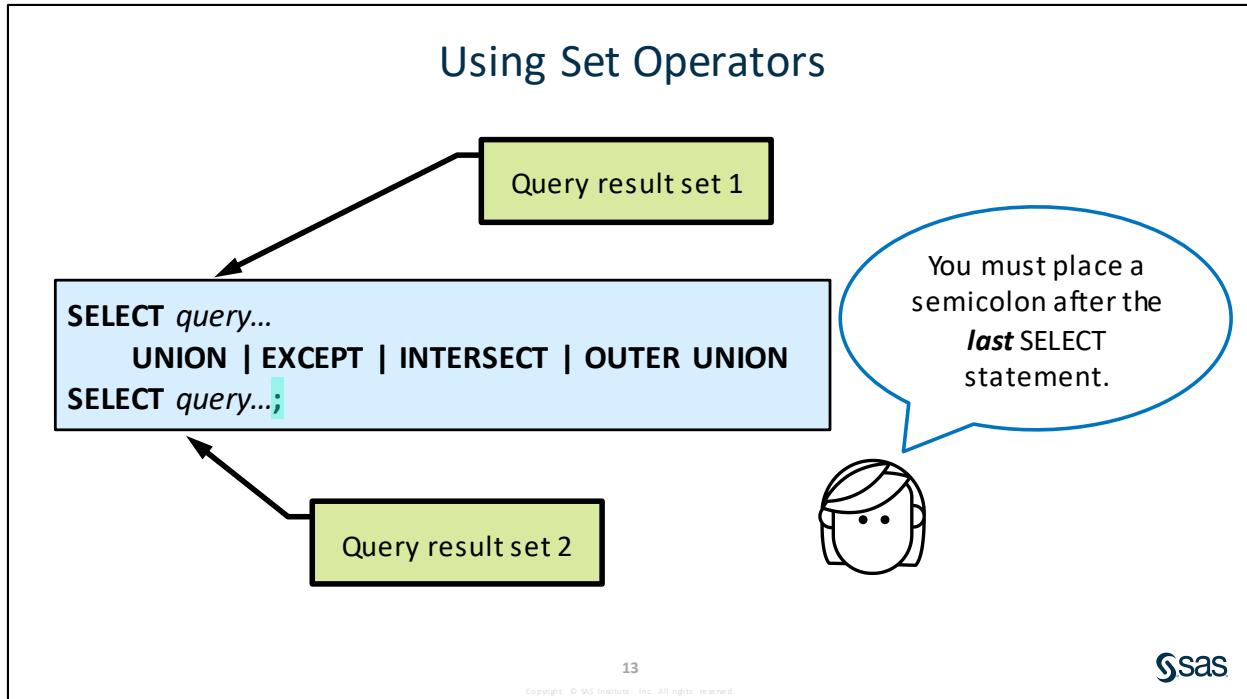
10

Copyright © SAS Institute Inc. All rights reserved.

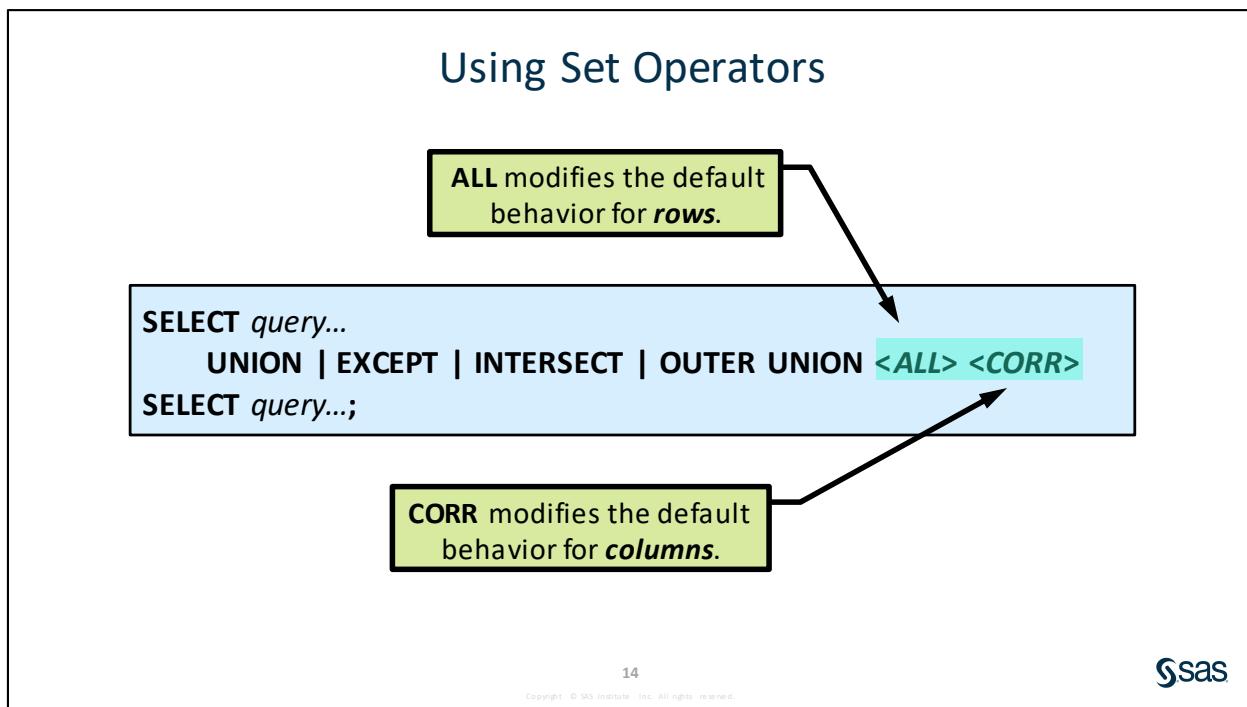


The **CustomerID** column exists in all tables.





You must place a semicolon after the last SELECT statement only.



5.2 INTERSECT, EXCEPT, and UNION

INTERSECT Operator

salesemail

CustomerID	EmailResp
1939774314	Accepted
1999302252	Declined
1963960449	Declined
1908694347	Accepted
1960311448	Accepted
1905044343	Declined

salesphone

CustomerID	SalesRep	PhoneResp
1939774314	121038	Declined
1999302252	120145	Call Back
1999302252	120145	Accepted
1963960449	120145	Declined
1987175132	120145	Declined
1970095442	121137	Accepted

Which customers have responded to both *email* and *phone* sales?



16

INTERSECT Operator

```
SELECT query...
INTERSECT <ALL> <CORR>
SELECT query...;
```

```
proc sql;
select CustomerID
  from sq.salesemail
intersect
select CustomerID
  from sq.salesphone;
quit;
```

Result Set 1

CustomerID
1939774314
1999302252
1963960449
...

Result Set 2

CustomerID
1939774314
1999302252
1999302252
...

17

Default Behavior of the INTERSECT Operator

Result Set 1

CustomerID
1939774314
1999302252
1963960449
1908694347
1960311448
1905044343

Result Set 2

CustomerID
1939774314
1999302252
1999302252
1963960449
1987175132
1970095442

First, **duplicate** rows are removed from each of the intermediate result sets if they exist.

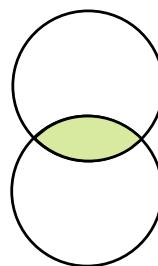
Default Behavior of the INTERSECT Operator

Result Set 1

CustomerID
1939774314
1999302252
1963960449
1908694347
1960311448
1905044343

Result Set 2

CustomerID
1939774314
1999302252
1999302252
1963960449
1987175132
1970095442



Rows from the **first** intermediate result set that are also in the **second** intermediate result set are selected.

INTERSECT Operator

```
title "Customers Who Have Responded to
Both Email and Phone Sales";

proc sql;
select CustomerID
  from sq.salesemail
intersect
select CustomerID
  from sq.salesphone;
quit;
```

Customers Who Have Responded to Both Email and Phone Sales

CustomerID
1939774314
1963960449
1999302252

20


Copyright © SAS Institute Inc. All rights reserved.

5.01 Activity

Open **s105a01.sas** from the **activities** folder and perform the following to find unique customers who have responded by **phone** and **email**:

1. Run the first queries to preview the **sq.salesemail** and **sq.salesphone** tables. Examine the columns in both tables.
2. In the **Intersect** section, examine and run the query. Did the query run successfully? Why not?
3. Add the **CORR** keyword after the **INTERSECT** set operator. Run the query. Did the query run successfully? Why?

21


Copyright © SAS Institute Inc. All rights reserved.

Set Operators and Joins

```
proc sql;
select CustomerID
  from sq.salesemail
intersect
select CustomerID
  from sq.salesphone;
quit;
```

```
proc sql;
select distinct e.CustomerID
  from sq.salesemail as e inner join
       sq.salesphone as p
     on e.CustomerID = p.CustomerID;
quit;
```

You can use an **inner join** to produce identical results.

CustomerID
1939774314
1963960449
1999302252

24

Copyright © SAS Institute Inc. All rights reserved.



You can use an INNER JOIN to produce identical results. In this example, we use an inner join to find all matches of the two tables. The DISTINCT keyword removes any duplicate rows resulting in the same result as the INTERSECT set operator.

EXCEPT Operator

saleslist

UserID	CustomerID	...
marremartinez6531@ismissing.com	1939774314	...
kimlihuffman843@fakeemail.com	1958716829	...
heacamoredock827@invalid.com	1999302252	...
heljaboone8613@invalid.com	1963960449	...
jambeerskin7521@fakeemail.com	1987175132	...
aranihale6320@ismissing.com	1970095442	...
terlowwhite687@invalid.com	1908694347	...
barmablanton521@n/a.com	1918638906	...
wilhoflores5912@notreal.com	1960311448	...
normidameron9028@voidemail.com	1905044343	...

salesemail

CustomerID	EmailResp
1939774314	Accepted
1999302252	Declined
1963960449	Declined
1908694347	Accepted
1960311448	Accepted
1905044343	Accepted

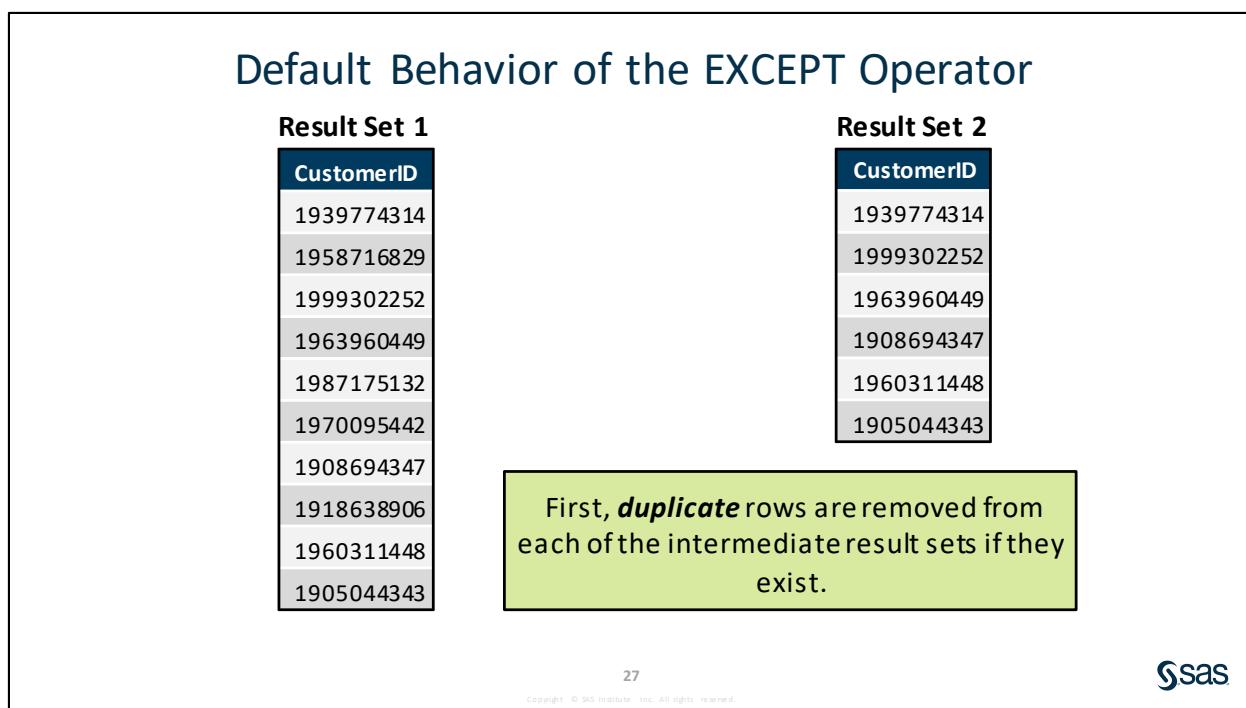
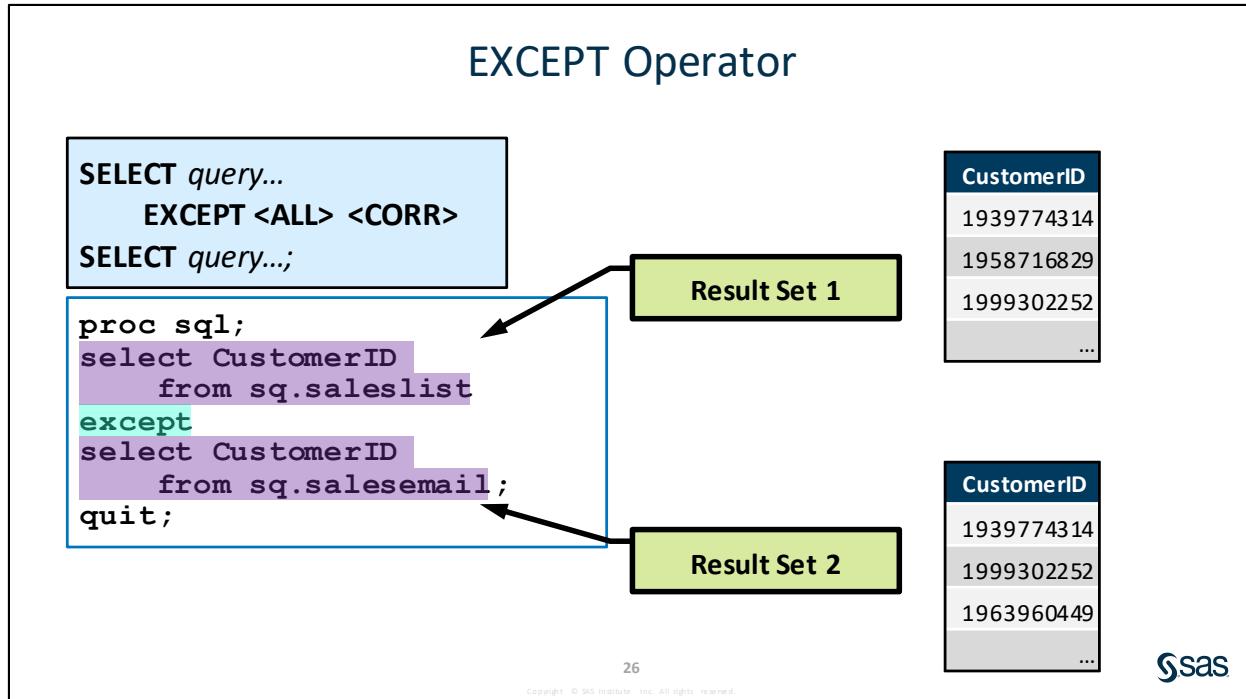
List of customers who have not responded to our sales **email**



25

Copyright © SAS Institute Inc. All rights reserved.





Default Behavior of the EXCEPT Operator

Result Set 1

CustomerID
1939774314
1958716829
1999302252
1963960449
1987175132
1970095442
1908694347
1918638906
1960311448
1905044343

Result Set 2

CustomerID
1939774314
1999302252
1963960449
1908694347
1960311448
1905044343

Rows from the **first** intermediate result set that are **not in the second** intermediate result set are selected.

EXCEPT Operator

```
title "Customers Who Haven't Responded to the Sales Email";
proc sql;
select CustomerID
  from sq.saleslist
 except
select CustomerID
  from sq.salesemail;
quit;
```

Customers Who Haven't Responded to the Sales Email
CustomerID
1918638906
1958716829
1970095442
1987175132

5.02 Activity

Open **s105a02.sas** from the **activities** folder and perform the following to find all target customers who have not responded to our sales phone call:

1. Run the first queries to preview the **sq.saleslist** and **sq.salesphone** tables. Examine the columns in both tables.
2. Complete the query to find all customers from the **sq.saleslist** table who have not responded to our sales call in **sq.salesphone**.
3. How many customers have not responded to our phone call?

Set Operators and Subqueries

```
select distinct CustomerID
  from sq.saleslist
 where CustomerID not in(select CustomerID
                           from sq.salesphone);
```

```
select CustomerID
  from sq.saleslist
 except
 select CustomerID
   from sq.salesphone;
```

CustomerID
1905044343
1908694347
1918638906
1958716829
1960311448

You can use a **subquery** to produce identical results.



You can use a subquery to produce identical results. In this example, we use a subquery to return all **CustomerID** values in the **salesphone** table. Then we use the WHERE clause to subset for all **CustomerID** values in the **saleslist** table that are not in the list returned by our subquery. The DISTINCT keyword removes any duplicate rows, resulting in the same result as the EXCEPT set operator.

UNION Operator

salesemail	
CustomerID	EmailResp
1939774314	Accepted
1999302252	Declined
1963960449	Declined
1908694347	Accepted
1960311448	Accepted
1905044343	Declined

salesphone		
CustomerID	SalesRep	PhoneResp
1939774314	121038	Declined
1999302252	120145	Call Back
1999302252	120145	Accepted
1963960449	120145	Declined
1987175132	120145	Declined
1970095442	121137	Accepted
1955264298	121038	Declined



Copyright © SAS Institute Inc. All rights reserved.

33

Sas

Find the total number of **unique** customers who responded to either **phone** or **email**.

UNION Operator

```
SELECT query...
UNION <ALL> <CORR>
SELECT query...;
```

```
proc sql;
select CustomerID
from sq.salesemail
union
select CustomerID
from sq.salesphone;
quit;
```

Result set 1

CustomerID
1939774314
1999302252
1963960449
...

Result set 2

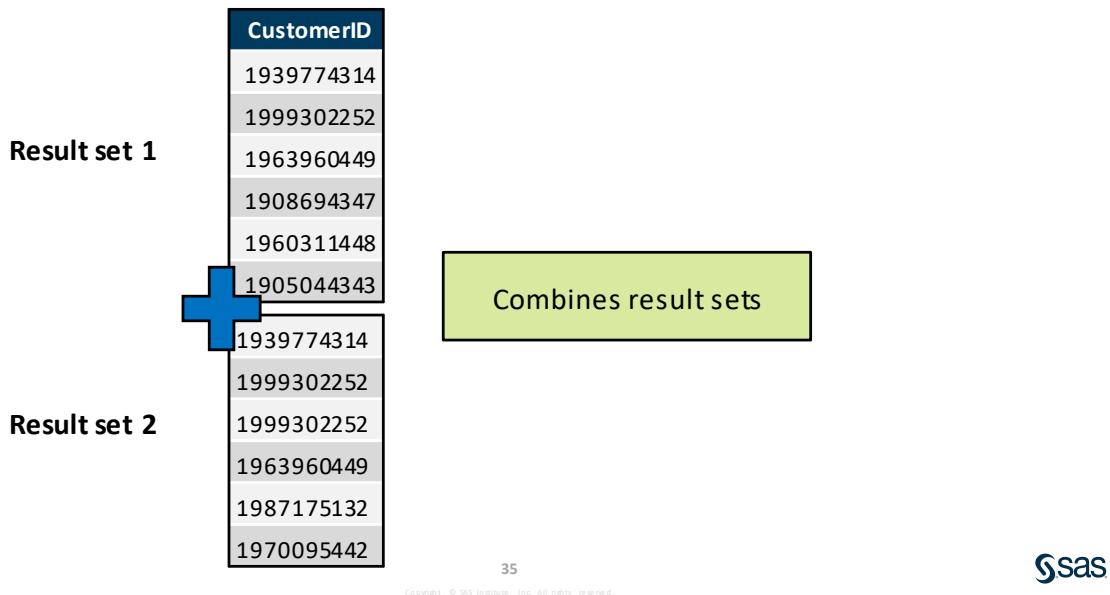
CustomerID
1939774314
1999302252
1999302252
...

Copyright © SAS Institute Inc. All rights reserved.

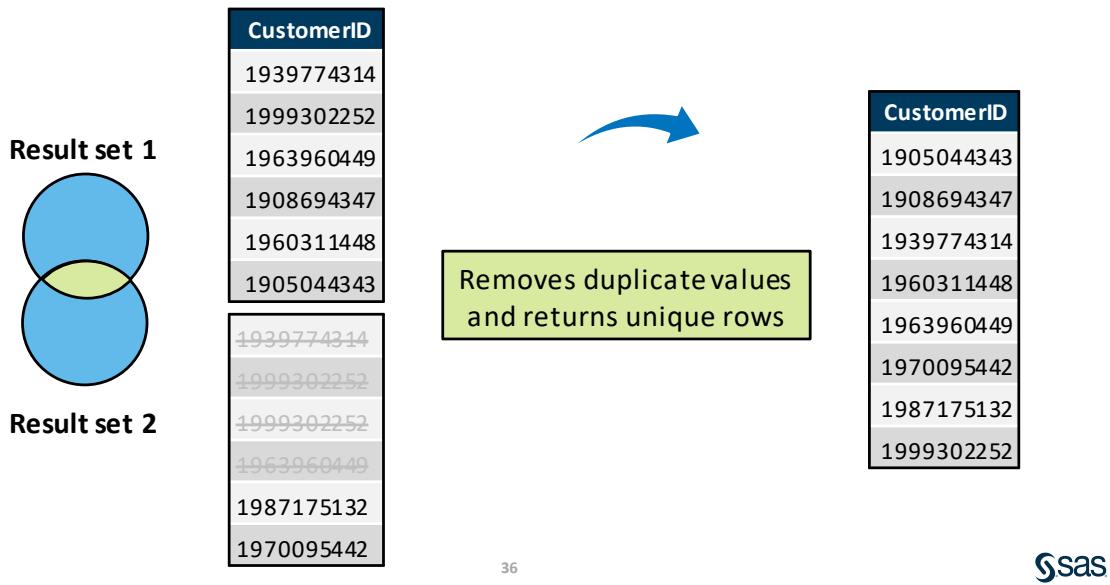
34

Sas

Default Behavior of the UNION Operator



Default Behavior of the UNION Operator





Using the UNION Set Operator

Scenario

Use the UNION set operator to count the number of unique customers who responded to an email or a phone sales attempt.

Files

- **s105d01.sas**
- **salesemail** – a SAS table that contains email response information from targeted customers
- **salesphone** – a SAS table that contains phone response information from targeted customers

Syntax

```
PROC SQL;
  SELECT query...
    UNION <ALL> <CORR>
  SELECT query...;
  QUIT;
```

Notes

- The UNION set operator produces all unique rows from both queries.
- The CORR keyword overlays columns that have the same name in both tables. When used with EXCEPT, INTERSECT, and UNION, CORR suppresses columns that are not in both tables.

Demo

1. Open the **s105d01.sas** program in the **demos** folder and find the **Demo** section. Under **Explore the SALESEMAIL and SALESPHONE Tables**, run the two queries. Discuss the tables and the desired outcome.

```
proc sql;
  select *
    from sq.salesemail;
  select *
    from sq.salesphone;
quit;
```

2. In the next section, complete the query to find all unique customers who responded to either an email or phone call. Begin with the SELECT statement and select all columns from the **sq.salesemail** table.

```
proc sql;
  select *
    from sq.salesemail
quit;
```

3. Use the UNION set operator followed by another SELECT statement to select all columns from the **sq.salesphone** table. Run the query and examine the syntax error.

```
proc sql;
select *
  from sq.salesemail
 union
select *
  from sq.salesphone;
quit;
```

4. Add the keyword CORR after the UNION set operator. Run the query and examine the log and results.

Note: The CORR keyword aligns the columns that have the same name in both tables and removes any columns not found in both tables.

```
proc sql;
select *
  from sq.salesemail
 union corr
select *
  from sq.salesphone;
quit;
```

5. Remove the CORR keyword, and specify the **CustomerID** column in both SELECT clauses. Run the query and examine the log and results.

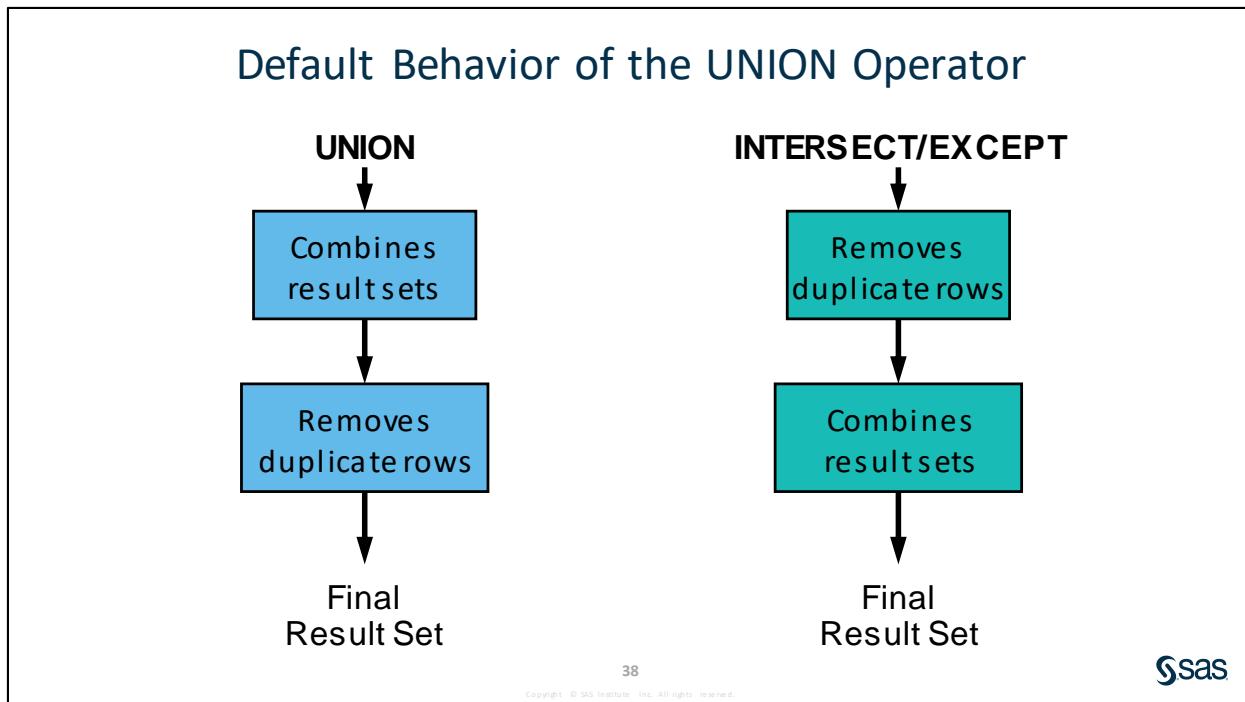
```
proc sql;
select CustomerID
  from sq.salesemail
 union corr
select CustomerID
  from sq.salesphone;
quit;
```

6. Add another SELECT statement at the first line in the SQL procedure. Use the COUNT(*) function to count all rows. Name the column **TotalNum**. Add a FROM clause and use the previous query as a subquery in the FROM clause (in-line view). Be sure to add parentheses around the subquery. Run the query and examine the results.

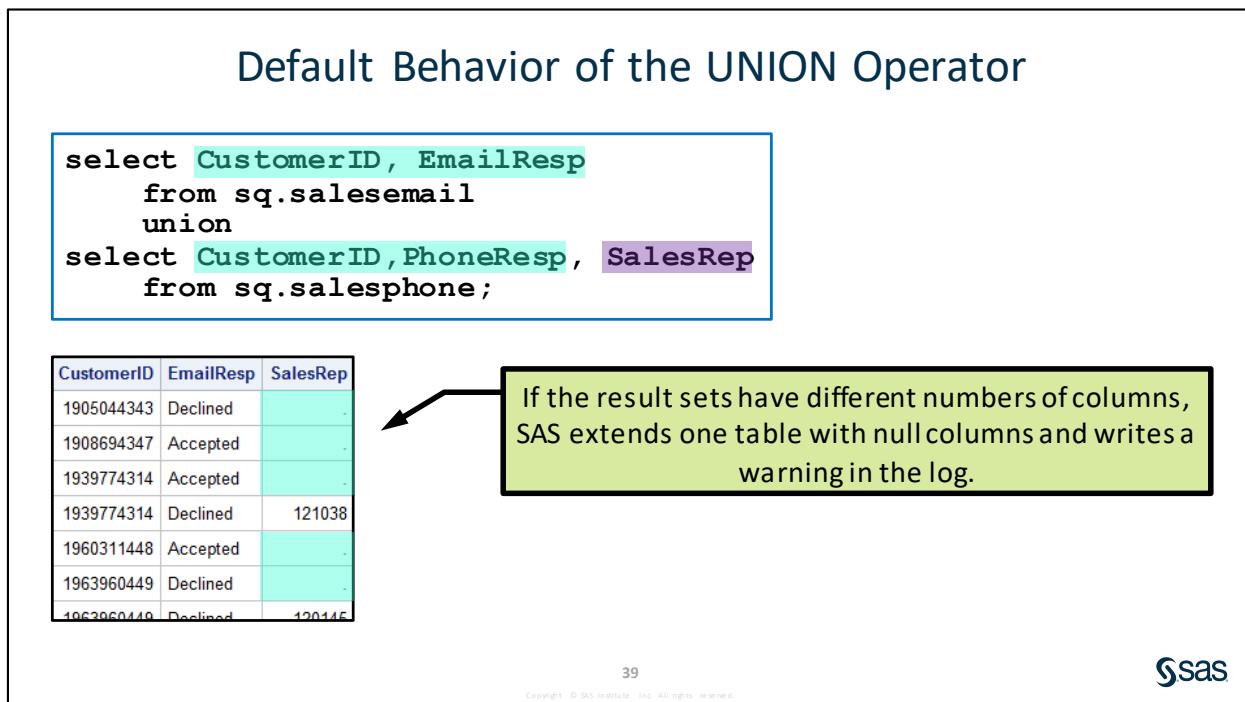
```
proc sql;
select count(*) as TotalNum
  from (select CustomerID
         from sq.salesemail
        union
        select CustomerID
         from sq.salesphone);
quit;
```

TotalNum
8

End of Demonstration



The UNION set operator works in a different order than the INTERSECT and EXCEPT operators. The UNION set operator first combines results sets and then removes duplicate rows. INTERSET and EXCEPT remove duplicate rows and then combine result sets.



In these cases, a note is written in the log: "WARNING: A table has been extended with null columns to perform the UNION set operation."

Behavior of UNION Operator with ALL

```
proc sql;
  select CustomerID
    from sq.salesemail
  union all
  select CustomerID
    from sq.salesphone
  order by CustomerID;
quit;
```

CustomerID
1905044343
1908694347
1939774314
1939774314
1960311448
1963960449
1963960449
1970095442
1987175132
1999302252
1999302252
1999302252

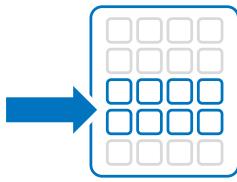
The **ALL** keyword does *not* remove the duplicate rows.



40

The ALL keyword does *not* remove the duplicate rows.

ALL Keyword Considerations



Use when **duplicates** in final result will not cause problems



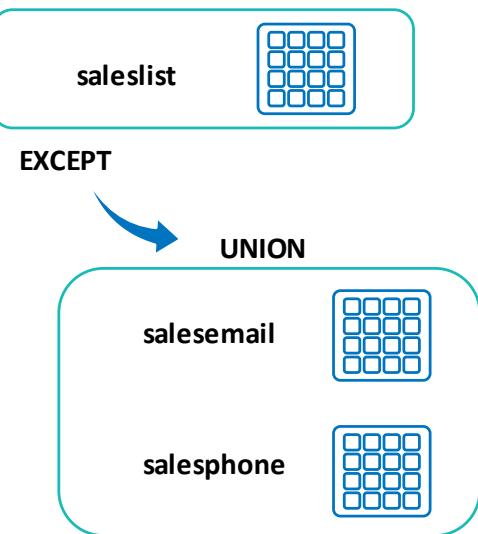
Use when **duplicates** are not possible

Using the ALL keyword improves efficiency of the set operators because SAS does not make a second pass to remove duplicates.

41

The ALL keyword is applicable only in the INTERSECT, UNION, and EXCEPT set operators.

Combining Set Operators



Create a list of customers who **have not responded** to either **email or phone** sales attempts.



42

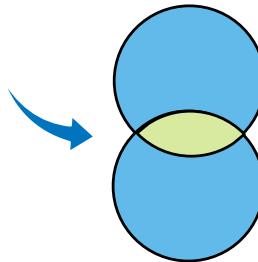
Combining Set Operators

```

proc sql;
select CustomerID
  from sq.saleslist
 except
(select customerid
   from sq.salesemail
 union
 select customerID
   from sq.salesphone);
quit;

```

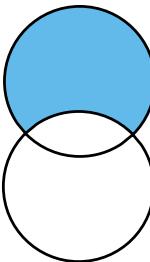
Find a unique list of customers who have responded using an in-line view.



43

Combining Set Operators

```
proc sql;
  select CustomerID
    from sq.saleslist
  except
  (select customerid
    from sq.salesemail
   union
   select customerID
     from sq.salesphone);
quit;
```



CustomerID
1918638906
1958716829

We have **two** customers who have not responded to any of our sales attempts.

44

Copyright © SAS Institute Inc. All rights reserved.



Syntax Summary



```
SELECT query...
UNION | EXCEPT | INTERSECT <ALL> <CORR>
SELECT query...;
```

Set Operators

45

Copyright © SAS Institute Inc. All rights reserved.





Practice

Level 1

1. Using the EXCEPT SET Operator

Use the EXCEPT SET operator to generate a report listing of merchants in the **sq.merchant** table who are not listed in the **sq.transaction** table.

- Select **MerchantID** from the **sq.merchant** table.
- Use the EXCEPT SET operator.
- Select **MerchantID** from the **sq.transaction** table.
- Order the results by **MerchantID**.
- Add an appropriate title.

Results

Merchants without Transactions	
	Merchant ID
	502136
	517039
	518339
	565543
	565603
	585048

- How many merchants do not have a transaction?

2. Using the UNION SET Operator

Using the **sq.employee** table as input, create a report that displays the total salary and total number of employees. In addition, add rows to display the salary information for US and AU employees separately. Use the UNION SET operator to combine the three rows of output for the final report.

- Write a query to create a report for all employees.
- Create the first row of the report. Include the constant text **Total Paid to All Employees**, the sum of **Salary** formatted with the DOLLAR. format, and the total number of employees using the COUNT function. Name the count column **Total**.
 - Run the query and compare your results.

Results

		Total
Total Paid to All Employees	\$16,129,600	424

- b. Write a query to create a report for all US employees.
- 1) Create the second row of the report. Include the constant text **Total Paid to US Employees**, the sum of **Salary** formatted with the DOLLAR. format, and the total number employees using the COUNT function. Name the count column **Total**.
 - 2) Filter the data for US employees by using the UPCASE function to select rows with the **Country** value *US*.
 - 3) Run the query and compare your results.

Results

		Total
Total Paid to US Employees	\$12,790,135	316

- c. Write another query to create the last row of the report for all AU employees.
- 1) Follow the previous steps. Replace US with AU.
 - 2) Run the query and compare your results.

Results

		Total
Total Paid to AU Employees	\$3,339,465	108

- d. Combine the results of all three queries into a single query using UNION SET operators.
- 1) Order the results by descending total salary.
 - 2) Add an appropriate title.

Results

Country Specific Salary Information		
		Total
Total Paid to All Employees	\$16,129,600	424
Total Paid to US Employees	\$12,790,135	316
Total Paid to AU Employees	\$3,339,465	108

- e. Which country has more employees?

Level 2

3. Using the EXCEPT SET Operator with the DISTINCT Keyword

Using the **sq.statepopulation** table, generate a list of state codes for states without any customers.

- a. Write a query to list the unique **Name** values in the **sq.statepopulation** table. This list represents all available states.

- b. Write a separate query to list the unique **State** values from the **sq.customer** table. This list represents all states where customers reside.
- c. Combine the queries using the EXCEPT SET operator to display states with no customers. Add an appropriate title.

Results

States with No Customers	
Name	
PR	

- d. For which value (or values) of **State** are there no customers?

Challenge

4. Using SET Operators to Summarize Data

Determine what percentage of customers have accepted either the phone or email offer. The **sq.saleslist** table contains the full list of customers presented with an offer. The **sq.salesemail** and **sq.salesphone** tables contain email and phone responses.

- a. Write a query to use a UNION SET operation to combine the **CustomerID** values for customers who accepted either the phone or email offer. This will form the basis of your in-line view.

Results

CustomerID
1908694347
1939774314
1960311448
1970095442
1999302252

- b. Write a query to count the number of customers who have accepted either offer (step a above).
 - 1) Use the following formula to calculate the rate of offer acceptance:

```
select count(*) / (select count(*) from sq.saleslist)
from (your in-line view code from step 1)
```

 - 2) Name the calculated column **PctResp**. Format it as a percent with no decimals.
 - 3) Label the new column **Offer Acceptance Rate**.
 - 4) Add an appropriate title to the report.

Results



- c. What is the acceptance rate?

End of Practices

5.3 OUTER UNION

OUTER UNION Operator

salesemail

CustomerID	EmailResp
1939774314	Accepted
1999302252	Declined
1963960449	Declined
1908694347	Accepted
1999302252	Accepted

salesphone

CustomerID	SalesRep	PhoneResp
1939774314	121038	Declined
1999302252	120145	Call Back
1999302252	120145	Accepted
1963960449	120145	Declined
1999302252	120145	Declined

Desired Results

CustomerID	Resp	SalesRep
1939774314	Accepted	.
1999302252	Declined	.
1963960449	Declined	.
1908694347	Accepted	.
1960311448	Accepted	.
1905044343	Declined	.
1939774314	Declined	121038
1999302252	Call Back	120145
1999302252	Accepted	120145
1963960449	Declined	120145
1987175132	Declined	120145
1970095442	Accepted	121137



Create a report that contains a list of **all** customer **phone** and **email** responses.



48

Copyright © SAS Institute Inc. All rights reserved.

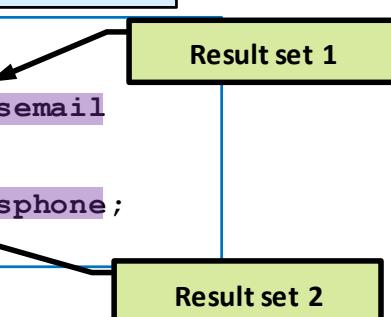
OUTER UNION Operator

```
SELECT query...
    OUTER UNION <CORR>
SELECT query...;
```



```
proc sql;
select *
  from sq.salesemail
outer union
select *
  from sq.salesphone;
quit;
```

Result set 1



Result set 2

CustomerID	EmailResp
1905044343	Declined
1908694347	Accepted
...	...

CustomerID	SalesRep	PhoneResp
1939774314	121038	Declined
1963960449	120145	Declined
...



49

Copyright © SAS Institute Inc. All rights reserved.

Default Behavior of the OUTER UNION Operator

CustomerID	EmailResp	CustomerID	SalesRep	PhoneResp
1939774314	Accepted	.	.	.
1999302252	Declined	.	.	.
1963960449	Declined	.	.	.
1908694347	Accepted	.	.	.
1960311448	Accepted	.	.	.
1905044343	Declined	.	.	.
.		1939774314	121038	Declined
.		1999302252	120145	Call Back
.		1999302252	120145	Accepted
.		1963960449	120145	Declined
.		1987175132	120145	Declined
.		1970095442	121137	Accepted

ALL **columns** and **rows** are combined. Same name columns are **not overlaid**.



Using the OUTER UNION Set Operator

Scenario

Use the OUTER UNION set operator with the CORR keyword to combine the **salesemail** and **salesphone** tables.

Files

- **s105d02.sas**
- **salesemail** – a SAS table that contains email response information from targeted customers
- **salesphone** – a SAS table that contains phone response information from targeted customers

Syntax

```
PROC SQL;
SELECT query...
    OUTER UNION <CORR>
SELECT query...;
QUIT;

table(RENAME=(old-name-1=new-name-1 ...))
```

Notes

- The OUTER UNION set operator produces all unique rows from both queries.
- The CORR keyword overlays columns that have the same name in both tables. When used with OUTER UNION, CORR includes columns that are not in both tables
- The RENAME= data set option changes the name of a column.

Demo

1. Open the **s105d02.sas** program in the **demos** folder and find the **Demo** section. Run the query to perform an OUTER UNION concatenation of the **sq.salesemail** and **sq.salesphone** tables. Examine the results.

```
proc sql;
select *
  from sq.salesemail
  outer union
select *
  from sq.salesphone;
quit;
```

2. Add the CORR keyword after the OUTER UNION set operator. Run the query and examine the results.

```
proc sql;
select *
  from sq.salesemail
  outer union corr
```

```
select *
  from sq.salesphone;
quit;
```

3. Add the RENAME= option after the **salesemail** table and rename the column **EmailResp** to **Resp**. After the **salesphone** table, rename the column **PhoneResp** to **Resp**. Run the query and examine the results.

```
proc sql;
select *
  from sq.salesemail(rename=(EmailResp=Resp))
  outer union corr
select *
  from sq.salesphone(rename=(PhoneResp=Resp));
quit;
```

4. Remove the RENAME= option after each table. Modify the first SELECT clause and select the column **CustomerID**. In the first clause, also select **EmailResp** and change the column name to **Resp** using the AS keyword. Modify the second SELECT clause and select the **CustomerID**, **SalesRep**, and **PhoneResp** columns. Change the **PhoneResp** column name to **Resp** using the AS keyword. Run the query and examine the results.

```
proc sql;
select CustomerID, EmailResp as Resp
  from sq.salesemail(rename=(EmailResp=Resp))
  outer union corr
select CustomerID, SalesRep, PhoneResp as Resp
  from sq.salesphone(rename=(EmailResp=Resp));
quit;
```

5. Add the CREATE TABLE statement to create a table from the query results. Name the table **response1**.

```
proc sql;
create table response1 as
select CustomerID, EmailResp as Resp
  from sq.salesemail
  outer union corr
select CustomerID, SalesRep, PhoneResp as Resp
  from sq.salesphone;
quit;
```

6. Find the **SAS DATA Step** section. Briefly describe how you can retrieve the same results using the SAS DATA step.

```
data response2;
  length Resp $12;
  set sq.salesemail(rename=(EmailResp=Resp))
    sq.salesphone(rename=(PhoneResp=Resp));
run;
```

End of Demonstration

SQL Versus Traditional SAS Programming

```

proc sql;
create table response1 as
select CustomerID, EmailResp as Resp
  from sq.salesemail
outer union corr
select CustomerID, SalesRep,
      PhoneResp as Resp
  from sq.salesphone;
quit;

data response2;
  length Resp $12;
  set sq.salesemail(rename=(EmailResp=Resp))
    sq.salesphone(rename=(PhoneResp=Resp));
run;

```

52



Copyright © SAS Institute Inc. All rights reserved.

The OUTER UNION set operator is similar to the DATA step with multiple tables listed in the SET statement. OUTER UNION in SQL and the DATA step with the SET statement can produce similar results.

Syntax Summary

SELECT query...
OUTER UNION <CORR>
SELECT query...;



OUTER UNION

table(RENAME=(old-name-1=new-name-1 ...))

RENAME Data Set Option

53



Copyright © SAS Institute Inc. All rights reserved.

Beyond SQL Essentials

What if you want to ...

... learn more about combining and modifying SAS data sets?

- Read [Combining and Modifying SAS Data Sets: Examples, Second Edition](#).
- View [Combining SAS Data Sets: Methods](#) in the SAS documentation

... learn more about using SET operators?

- Read [SQL Set Operators: So Handy Venn You Need Them](#).

... learn more about the SQL procedure?

- Visit the **SQL Procedure** section on the ELP for a list of SAS papers.



Practice

Level 1

5. Using the OUTER UNION SET Operator

Create a report that shows the email and phone offer responses along with the sales representative if available.

- Using the **sq.salephone** table as input, write a query to list the following columns:
 - CustomerID**
 - a new column named **Response** based on the existing **PhoneResp** column
 - SalesRep** labeled **Sales Rep**
 - a new column named **Channel** with the constant text **Phone**

Run the query and compare your results.

Results

CustomerID	Response	Sales Rep	Channel
1939774314	Declined	121038	Phone
1999302252	Call Back	120145	Phone
1999302252	Accepted	120145	Phone
1963960449	Declined	120145	Phone
1987175132	Declined	120145	Phone
1970095442	Accepted	121137	Phone

- Using the **sq.salesemail** table as input, write a query to list the following columns:

- CustomerID**
- a new column named **Response** based on the existing **EmailResp** column
- a new column named **Channel** with the constant text **Email**

Run the query and compare your results.

Results

CustomerID	Response	Channel
1905044343	Declined	Email
1908694347	Accepted	Email
1939774314	Accepted	Email
1960311448	Accepted	Email
1963960449	Declined	Email
1999302252	Declined	Email

- c. Combine the two query results using the OUTER UNION SET operation.
- 1) Be mindful of the column alignment. Use the SET operator modifiers as needed.
 - 2) Order the results by **CustomerID** and **Response**.
 - 3) Add an appropriate title.
 - 4) Run the query to generate the final results.

Results

Offer Results with Sales Rep			
CustomerID	Response	Sales Rep	Channel
1905044343	Declined	.	Email
1908694347	Accepted	.	Email
1939774314	Accepted	.	Email
1939774314	Declined	121038	Phone
1960311448	Accepted	.	Email
1963960449	Declined	.	Email
1963960449	Declined	120145	Phone
1970095442	Accepted	121137	Phone
1987175132	Declined	120145	Phone
1999302252	Accepted	120145	Phone
1999302252	Call Back	120145	Phone
1999302252	Declined	.	Email

- d. To how many sales representatives can we attribute an accepted offer?

End of Practices

5.4 Solutions

Solutions to Practices

1. Using the EXCEPT SET Operator

```
/*s105s01.sas*/
title 'Merchants without Transactions';
proc sql;
select MerchantID
  from sq.merchant
 except
select MerchantID
  from sq.transaction
 order by MerchantID;
quit;
title;
```

How many merchants do not have a transaction? **Six merchants do not have a transaction.**

2. Using the UNION SET Operator

```
/*s105s02.sas*/
/*a*/
proc sql;
select 'Total Paid to All Employees',
       sum(Salary) format=dollar16.,
       count(*) as Total
  from sq.employee;
quit;

/*b*/
proc sql;
select 'Total Paid to US Employees',
       sum(Salary) format=dollar16.,
       count(*) as Total
  from sq.employee
 where upcase(Country)='US';
quit;

/*c*/
proc sql;
select 'Total Paid to AU Employees',
       sum(Salary) format=dollar16.,
       count(*) as Total
  from sq.employee
 where upcase(Country)='AU';
quit;
```

```
/*d*/
title 'Country Specific Salary Information';
proc sql;
select 'Total Paid to All Employees',
       sum(Salary) format=dollar16.,
       count(*) as Total
  from sq.employee
 union
select 'Total Paid to US Employees',
       sum(Salary) format=dollar16.,
       count(*) as Total
  from sq.employee
 where upcase(Country)='US'
 union
select 'Total Paid to AU Employees',
       sum(Salary) format=dollar16.,
       count(*) as Total
  from sq.employee
 where upcase(Country)='AU'
 order by 2 desc;
quit;
title;
```

Which country has more employees? **The US has more employees.**

3. Using the EXCEPT SET Operator with the Distinct Keyword

```
/*s105s03.sas*/

title 'States with No Customers';
proc sql;
select Name
  from sq.statepopulation
 except
select distinct State
  from sq.customer;
quit;
title;
```

For which value (or values) of **State** are there no customers?

PR (Puerto Rico) contains no customers.

4. Using SET Operators to Summarize Data

```
/*s105s04.sas*/

title 'Acceptance Rate';
proc sql;
select count(*)/(select count(*)
                  from sq.saleslist) as PctResp format=percent5.
                                         label='Offer Acceptance
                                               Rate'
  from (select CustomerID
```

```
        from sq.salesemail
        where EmailResp = 'Accepted'
        union
        select CustomerID
            from sq.salesphone
            where PhoneResp = 'Accepted') ;
quit;
title;
```

What is the acceptance rate? **50%**

5. Using the OUTER UNION Set Operator

```
/*s105s05.sas*/
title 'Offer Results with Sales Rep';
proc sql;
select CustomerID,
       PhoneResp as Response,
       SalesRep 'Sales Rep',
       'Phone' as Channel
      from sq.salesphone
      outer union corr
select CustomerID,
       EmailResp as Response,
       'Email' as Channel
      from sq.salesemail
      order by 1,2;
quit;
title;
```

To how many sales representatives can we attribute an accepted offer?

Two sales representatives

End of Solutions

Solutions to Activities and Questions

continued...

5.01 Activity – Correct Answer

2. In the **Intersect** section, examine and run the query. Did the query run successfully? Why not? **No.** By default, the **INTERSECT** set operator matches by column position, and the data *must* be of the same type.

```
proc sql;
  select *
    from sq.salesemail
  intersect
  select *
    from sq.salesphone;
quit;
```

CustomerID	EmailResp
1939774314	Accepted
1999302252	Declined
1963960449	Declined
1908694347	Accepted
1960311448	Accepted

CustomerID	SalesRep	PhoneResp
1939774314	120148	Declined
1999302252	120145	Call Back
1999302252	120145	Accepted
1963960449	120145	Declined
1987175132	120145	Declined

EmailResp is character. SalesRep is numeric

22



Copyright © SAS Institute Inc. All rights reserved.

5.01 Activity – Correct Answer

3. Add the CORR option after the INTERSECT keyword. Did the query run successfully? Why? **Yes.** The CORR keyword matches columns by name, and nonmatching columns are removed from the intermediate result sets.

```
proc sql;
  select *
    from sq.salesemail
  intersect corr
  select *
    from sq.salesphone;
quit;
```

CustomerID	EmailResp
1939774314	Accepted
1999302252	Declined
1963960449	Declined
1908694347	Accepted
1960311448	Accepted

CustomerID	SalesRep	PhoneResp
1939774314	121038	Declined
1999302252	100145	Call Back
1999302252	120145	Accepted
1963960449	120145	Declined
1987175132	120145	Declined

CORR matches by column names and removes columns not in both tables.

23



Copyright © SAS Institute Inc. All rights reserved.

5.02 Activity – Correct Answer

```
select CustomerID  
      from sq.saleslist  
except  
select CustomerID  
      from sq.salesphone;
```

```
select *  
      from sq.saleslist  
except corr  
select *  
      from sq.salesphone;
```

Five customers

CustomerID
1905044343
1908694347
1918638906
1958716829
1960311448

Lesson 6 Using and Creating Macro Variables in SQL

6.1	Creating User-Defined Macro Variables.....	6-3
6.2	Creating Data-Driven Macro Variables with PROC SQL	6-6
	Demonstration: Using a PROC SQL Data-Driven Macro Variable.....	6-10
	Demonstration: Concatenating Values in Macro Variables.....	6-17
	Practice.....	6-22
6.3	Solutions	6-28
	Solutions to Practices	6-28
	Solutions to Activities and Questions.....	6-33

6.1 Creating User-Defined Macro Variables

Efficiently Changing the Filter Value

```
proc sql;
create table CustomerGA as
select CustomerID, Employed, Race,
      Married, State, CreditScore
from sq.customer
where State="GA" and
      CreditScore > 650;
quit;
```

$GA \Leftrightarrow NC$

$650 \Leftrightarrow 700$

To change to the values
to **NC** and **700**, you
would have to manually
adjust the query.

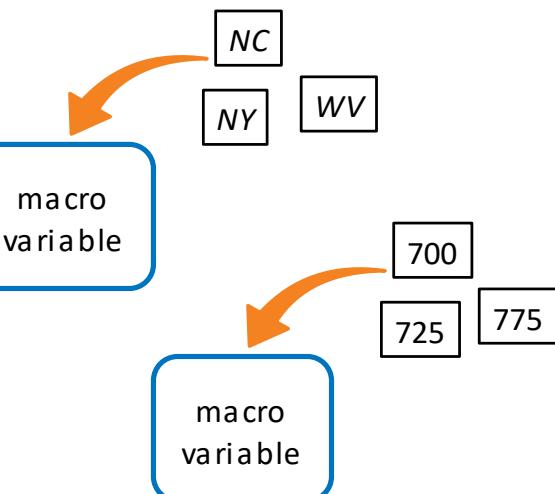


Sas

3

Copyright © SAS Institute Inc. All rights reserved.

Efficiently Changing the Filter Value



A SAS macro variable
stores **text** that is
substituted in your code
when it runs. It's like
an automatic
find-and-replace.



Sas

4

Copyright © SAS Institute Inc. All rights reserved.

Creating User-Defined Macro Variables

```
%LET macro-variable=value;
```

```
%let State=GA;
%let CreditMin=650;

proc sql;
create table CustomerGA as
select CustomerID, Employed, Race,
      Married, State, CreditScore
from sq.customer
where State="GA" and
      CreditScore > 650;
quit;
```

create
the
macro
variables

Macro Variables

Name	Value
State	GA
CreditMin	650

Macro variables are temporary, so when you exit SAS, they are deleted.

5


Copyright © SAS Institute Inc. All rights reserved.

Macro language statements begin with a % sign. To create a user-defined macro variable, you use the %LET statement. In the %LET statement, you specify the name of the macro variable, followed by an equal sign and then the text string that you want to store.

It is recommended that you do not include quotation marks when you define the macro variable value. Use quotation marks when necessary after the macro variable is resolved.

Using User-Defined Macro Variables

```
&macro-variable;
```

```
%let State=GA;
%let CreditMin=650;

proc sql;
create table Customer&State as
select CustomerID, Employed, Race,
      Married, State, CreditScore
from sq.customer
where State="&State" and
      CreditScore > &CreditMin;
quit;
```

use the
macro
variables

When the query executes,
values of **State** and
CreditMin are replaced by
GA and **650**.



6

Copyright © SAS Institute Inc. All rights reserved.

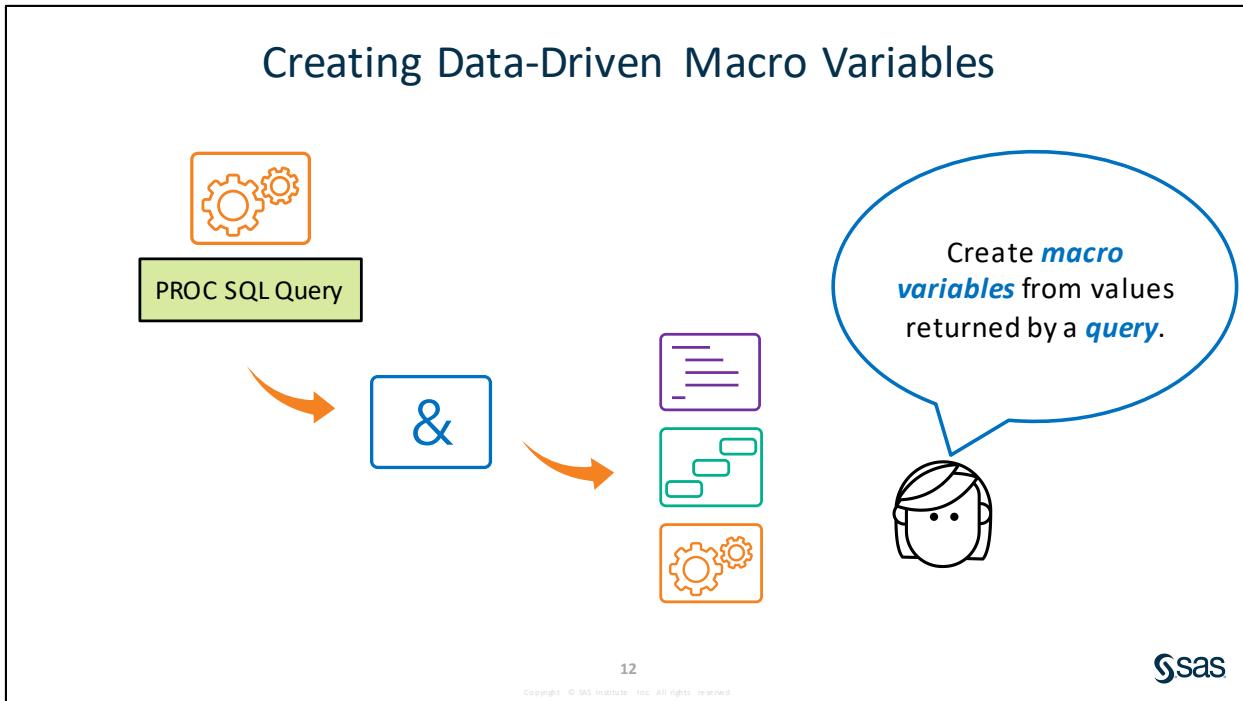
Macro variables must be enclosed in double quotation marks when the macro variables are used as a character string.

6.01 Activity

Open **s106a01.sas** from the **activities** folder and perform the following tasks to create and use user-defined macro variables:

1. Run the program. Examine the log and results. Confirm that the name of the newly created table is **customerga** and contains 957 rows.
2. Replace the values **GA** and **650** in the %LET statements with **NC** and **700**. Run the program. Examine the log and results. What is the name of the newly created table? How many rows?
3. Change the double quotation marks in the WHERE clause expression to single quotation marks. Run the query. How many rows are in the new table?

6.2 Creating Data-Driven Macro Variables with PROC SQL



In PROC SQL, you can create macro variables from values returned by a query. You can then reference those stored macro variables in other PROC steps, DATA steps, titles, and footnotes.

Think of it this way: ***Build once, use many.***

Subquery Returning a Value

```
select Name, PopEstimate1
  from sq.statepopulation
 where PopEstimate1 > (select avg(PopEstimate1)
                           from sq.statepopulation);
```

Average Estimated Population for Next Year is: ???

Name	PopEstimate1
AZ	6945452
CA	39209127
FL	20629982
GA	10304763
IL	12826895

How can we document the average value of PopEstimate1 in the **title**?



13

Steps for Creating Data-Driven Macro Variables



Create the Macro Variable
Using PROC SQL



Use the Macro Variable

14

Step 1: Creating the Macro Variable Using PROC SQL

```
SELECT col-name
  INTO :macvar
  FROM table;
  select avg(PopEstimate1)
    into :AvgEst1
    from sq.statepopulation;
```

6278420

Macro Variables

Name	Value
AvgEst1	6278420

The **value** is stored in the **AvgEst1** macro variable.



15

Sas

If the query produces more than one row of output, the macro variable will contain only the value from the first row.

If the query has no rows in its output, then the macro variable is not modified. If the macro variable does not exist yet, it will not be created.

Displaying Macro Variable Values

%PUT text;

%PUT displays the resolved macro value in the SAS log.

```
%put AvgEst is &AvgEst1;
%put &=AvgEst1;
```

```
%put AvgEst is &AvgEst1;
AvgEst is 6278420

%put &=AvgEst1;
AVGEST1= 6278420
```

16

Sas

A %PUT statement that contains **&=** followed by the name of the macro variable displays the name and value of a macro variable.

Step 2: Using the Macro Variable

```
title "Average Estimated Population for Next Year: &AvgEst1";  
proc sql;  
select Name, PopEstimate1  
from sq.statepopulation  
where PopEstimate1 > &AvgEst1;  
quit;
```

Average Estimated Population for Next Year: 6278420

Name	PopEstimate1
AZ	6945452
CA	39209127
FL	20629982
GA	10304763
IL	12826895
IN	6633344
MA	6826022

Now we can use the macro variable **AvgEst1** in *titles* and *queries*.



Sas



Using a PROC SQL Data-Driven Macro Variable

Scenario

Use PROC SQL to create and use a macro variable.

Files

- **s106d01.sas**
- **statepopulation** – a SAS table that contains estimated yearly population by state

Syntax

```
PROC SQL NOPRINT;
SELECT col-name
    INTO :macvar
    FROM table;
QUIT;

%PUT &=macvar;
OPTIONS SYMBOLGEN;
```

Notes

- A macro variable stores a text string that can be substituted into a SAS program.
- If the query produces more than one row of output, the macro variable will contain only the value from the first row.
- Macro variables can be referenced in a program by preceding the macro variable name with & (ampersand).
- The %PUT statement writes the value of the macro variable to the SAS log.
- The SYMBOLGEN option displays a note identifying the macro variable and its resolved value in the log.

Demo

1. Open the **s106d01.sas** program in the **demos** folder and find the **Demo** section. Run the first query in the **Create the Macro Variable Using PROC SQL** section. View the results.

```
proc sql;
select avg(PopEstimate1)
    into :AvgEst1
    from sq.statepopulation;
quit;
```

2. In the same query, add the NOPRINT option in the PROC SQL statement. At the end of the query, add a %PUT statement to view the value of the newly created macro variable in the SAS log. Run the query and view the SAS log.

```
proc sql noprint;
select avg(PopEstimate1)
  into :AvgEst1
  from sq.statepopulation;
quit;
%put &AvgEst1;
```

```
43      %put &AvgEst1;
AVGEST1= 6278420
```

3. Move to **STEP 2** and apply the newly created macro variable **AvgEst1** in the TITLE statement and the WHERE clause. Run the query. View the log and results.

Note: To format a macro variable in the TITLE statement, you can use `%left(%qsysfunc(putn(&AvgEst1,dollar16.)))`. Scroll to the bottom of the program file for this demo to view the solution code.

```
title "Average Estimated Population for Next Year: &AvgEst1";
proc sql;
select Name, PopEstimate1
  from sq.statepopulation
 where PopEstimate1 > &AvgEst1;
quit;
```

4. Above the query, add the statement OPTIONS SYMBOLGEN to see the value that was substituted in the code. Below the query, turn off the options by adding the OPTIONS NOSYMBOLGEN statement. Run the query and examine the log.

```
options symbolgen;

...sql query

options nosymbolgen;
```

```
31      options symbolgen;
SYMBOLGEN: Macro variable AVGEST1 resolves to 6278420
32      title "Average Estimated Population for Next Year:&AvgEst1";
33      proc sql;
34      select Name, PopEstimate1
35          from sq.statepopulation
36          where PopEstimate1
SYMBOLGEN: Macro variable AVGEST1 resolves to 6278420
36      !                               > &AvgEst1;
37      quit;
```

End of Demonstration

Displaying Macro Variable Values

```
OPTIONS SYMBOLGEN | NOSYMBOLGEN;
```

Use the SYMBOLGEN system option to see the value that was substituted in the code.

```
options symbolgen;
title "Average Estimated Population for Next Year: &AvgEst1";
proc sql;
select Name, PopEstimate1
from sq.statepopulation
where PopEstimate1 > &AvgEst1;
quit;
options nosymbolgen;
```

```
31      options symbolgen;
SYMBOLGEN: Macro variable AVGEST1 resolves to 6278420
32      title "Average Estimated Population for Next Year: &AvgEst1";
33      proc sql;
```

19



The SYMBOLGEN option displays the results of resolving macro variable references in the SAS log. This option is useful for debugging.

Creating Multiple Macro Variables

```
proc sql noprint;
select avg(PopEstimate1), min(PopEstimate1),
       max(PopEstimate1), count(PopEstimate1)
  into :AvgEst1, :MinEst1, :MaxEst1, :TotalCount
  from sq.statepopulation;
quit;
```

The INTO clause can create multiple macro variables from query results.

Macro Variables

Name	Value
AvgEst1	6278420
MinEst1	584290
MaxEst1	39209127
TotalCount	52

20



You list the names of the macro variables to be created in the INTO clause. Precede each macro variable name with a colon. PROC SQL generates the values that are assigned to these variables by executing the query.

Displaying Macro Variables

```
%put &AvgEst1;
%put &TotalCount;
%put Min Value is &MinEst1;
%put Max: &MaxEst1;
```

Display each new macrovariable and its value in the log.

```
37      %put &AvgEst1;
AVGEST1= 6278420
38      %put &TotalCount;
TOTALCOUNT=      52
39      %put Min Value is &MinEst1;
Min Value is 584290
40      %put Max: &MaxEst1;
Max: 39209127
```

Do you notice anything interesting about the way the values are *displayed*?



Sas

21

Copyright © SAS Institute Inc. All rights reserved.

Displaying Macro Variables

```
37      %put &AvgEst1;
AVGEST1= 6278420
38      %put &TotalCount;
TOTALCOUNT= 52
39      %put Min Value is &MinEst1;
Min Value is 584290
40      %put Max: &MaxEst1;
Max: 39209127
```

By default, macro variables that contain numeric values are *formatted using* the BEST8. format.



Sas

22

Copyright © SAS Institute Inc. All rights reserved.

When storing a value in a single macro variable, PROC SQL preserves leading or trailing blanks. By default, when creating macro variables that contain numeric values, the values are formatted using the BEST8. format, which preserves leading blanks for numbers that have fewer than eight digits. This can also cause the value to be displayed using scientific notation for numbers that have more than eight digits. You can use another format, such as the *w.* format, to display the value without using scientific notation.

Removing Leading and Trailing Blanks

INTO :macvar TRIMMED

The TRIMMED option can be used to trim the leading and trailing blanks.

```
proc sql noprnt;
select avg(PopEstimate1), min(PopEstimate1),
       max(PopEstimate1), count(PopEstimate1)
  into :AvgEst1 trimmed, :MinEst1 trimmed,
       :MaxEst1 trimmed, :TotalCount trimmed
  from sq.statepopulation;
quit;
%put &AvgEst1;
%put &TotalCount;
%put Min Value is &MinEst1;
%put Max: &MaxEst1;
```

```
48      %put &AvgEst1;
AVGEST1=6278420
49      %put &TotalCount;
TOTALCOUNT=52
50      %put Min Value is &MinEst1;
Min Value is 584290
51      %put Max: &MaxEst1;
Max: 39209127
```

23


Copyright © SAS Institute Inc. All rights reserved.

6.02 Activity

Open **s106a02.sas** from the **activities** folder and perform the following tasks to store a number larger than eight digits in a macro variable:

- Run the program in **STEP 1** to create the macro variables **MaxPop** and **TotalCtry**. The macro variables store the estimated maximum three-year population estimate for a country and the total number of countries. View the log. Notice that **MaxPop** stores a value in scientific notation.
- Examine **STEP 2** of the program to find the country with the maximum estimated three-year population. Run the program to use the macro variables that you created. Confirm that no rows were returned.
- Add the **FORMAT=10.** column modifier to the **max(estYear3Pop)** column in **STEP 1**. Run the entire program. Which country has the largest three-year estimated population?

24


Copyright © SAS Institute Inc. All rights reserved.

Subquery Returning Multiple Values

```
proc sql;
create table division3 as
select *
  from sq.customer
 where State in (select Name
                  from sq.statepopulation
                 where Division = '3');
quit;
```

"IL","IN","MI","OH","WI"

How can we store a
list of values in a
macro variable?



Sas

29

Copyright © SAS Institute Inc. All rights reserved.

Concatenating Values in Macro Variables

```
SELECT col-name
  INTO :macvar SEPARATED BY "delimiter"
  FROM table;
```

```
proc sql noprint;
select Name
  into :StateList separated by ","
  from sq.statepopulation
 where Division = '3';
quit;
```

```
%put &=StateList;
```

48 %put &=StateList;
STATELIST=IL,IN,MI,OH,WI

30

Sas

Concatenating Values in Macro Variables

```
title "States in Division 3: &StateList";
proc sql;
create table division3 as
select *
  from sq.customer
 where State in (&StateList);
quit;
```

```
48      %put &StateList;
STATELIST=IL,IN,MI,OH,WI
```

Will you be able to use the **StateList** macro variable to subset the table?



Sas

31

Copyright © SAS Institute Inc. All rights reserved.

Concatenating Values in Macro Variables

```
title "States in Division 3: &StateList";
proc sql;
create table division3 as
select *
  from sq.customer
 where State in (IL,IN,MI,OH,WI);
quit;
```

Character values need to each be enclosed in ***quotation marks***.



Sas

32

Copyright © SAS Institute Inc. All rights reserved.



Concatenating Values in Macro Variables

Scenario

Use PROC SQL to create a concatenated list of state values separated by a comma, with each value enclosed in quotation marks.

Files

- **s106d02.sas**
- **statepopulation** – a SAS table that contains estimated yearly population by state
- **customer** – a SAS table that contains one row per customer

Syntax

```
PROC SQL;
SELECT col-name
    INTO :macvar SEPARATED BY "delimiter"
    FROM table;
QUIT;

QUOTE(argument)
STRIP(argument)
```

Notes

- You can concatenate the values of one column into one macro variable.
- Use the SEPARATED BY keywords to specify a character to delimit the values in the macro variable.
- The QUOTE function adds double quotation marks to a character value.
- The STRIP function returns a character string with all leading and trailing blanks removed.

Demo

1. Open the **s106d02.sas** program in the **demos** folder and find the **Demo** section. Run the code in **Step 1: Create Macro Variables**. Examine the results and log to see the values of the newly created macro variables **&Division** and **&StateList**.

```
%let Division=3;

proc sql;
select Name
    into :StateList SEPARATED BY ","
    from sq.statepopulation
    where Division = "&Division";
quit;
%put &Division;
%put &StateList;
```

2. Discuss the query under **Step 2: Use Macro Variables**. The table being created will end with the value of the macro variable **&Division**. The **customer** table will attempt to be subset by a list of values from the **&StateList** macro variable. Run the query and examine the errors.

Note: The name of the new table will end with the value of the macro variable.

```
options symbolgen;
proc sql;
create table division&Division as
select *
  from sq.customer
  where State in (&StateList);
quit;
options nosymbolgen;
```

3. Move back to **Step 1**. Add the QUOTE function around the **Name** column. Run the query and %PUT statements. Examine the results and log.

```
proc sql;
select quote(Name)
  into :StateList SEPARATED BY ","
  from sq.statepopulation
  where Division = "&Division";
quit;
%put &=Division;
%put &=StateList;
```

4. Add the STRIP function inside the QUOTE function. Run the query and %PUT statements. Examine the results and log.

```
proc sql;
select quote(strip(Name))
  into :StateList SEPARATED BY ","
  from sq.statepopulation
  where Division = "&Division";
quit;
%put &=Division;
%put &=StateList;
```

5. Move to **Step 2**. Run the query. Examine the results and log.

```
options symbolgen;
proc sql;
create table division&Division as
select *
  from sq.customer
  where State in (&StateList);
quit;
options nosymbolgen;
```

```

42      proc sql;
43      create table division&Division as
SYMBOLGEN: Macro variable DIVISION resolves to 3
44      select *
45      from sq.customer
46      where State in (&StateList);
SYMBOLGEN: Macro variable STATELIST resolves to "IL","IN","MI","OH","WI"
NOTE: Table WORK.DIVISION3 created, with 16022 rows and 22 columns.

```

6. Move to **Step 1**. Change the value in the %LET statement to 9 and add the NOPRINT option in the PROC SQL statement to finalize the program. Run the entire program. Examine the results and log.

```

%let Division=9;

proc sql noprint;
...
```

```

51      options symbolgen;
52      proc sql;
53      create table division&Division as
SYMBOLGEN: Macro variable DIVISION resolves to 9
54      select *
55      from sq.customer
56      where State in (&StateList);
SYMBOLGEN: Macro variable STATELIST resolves to "AK","CA","HI","OR","WA"
NOTE: Table WORK.DIVISION9 created, with 22296 rows and 22 columns.

```

End of Demonstration

Using Formats when Creating Macro Variables

FORMAT=\$QUOTEw.

```
proc sql noprint;
select Name format=$quote4.
  into :StateList SEPARATED BY ", "
  from sq.statepopulation
  where Division = '3';
quit;

%put &=StateList;
```

To ensure that values are not truncated,
you should specify a width as long as the
longest value +2.

→ STATELIST=" IL ", " IN ", " MI ", " OH ", " WI "

34



Strings larger than six characters are truncated when you use the QUOTE format. To ensure that values are not truncated, you should be sure to specify a width as long as the longest value +2. When using other formats, be sure to check the SAS documentation.

Syntax Summary

```
SELECT col-name, col-name
  INTO :macvar1, :macvar2 <TRIMMED>
  FROM table;
```

Create Data-Driven Macro Variables



```
SELECT col-name
  INTO :macvar1 SEPARATED BY "delimiter"
  FROM table;
```

Concatenate Macro Values

PROC SQL NOPRINT;

Suppress Display

```
OPTIONS SYMBOLGEN | NOSYMBOLGEN
```

Display Resolved Macro Variables

```
%PUT text &=macvar;
```

Display Macro Value

35



Beyond SQL Essentials

What if you want to ...

... learn more about SAS functions

- Take the [SAS Functions by Example](#) course.
- Read [SAS Functions by Example](#).
- View the [SAS Function Documentation](#).
- Create your own functions with [PROC FCMP](#).

... learn more about the SAS macro language

- Take the [SAS Macro Language 1: Essentials](#) course.
- Read [SAS Macro Programming Made Easy](#).
- View the [SAS 9.4 Macro Language: Reference Documentation](#).

... learn more about using the SAS macro language and SQL

- Visit the **SAS Macro Language** section on the [ELP](#) for a list of SAS papers and blogs.



Practice

Level 1

1. Creating a Macro Variable from an SQL Query

Using the **sq.statepopulation** table, write a program that dynamically returns states in a specified region that have a three-year estimated population change greater than the median population change for the entire region.

- a. Write a query to create the macro variable **MedianEst** to store the value of the median **nPopChg3** of Region 1.
 - 1) Calculate the median of **nPopChg3**.
 - 2) Use the INTO clause to create a macro variable named **MedianEst**. Use the TRIMMED keyword.
 - 3) Use the **sq.statepopulation** table.
 - 4) Filter the results for rows in the **Region** column with the value of 1. The value 1 is character.
 - 5) Add the NOPRINT option in the PROC SQL statement.
 - 6) Below the query view the new macro variable in the log using %PUT.

```
%put &=MedianEst;
```

- 7) Run the query and compare your log.

49	%put &=MedianEst; MEDIANEST=3341
----	-------------------------------------

- b. Write another query to select states and their three-year population change in Region 1 that have a higher estimate than the median of all states in that region.
 - 1) Select the **Name** and **nPopChg3** columns. Format **nPopChg3** so that values are displayed with commas.
 - 2) Use the **sq.statepopulation** table.
 - 3) Filter rows for states in region 1 that have an **nPopChg3** value greater than the macro variable **&MedianEst** created in step 1.
 - 4) Order by **nPopChg3** descending.
 - 5) Add the following titles:
 - a) title 1 – **States in Region 1 with a 3-Year Estimated Population Change Greater than the Median**
 - b) title 2 – **Median Estimate: &MedianEst**
 - 6) Run the query and compare your results.

Results

**States in Region 1 with a 3-Year Estimated Population Change Greater than the Median
Median Estimate: 3341**

Name	nPopChg3
MA	38,903
NJ	19,977
PA	16,613
NH	6,691

- c. At the beginning of your program, create a macro variable named **RegionNum** to specify the region to analyze.
 - 1) Use the %LET statement to create the macro variable **RegionNum** and set the value equal to 1.
 - 2) In your program, replace every location of the character value 1 with **&RegionNum**. Make sure that the macro variable is enclosed in double quotation marks.

Hint: You need to do this replacement in the WHERE clause in each query and in the TITLE statement.

 - 3) Run the entire program and confirm that the results are the same as step b.
- d. In the %LET statement replace the value 1 with the value 2. Run the entire program.

Results

**States in Region 2 with a 3-Year Estimated Population Change Greater than the Median
Median Estimate: 15174**

Name	nPopChg3
MN	43,024
IN	31,796
OH	25,313
WI	21,517
MI	19,468
MO	17,840

- e. How many states in Region 2 have a higher three-year estimated population change than the median of all states in that region?

Level 2

2. Creating a Macro Variable with a List of Values from an SQL Query

Write a program that dynamically determines a list of countries in a specified **region**. Then determine the percentage of people who have borrowed from a financial institution or used a credit card (% age 15+), and whether it is increasing or decreasing from the year 1 estimate to the year 3 estimate. Use the **sq.globalmetadata** table for country region information and the **sq.globalindex** table for country financial and population information.

- a. Using **sq.globalmetadata**, write a query to create a macro variable that lists the distinct values of **CountryCode** for a specified region. Ensure that the values are enclosed in quotation marks and separated by commas.
- 1) Create a macro variable named **RegionValue** with the value *South Asia*.
 - 2) Use the **RegionValue** macro variable in a query to create a data-driven macro variable named **Countries** that lists the country codes in the specified region. The value list should be comma separated, and each value should be enclosed in quotation marks.
 - 3) Use the NOPRINT option in the PROC SQL statement.
 - 4) Use %PUT below your query to view the value of the macro variable **Countries**.
 - 5) Run the query and %PUT statement. Compare your results.

Results

```
50      %put &=Countries;
COUNTRIES="AFG", "BGD", "BTN", "IND", "LKA", "NPL", "PAK"
```

- b. Write another query to select the countries in the region that you specified. Convert the whole numbers to percentages, and determine whether the estimated value is increasing, decreasing, or unknown using the **sq.globalfindex** table.
- 1) Select the **CountryCode** and **IndicatorName** columns. Create three new columns.
 - a) First divide **EstYear1** by 100 and name the column **EstYear1PCT**. Format using percentages.
 - b) Follow the previous step for **EstYear3**.
 - c) Use the CASE expression to determine whether the value is *Increasing*, *Decreasing*, or *Unknown*. The value is *Unknown* when the value is null. Name the column **Forecast**.

Hint: In the CASE expression, make sure that the first WHEN tests whether a value is missing.
 - 2) Filter rows by **CountryCode** using the **Countries** macro variable and whether **IndicatorName** is equal to *Borrowed from a financial institution or used a credit card (% age 15+)*.
 - 3) Order by **Forecast**.
 - 4) Add the title **Countries in &RegionValue**.
 - 5) Run the query and compare the results.

Results

Countries in South Asia					
CountryCode	IndicatorName		EstYear1PCT	EstYear3PCT	Forecast
AFG	Borrowed from a financial institution or used a credit card (% age 15+)		4.49%	3.81%	Decreasing
BGD	Borrowed from a financial institution or used a credit card (% age 15+)		10.0%	9.21%	Decreasing
LKA	Borrowed from a financial institution or used a credit card (% age 15+)		19.7%	17.4%	Decreasing
IND	Borrowed from a financial institution or used a credit card (% age 15+)		9.12%	8.15%	Decreasing
PAK	Borrowed from a financial institution or used a credit card (% age 15+)		1.57%	2.64%	Increasing
NPL	Borrowed from a financial institution or used a credit card (% age 15+)		12.0%	13.6%	Increasing
BTN	Borrowed from a financial institution or used a credit card (% age 15+)		4.21%	.	Unknown

- c. Change the **RegionValue** macro variable at the beginning of the program to *North America*. Run the program.

Results

Countries in North America					
CountryCode	IndicatorName		EstYear1PCT	EstYear3PCT	Forecast
CAN	Borrowed from a financial institution or used a credit card (% age 15+)		76.5%	82.8%	Increasing
USA	Borrowed from a financial institution or used a credit card (% age 15+)		64.6%	68.4%	Increasing

- d. What is the forecast when **CountryCode** is *CAN*?

3. Creating Dynamic Titles with Formats

Write a program that dynamically determines the total number of customers in specific states and uses that value to calculate the percentage of customers in a ZIP code in a specific state. Use the **sq.customer** table.

- a. Write a query using a macro variable to count the total number of customers in a specified state.
- 1) Use the %LET statement to create a macro variable named **StateValue**. Set the value equal to *NC*.
 - 2) Use the macro variable **StateValue** to subset the table and count the number of customers in that state. Store the number in a macro variable named **TotalCust**. Use the TRIMMED option.
 - 3) Use the NOPRINT option and view the value of **TotalCust** in the log using %PUT.
 - 4) Run the program and compare the results.

```
51           %put &=TotalCust;
TOTALCUST=2362
```

- b. Write another query to calculate the percentage of customers in each ZIP code for the specified state.
- 1) Select **Zip** and calculate the percentage of customers in a ZIP code by counting the number of customers with that **Zip** value and dividing it by the **TotalCust** macro variable. Name the new column **PctZip** and format it using a percent.
 - 2) Filter the rows by **State** using the macro variable **StateValue**.

- 3) Group the results by **Zip**.
- 4) Order by **PctZip** descending.
- 5) Add dynamic titles using the following:

```
title "Total Customers in &StateValue:  
      %left(%qsysfunc(putn(&TotalCust,comma15.)))";  
title2 "Percentage of Customers by Zip";  
title3 "Report Created on %left(%qsysfunc(today(),weekdate.))";
```

- 6) Run the program and compare the results.

Partial Results

Total Customers in NC: 2,362	
Percentage of Customers by Zip	
Report Created on Sunday, May 19, 2019	
Zip	PctZip
28201	16.43%
27601	8.89%
27006	6.31%
27401	5.63%
27701	5.55%
.....

- c. Change the value in the macro variable **State** from *NC* to *TX*. Run the program and compare the results.

Partial Results

Total Customers in TX: 9,893	
Percentage of Customers by Zip	
Report Created on Sunday, May 19, 2019	
Zip	PctZip
77001	13.87%
75201	7.79%
78201	7.65%
73301	7.17%
76101	4.45%

- d. In Texas, which **Zip** value has the highest percentage of customers?

Challenge

4. Splitting One Table into Many

Write a program that dynamically creates a new table for each distinct value in a column. Use the **Region** column in the **sq.globalmetadata** table.

Hint: Visit the Extended Learning page and view the SAS blog **How to split one data set into many** in the **SAS Macro Language section** or follow the direct link:

<https://blogs.sas.com/content/sasdumy/2015/01/26/how-to-split-one-data-set-into-many/>

How many tables were created?

End of Practices

6.3 Solutions

Solutions to Practices

1. Creating a Macro Variable from an SQL Query

```
/*s106s01.sas*/
/*a*/
proc sql noprint;
select median(nPopChg3)
   into :MedianEst trimmed
   from sq.statepopulation
   where Region="1";
quit;
%put &=MedianEst;

/*b*/
title "States in Region 1 with a 3-Year Estimated Population
      Change Greater than the Median";
title2 "Median Estimate: &MedianEst";
proc sql;
select Name,nPopChg3 format=comma14.
   from sq.statepopulation
   where Region="1" and
         nPopChg3 > &MedianEst
   order by nPopChg3 desc;
quit;
title;

/*c*/
%let RegionNum=2;

proc sql noprint;
select median(nPopChg3)
   into :MedianEst trimmed
   from sq.statepopulation
   where Region="&RegionNum";
quit;
%put &=MedianEst;

title "States in Region &RegionNum with a 3-Year Estimated
      Population Change Greater than the Median";
title2 "Median Estimate: &MedianEst";
proc sql;
select Name,nPopChg3 format=comma14.
   from sq.statepopulation
   where Region="&RegionNum" and
         nPopChg3 > &MedianEst
```

```

    order by nPopChg3 desc;
quit;
title;

```

How many states in Region 2 have a higher three-year estimated population change than the median of all states in that region? **Six**

2. Creating a Macro Variable with a List of Values from an SQL Query

```

/*s106s02.sas*/

/*a*/
%let RegionValue=South Asia;
proc sql noprint;
select quote(strip(CountryCode))
   into :Countries separated by ","
   from sq.globalmetadata
   where Region="&RegionValue";
quit;
%put &=Countries;

/*b*/
title "Countries in &Region";
proc sql;
select CountryCode, IndicatorName,
   EstYear1/100 as EstYear1PCT format=percent7.2,
   EstYear3/100 as EstYear3PCT format=percent7.2,
   case
      when (EstYear1 is null or EstYear3 is null) then "Unknown"
      when calculated EstYear1PCT < calculated EstYear3PCT
           then "Increasing"
      when calculated EstYear1PCT > calculated EstYear3PCT
           then "Decreasing"
      else "No Change"
   end as Forecast
   from sq.globalindex
   where CountryCode in (&Countries) and
         IndicatorName="Borrowed from a financial institution or
                     used a credit card (% age 15+)"
   order by Forecast;
quit;
title;

/*c*/
%let Region=North America;
proc sql noprint;
select quote(strip(CountryCode))
   into :Countries separated by ","
   from sq.globalmetadata
   where Region="&Region";
quit;

```

```
%put &=Countries;

title "Countries in &Region";
proc sql;
select CountryCode, IndicatorName,
       EstYear1/100 as EstYear1PCT format=percent7.2,
       EstYear3/100 as EstYear3PCT format=percent7.2,
       case
           when (EstYear1 is null or EstYear3 is null) then "Unknown"
           when calculated EstYear1PCT < calculated EstYear3PCT
               then "Increasing"
           when calculated EstYear1PCT > calculated EstYear3PCT
               then "Decreasing"
           else "No Change"
       end as Forecast
from sq.globalindex
where CountryCode in (&Countries) and
      IndicatorName="Borrowed from a financial institution or
                     used a credit card (% age 15+)"
order by Forecast;
quit;
title;
```

What is the forecast when **CountryCode** is CAN? **Increasing**

3. Creating Dynamic Titles with Formats

```
/*s106s03.sas*/
/*a*/
%let StateValue=NC;
proc sql noprint;
select count(*) as Total
      into :TotalCust trimmed
      from sq.customer
      where State=&StateValue";
quit;
%put &=TotalCust;

/*b*/
title "Total Customers in &StateValue:
      %left(%qsysfunc(putn(&TotalCust,comma15.)))";
title2 "Percentage of Customers by Zip";
title3 "Report Created on %left(%qsysfunc(today(),weekdate.))";
proc sql;
select Zip, count(*)/&TotalCust as PctZip format=percent8.2
      from sq.customer
      where State=&StateValue"
      group by Zip
      order by PctZip desc;
quit;
```

```

title;

/*c*/
%let StateValue=TX;

proc sql noprint;
select count(*) as Total
  into :TotalCust trimmed
  from sq.customer
  where State=&StateValue";
quit;
%put &=TotalCust;

title "Total Customers in &StateValue:
      %left(%qsysfunc(putn(&TotalCust,comma15.))) ";
title2 "Percentage of Customers by Zip";
title3 "Report Created on %left(%qsysfunc(today(),weekdate.)) ";
proc sql;
select Zip, count(*)/&TotalCust as PctZip format=percent8.2
  from sq.customer
  where State=&StateValue"
  group by Zip
  order by PctZip desc;
quit;
title;

```

In Texas, which **Zip** value has the highest percentage of customers? **77001**

4. Splitting One Table into Many

```

/*s106s04.sas*/

%let table=sq.globalmetadata;
%let column=Region;

proc sql noprint;
select distinct
  cat("DATA out_",
      compress(&column.,,'kad'),
      "; set &TABLE.(where=(&COLUMN.='"', &column.,"' )); run;");
      length=500
  into :allsteps separated by ';'
  from &table.;
quit;

%macro runSteps;
  &allsteps.;
%mend;

%runSteps;

```

How many tables were created? **Eight**

End of Solutions

Solutions to Activities and Questions

continued...

6.01 Activity – Correct Answer

```
1. %let State=GA;  
%let CreditMin=650;  
  
proc sql;  
create table Customer&State as  
select CustomerID, Employed, Race,  
       Married, State, CreditScore  
     from sq.customer  
   where State="&State" and  
         CreditScore > &CreditMin;  
quit;
```

NOTE: Table WORK.CUSTOMERGA created, with 957 rows and 6 columns.



The new table contains rows where **State** equals **GA** and **CreditScore** is greater than **650**.

sas

continued...

6.01 Activity – Correct Answer

2. What is the name of the newly created table? The table name is CUSTOMERNC. How many rows? 700 rows

```
%let State=NC;  
%let CreditMin=700;  
  
proc sql;  
create table Customer&State as  
select CustomerID, Employed, Race,  
       Married, State, CreditScore  
  from sq.customer  
 where State="&State" and  
       CreditScore > &CreditMin;  
quit;
```

Change the values as necessary.

The new table contains rows where **State** equals **NC** and **CreditScore** is greater than **700**.



sas

6.01 Activity – Correct Answer

3. How many rows are in the new table? **Zero rows are returned.**

```
%let State=NC;
%let CreditMin=700;

proc sql;
create table Customer&State as
select CustomerID, Employed, Race,
      Married, State, CreditScore
    from sq.customer
   where State='&State' and
         CreditScore > &CreditMin;
quit;
```

The macro variable does not resolve in single quotation marks. The WHERE clause is searching for the character string '&State'.

10

Copyright © SAS Institute Inc. All rights reserved.



continued...

6.02 Activity – Correct Answer

- 1.

```
46      %put &MaxPop &TotalCtry;
MAXPOP=1.1413E9 TOTALCTRY=151
```

When storing a number, SAS uses the BEST8. format. If the number is larger than eight digits, scientific notation is often used.

25

Copyright © SAS Institute Inc. All rights reserved.



continued...

6.02 Activity – Correct Answer

2.

```
select distinct CountryCode, ShortName, Region,
      EstYear3Pop format=comma16.
  from sq.globalfull
 where EstYear3Pop = &MaxPop;
```

1141324637  1.1413E9

The scientific notation value is different from the original and thus might not find a match.

NOTE: No rows were selected.

26

Copyright © SAS Institute Inc. All rights reserved.



continued...

6.02 Activity – Correct Answer

3.

```
proc sql noprint;
select max(EstYear3Pop) format=10.,
       count(distinct CountryCode)
  into :MaxPop trimmed, :TotalCtry trimmed
  from sq.globalfull;
quit;
%put &=MaxPop &=TotalCtry;
```

The FORMAT column modifier formats the stored value of the macro variable.

46 %put &=MaxPop &=TotalCtry;
MAXPOP=1141324637 TOTALCTRY=151

27

Copyright © SAS Institute Inc. All rights reserved.



6.02 Activity – Correct Answer

3. Which country has the largest three-year estimated population? **China**

```
title "Country with the Largest 3 Year Estimated Population";
title2 "Out of &TotalCtry Countries";
proc sql;
select distinct CountryCode, ShortName, Region,
   EstYear3Pop format=comma16.
   from sq.globalfull
   where EstYear3Pop = &MaxPop;
quit;
title;
```

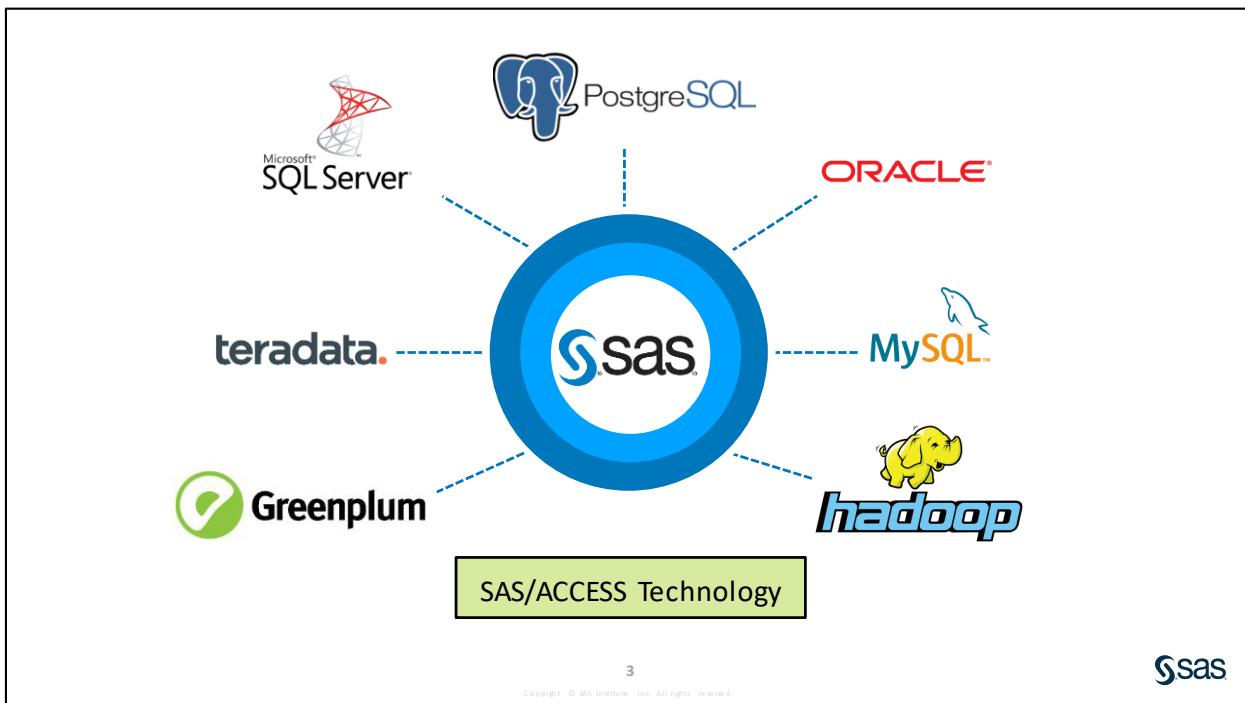
Country with the Largest 3 Year Estimated Population
Out of 151 Countries

CountryCode	ShortName	Region	EstYear3Pop
CHN	China	East Asia & Pacific	1,141,324,637

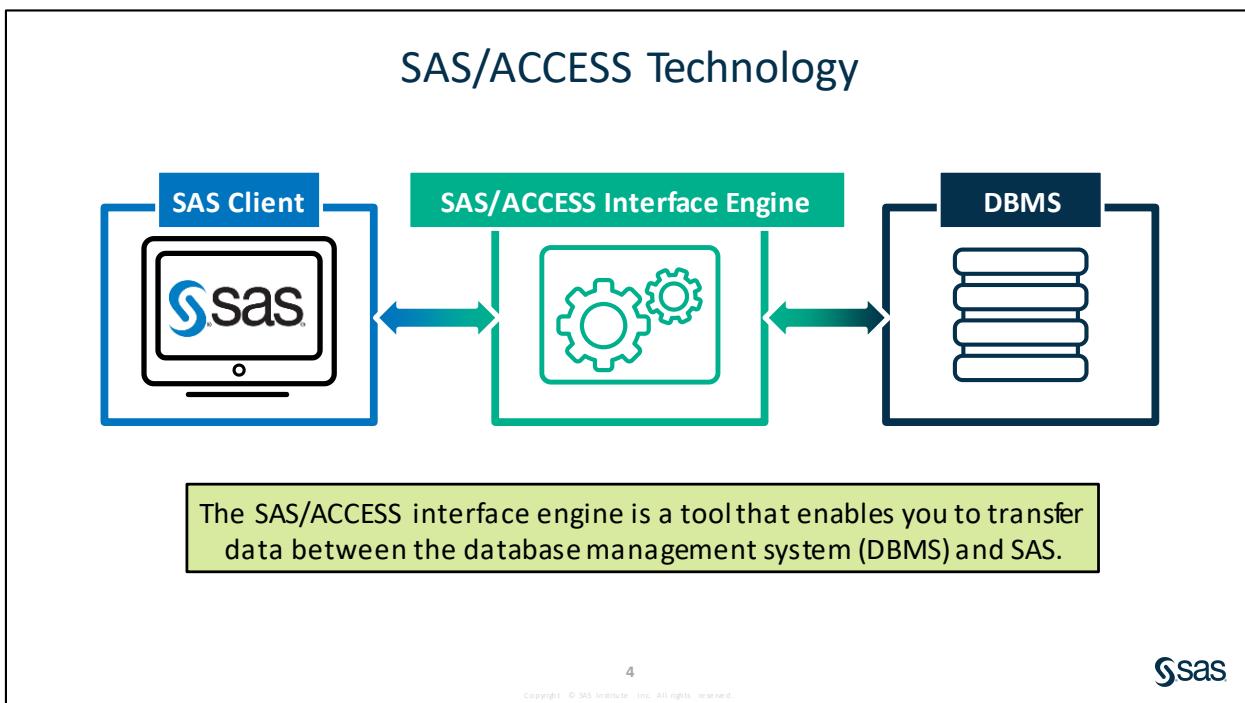
Lesson 7 Accessing DBMS Data with SAS/ACCESS®

7.1 Overview of SAS/ACCESS Technology	7-3
7.2 SQL Pass-Through Facility	7-6
Demonstration: Using an SQL Pass-Through Query	7-13
7.3 SAS/ACCESS LIBNAME Statement	7-18
Demonstration: Using the SAS/ACCESS LIBNAME Statement.....	7-20
7.4 FEDSQL Procedure.....	7-27
7.5 Solutions	7-39
Solutions to Activities and Questions.....	7-39

7.1 Overview of SAS/ACCESS Technology



SAS provides data access to more of your data sources so that you can make better decisions faster. These interfaces are out-of-the-box solutions that provide enterprise data access and integration between SAS and third-party databases. SAS/ACCESS interfaces enable your SAS solutions to read, write, and update data no matter what native databases or platforms you use.



For example, if you need to read data in Teradata, there is SAS/ACCESS Interface to Teradata. If you need to read Oracle, there is SAS/ACCESS Interface to Oracle.

Considerations when Working with DBMS

The diagram illustrates the interaction between SAS and a Database Management System (DBMS). A blue-bordered box contains a SAS logo icon. An arrow points from this icon to a second box containing a server rack icon. Another arrow points from the server rack icon back to the SAS icon. Below this, a green callout box contains the text: "For improved efficiency, process as much as possible inside the DBMS to limit data movement."

5

Copyright © SAS Institute Inc. All rights reserved.

There are many factors affecting the optimization of your programs when working with DBMSs. The best advice is to read the SAS/ACCESS documentation for your specific DBMS. Then try multiple solutions to a program and benchmark the programs to see which program is running more efficiently.

Connecting to DBMS

The diagram compares two methods for connecting SAS to a DBMS:

- SQL Pass-Through:** Shows a SAS icon connected directly to a database icon. A callout box below it states: "Code in the *native DBMS* syntax".
- LIBNAME Statement:** Shows a SAS icon connected to a database icon via a middle box labeled "Gears". A callout box below it states: "Enables SAS to *translate* SAS SQL to *native DBMS SQL*".

A speech bubble from a cartoon head says: "You must have the correct **SAS/ACCESS software** installed for your DBMS!"

6

Copyright © SAS Institute Inc. All rights reserved.

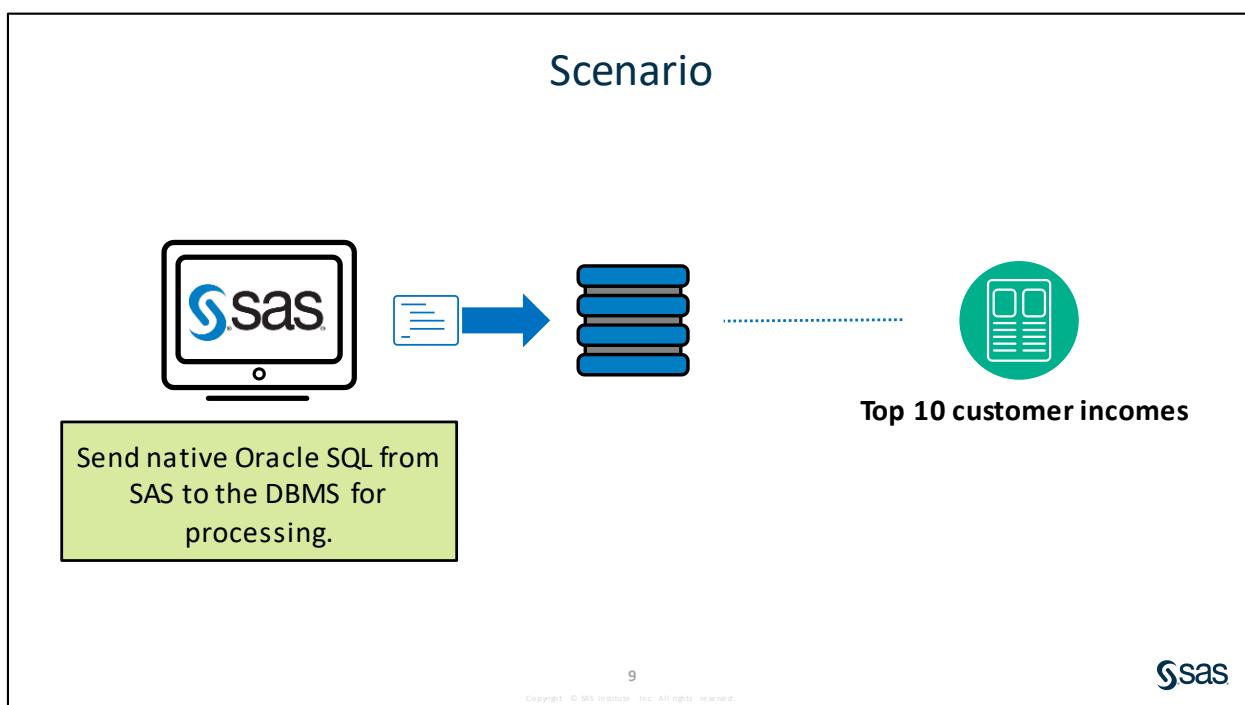
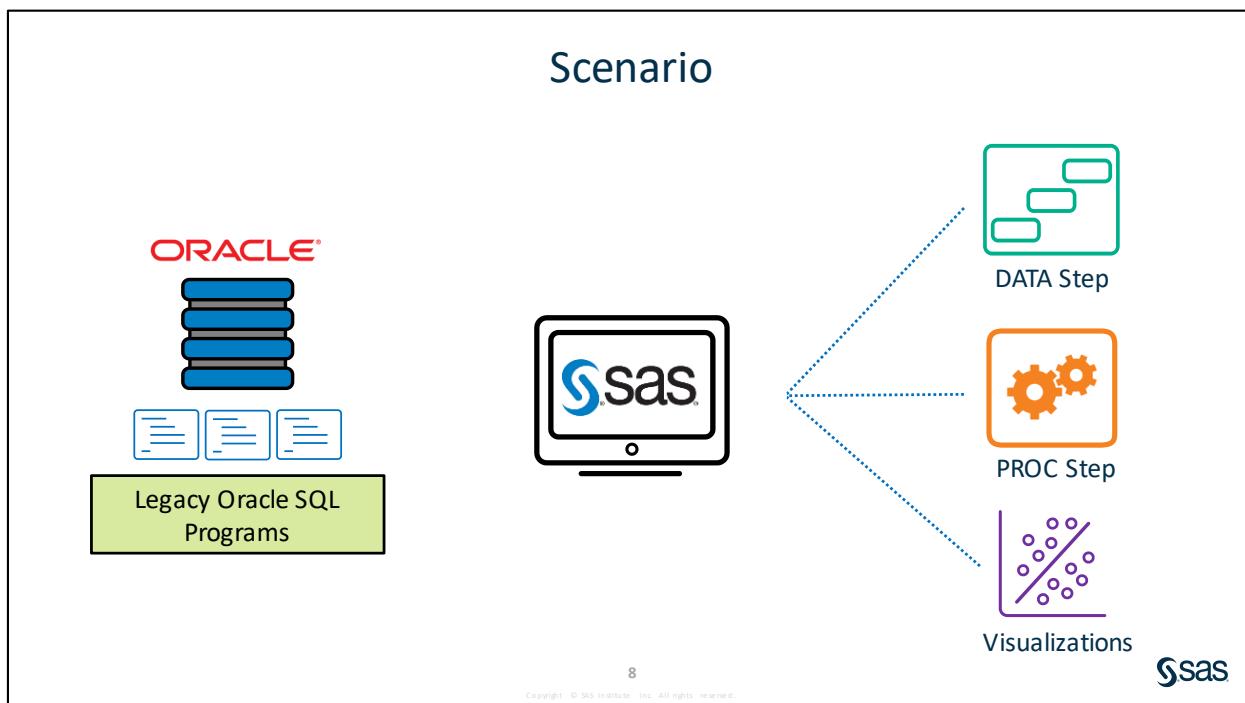
SQL pass-through is known as *explicit pass-through*. The LIBNAME statement is known as *implicit pass-through*.

The PRODUCT_STATUS procedure returns a list of the SAS Foundation products that are installed on your system, along with the version numbers of those products. It provides a quick method to determine whether a SAS product is available for your use. The results from PROC PRODUCT_STATUS are returned to the SAS log.

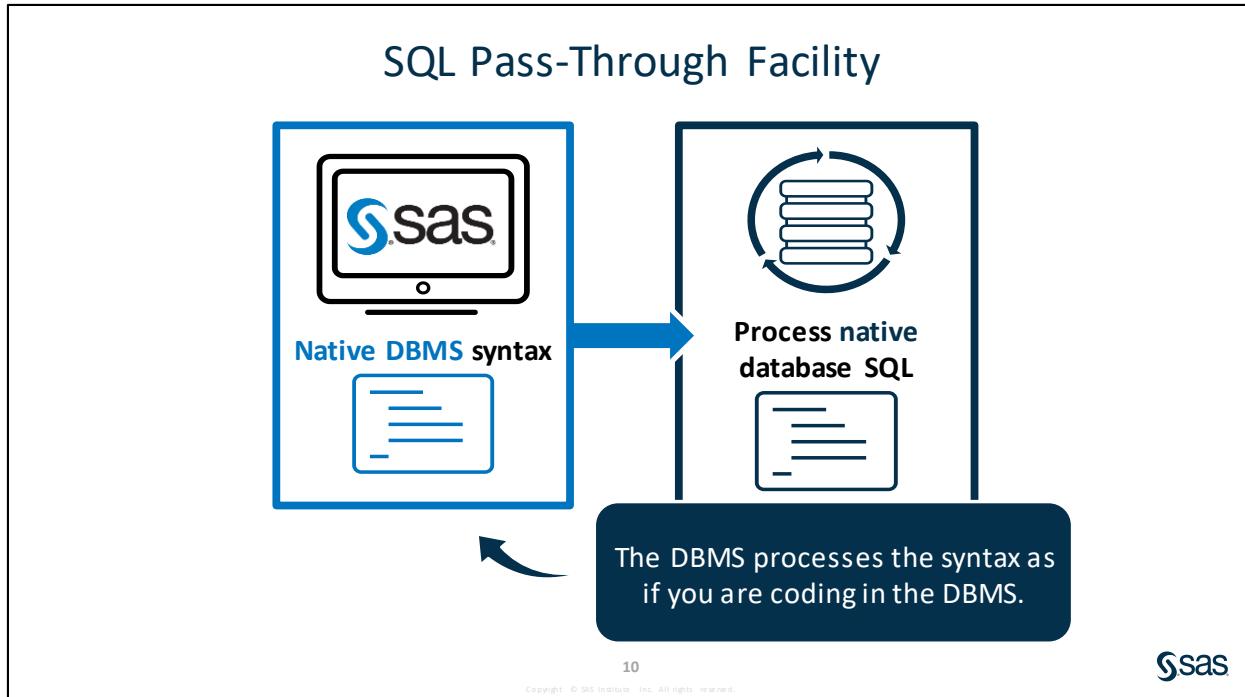
```
PROC PRODUCT_STATUS;  
RUN;
```

For more information about the PRODUCT_STATUS procedure, see "PRODUCT_STATUS Procedure" in the *SAS® 9.4 Procedures Guide, Seventh Edition* documentation. You can also find the direct link in the Course Links section on the ELP.

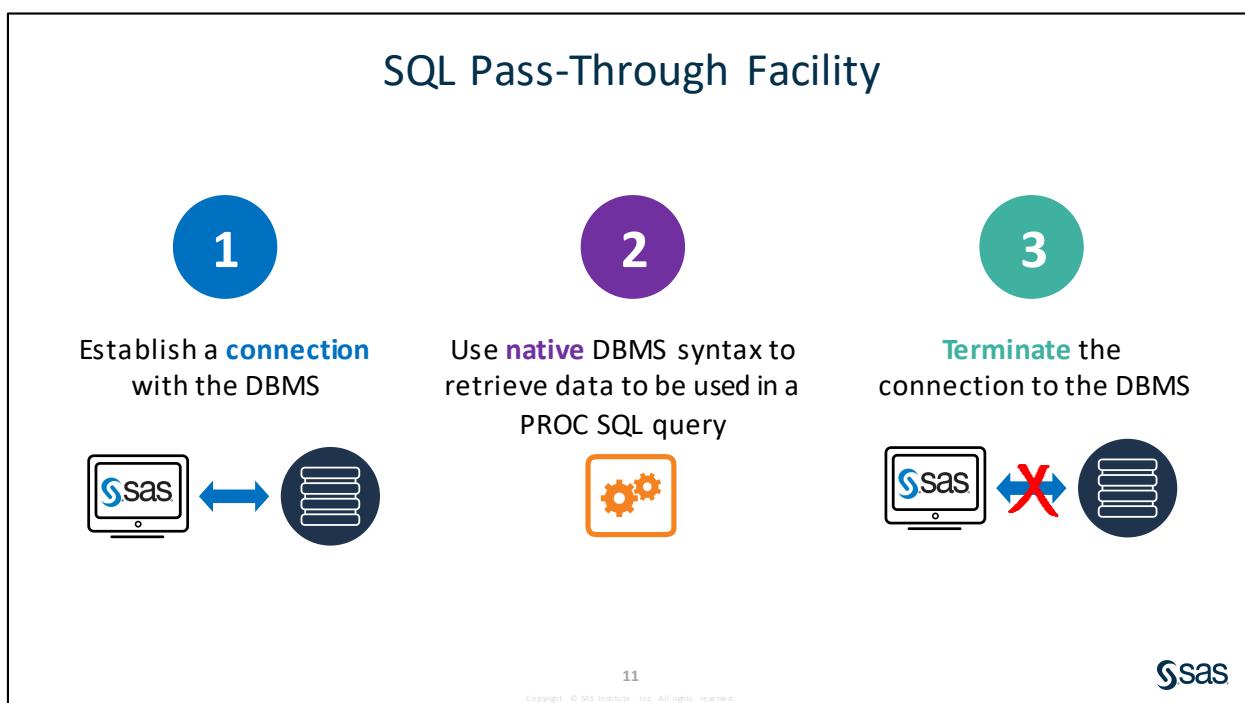
7.2 SQL Pass-Through Facility

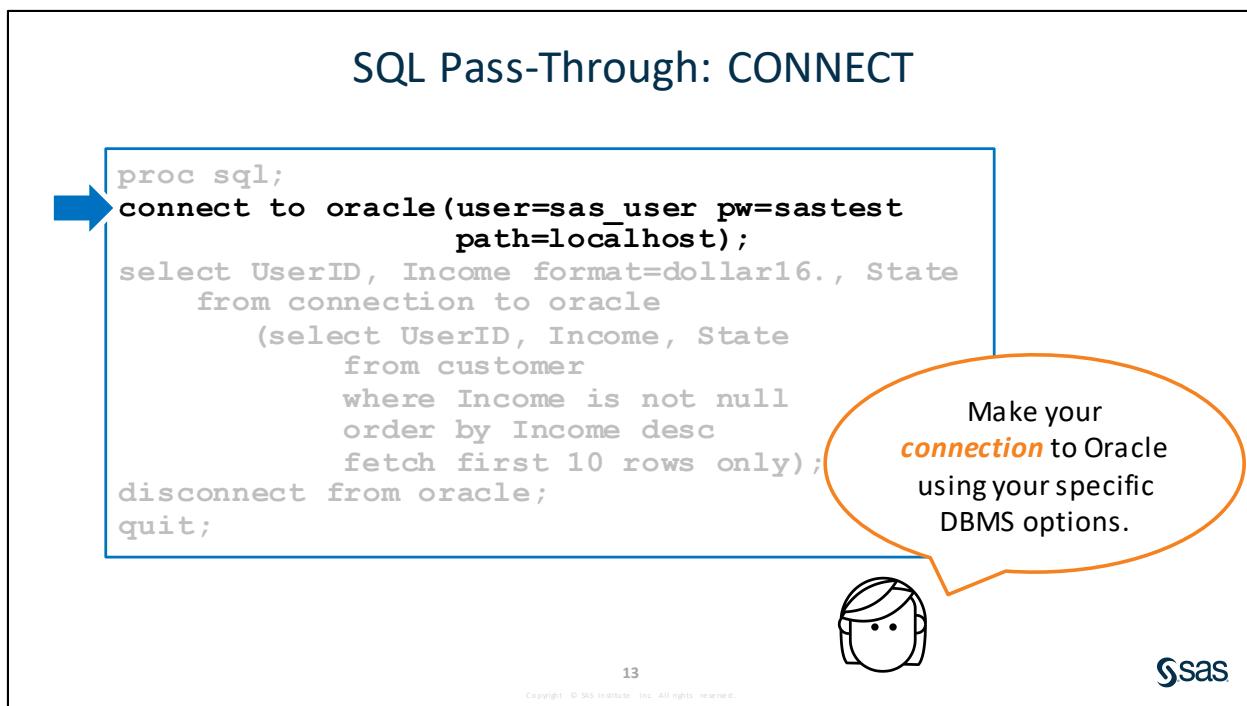
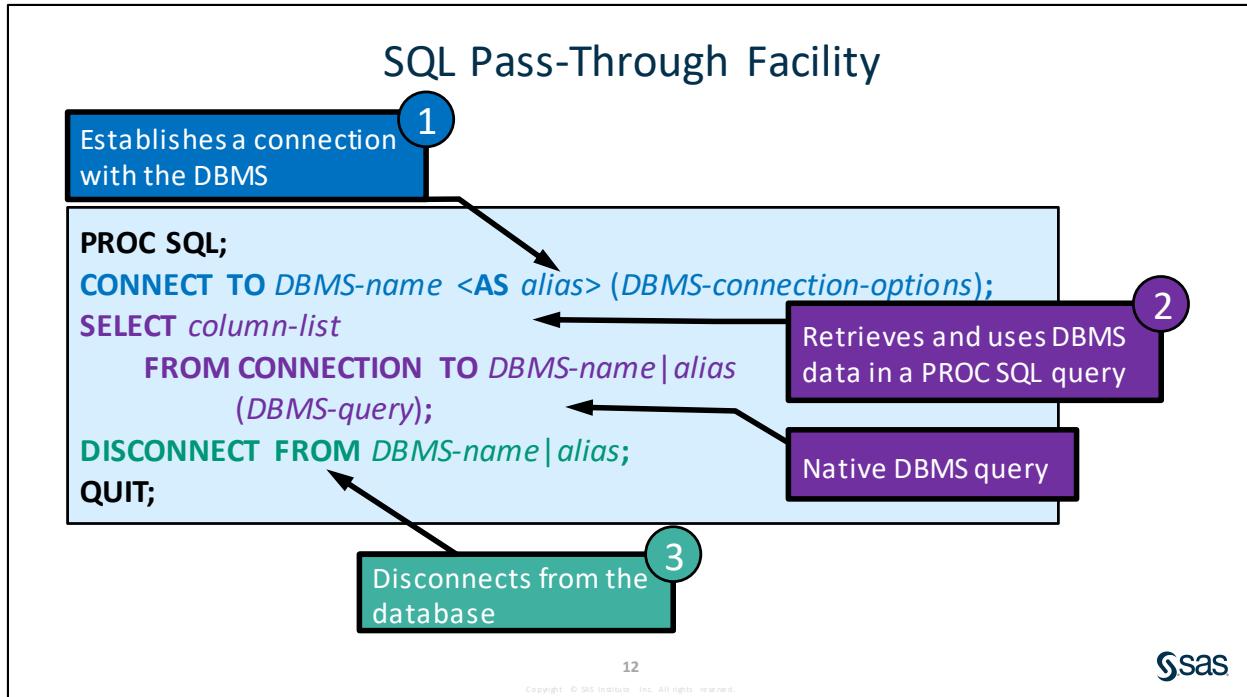


The customer data resides in your company's Oracle database, and you are already familiar with native Oracle SQL. You would like to take native Oracle SQL from SAS to the database for processing.



The SQL pass-through facility enables you to send DBMS-specific SQL statements directly to a DBMS for execution. The pass-through facility uses a SAS/ACCESS interface engine to connect to the DBMS. Therefore, you must have SAS/ACCESS software installed for your DBMS.





Here is what the CONNECT statement does:

- establishes the connection to the DBMS
- specifies the DBMS name
- provides connection information specific to the DBMS
- specifies an alias for the connection (optional)

Option	Specifies
USER=	Oracle user name
PW=	Oracle password associated with the Oracle user name
PATH=	Oracle driver, node, and database, or a database alias

For specific DBMS connection information, view the SAS documentation. For specific values of connectivity options or table names, contact your DBMS administrator.

SQL Pass-Through: SAS SELECT

```
proc sql;
connect to oracle(user=sas_user pw=sastest
                  path=localhost);
select UserID, Income format=dollar16., State
  from connection to oracle
        (select UserID, Income, State
         from customer
         where Income is not null
         order by Income desc
         fetch first 10 rows only);
disconnect from oracle;
quit;
```




The SELECT statement in PROC SQL **returns** results from the native Oracle query.



The SELECT statement in PROC SQL consists of the following:

- a SAS SQL SELECT expression
- a FROM CONNECTION TO component
- SQL code to be passed to the DBMS

You can use SAS features such as functions, aliases, formats, and labels to customize the pass-through query results.

SQL Pass-Through: DBMS Query

```

proc sql;
connect to oracle(user=sas_user pw=sastest
                  path=localhost);
select UserID, Income format=dollar16., State
  from connection to oracle
  (select UserID, Income, State
   from customer
   where Income is not null
   order by Income desc
   fetch first 10 rows only);
disconnect from oracle;
quit;

```



The query is processed by Oracle.



Native Oracle SQL syntax is sent to the **DBMS**.



15

Copyright © SAS Institute Inc. All rights reserved.

This query selects the Oracle DBMS customer table and is executed directly in Oracle. The DBMS processes the syntax as if you are coding directly in the DBMS. The DBMS results are read by SAS and processed based on the SELECT statement. You cannot use features specific to SAS inside the parentheses. Everything inside the parenthesis is SQL that is native to the Oracle DBMS.

When using a different schema from the default, the database table in the pass-through query is specified with the syntax *schema.table_name*.

The FETCH FIRST *n* ROWS ONLY syntax was used in Oracle Database Express Edition Release 18.4.0.0.0 (18c).

SQL Pass-Through: DISCONNECT

```
proc sql;
connect to oracle(user=sas_user pw=sastest
path=localhost);
select UserID, Income format=dollar16., State
from connection to oracle
(select UserID, Income, State
from customer
where Income is not null
order by Income desc
fetch first 10 rows only);
disconnect from oracle;
quit;
```

It's good practice to
disconnect from the
DBMS.



Sas

16

Copyright © SAS Institute Inc. All rights reserved.

You can close the connection to the DBMS by using one of the following methods:

- submitting a DISCONNECT statement
- terminating the SQL procedure (for example, with a QUIT statement)

SQL Pass-Through

```
1 proc sql;
2 connect to oracle(user=sas_user pw=sastest
path=localhost);
3 select UserID, Income format=dollar16., State
from connection to oracle
(select UserID, Income, State
from customer
where Income is not null
order by Income desc
fetch first 10 rows only);
4 disconnect from oracle;
5 quit;
```

Native DBMS query

USERID	INCOME	STATE
marremartinez6531@ismissing.com	\$229,306	IL
kimlihuffman843@fakeemail.com	\$228,165	NY
heacamoredock827@invalid.com	\$220,092	NY
heljaboone8613@invalid.com	\$188,954	NY
... 11 7504261 ...	\$170,372	PA

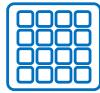
17

Sas

Saving SQL Pass-Through Query Results

CREATE TABLE | VIEW name AS ...;

```
proc sql;
connect to oracle(user=sas_user pw=sastest
path=localhost);
create table work.TotalCustomer as
select UserID, Income format=dollar16., State
from connection to oracle
...query...
```



table



view

18

Copyright © SAS Institute Inc. All rights reserved.



Query results from the SQL pass-through must be saved to a SAS table or view to be used in subsequent SAS processes.

The CREATE VIEW statement stores the pass-through query but does not check the validity of the query. It is important to execute the view after it has been created.



Using an SQL Pass-Through Query

Scenario

Use an SQL pass-through query to a Microsoft Access database using the SAS/ACCESS Interface to PC Files engine.

Files

- **s107d01.sas**
- **customer** – a Microsoft Access table that contains customer information

Syntax

```
PROC SQL;
CONNECT TO DBMS-name <AS alias> (DBMS-connection-options);
SELECT col-name, col-name
FROM CONNECTION TO DBMS-name|alias
(DBMS-query);
DISCONNECT FROM DBMS-name|alias;
QUIT;
```

Notes

- The CONNECT statement establishes the connection to the DBMS.
- The SELECT statement selects the results of the DBMS query.
- The DISCONNECT statement closes the connection to the DBMS.

Demo

1. Open the **s107d01.sas** program in the **demos** folder and find the **Demo** section. If you have not already done so, run the **libname.sas** program to define the **PATH** macro variable.
2. Run the Microsoft Access query. Discuss the syntax error.
3. Add a CONNECT TO PCFILES statement above the SELECT statement. After PCFILES, add the PATH= and DBPASSWORD=SASTEST options in parentheses. End the statement with a semicolon.

```
proc sql;
connect to pcfiles(path="&path/database/SQL_DB.accdb"
                    dbpassword=sastest);
select top 10, UserID, Income, State
  from customer
 order by Income desc;
quit;
```

4. Add the SELECT statement after the CONNECT TO statement, and select all columns using an asterisk. Add the FROM CONNECTION TO component and reference the **pcfiles** database. Enclose the original Microsoft Access query in parentheses.

```
proc sql;
connect to pcfiles(path=&path/database/SQL_DB.accdb"
                     dbpassword=sastest);
select * from connection to pcfiles
  (select top 10, UserID, Income, State
   from customer
   order by Income desc);
quit;
```

5. Add the DISCONNECT FROM statement to disconnect from the **pcfiles** database.

```
proc sql;
connect to pcfiles(path=&path/database/SQL_DB.accdb"
                     dbpassword=sastest);
select * from connection to pcfiles
  (select top 10, UserID, Income, State
   from customer
   order by Income desc);
disconnect from pcfiles;
quit;
```

6. Run the SQL pass-through query and view the results.

UserID	Income	State
marremartinez6531@ismissing.com	229306	IL
kimlihuffman843@fakeemail.com	228165	NY
heacamoredock827@invalid.com	220092.2	NY
heljaboone8613@invalid.com	188953.8	NY
iamheerskin7521@fakeemail.com	172372.7	IA

7. In the SELECT statement, remove the asterisk and add the columns **UserID**, **Income**, and **State**. Format the **Income** column using the DOLLAR16. format. Run the query and view the results.

```
proc sql;
connect to pcfiles(path=&path/database/SQL_DB.accdb"
                     dbpassword=sastest);
select UserID, Income format=dollar16., State
  from connection to pcfiles
  (select top 10, UserID, Income, State
   from customer
   order by Income desc);
disconnect from pcfiles;
quit;
```

UserID	Income	State
marremartinez6531@ismissing.com	\$229,306	IL
kimlihuffman843@fakeemail.com	\$228,165	NY
heacamoredock827@invalid.com	\$220,092	NY
heljaboone8613@invalid.com	\$188,954	NY
jambeerskin7521@fakeemail.com	\$172,373	LA
oranibale6220@ismissing.com	\$160,313	CA

End of Demonstration

7.01 Activity

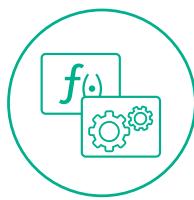
Open **s107a01.sas** from the **activities** folder and perform the following tasks to pass Microsoft Access SQL to the database:

1. Examine the native DBMS query. Run the entire query. Did it produce an error?
2. In the WHERE clause, replace IS MISSING with IS NULL. Run the entire query. Did it run successfully?

SQL Pass-Through Facility Advantages



DBMS can optimize queries.



Use DBMS-specific **functions** and stored **procedures**.



Use native DBMS **code**.

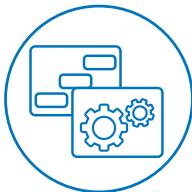


Combine SAS features and DBMS-specific features.

Here are some advantages of using the SQL pass-through facility:

- It enables the DBMS to optimize your queries. When sending your DBMS-specific query, all processing occurs in the database and enables the database to optimize that query.
- It enables you to use specific functions and stored procedures that are DBMS specific when SAS does not have the function that you need, or use a procedure that is already stored on the DBMS.
- If you are familiar with the DBMS syntax, you can use the native syntax inside SAS.
- It enables you to use both the power of SAS and the DBMS and combine the features of both.

SQL Pass-Through Facility Considerations



Query results must be **saved** to use in subsequent **SAS processes**.



DBMS-specific SQL knowledge is **required**.



Only SQL code **within** the parentheses is **passed** to the DBMS.

23

Copyright © SAS Institute Inc. All rights reserved.



Here are some SQL considerations related to the pass-through facility:

- To use the query results from an SQL pass-through in other SAS processes, you must save the results as a SAS table or view.
- You must have DBMS-specific SQL knowledge to use the pass-through facility.
- Only the code within the parentheses is passed to the DBMS. WHERE processing and SORT requests used in procedures that reference a PROC SQL pass-through view must be performed by SAS.

Syntax Summary



```

PROC SQL;
CONNECT TO DBMS-name <AS alias>(DBMS-connection-options);
SELECT column-list
    FROM CONNECTION TO DBMS-name|alias
        (DBMS-query);
DISCONNECT FROM DBMS-name|alias;
QUIT;

```

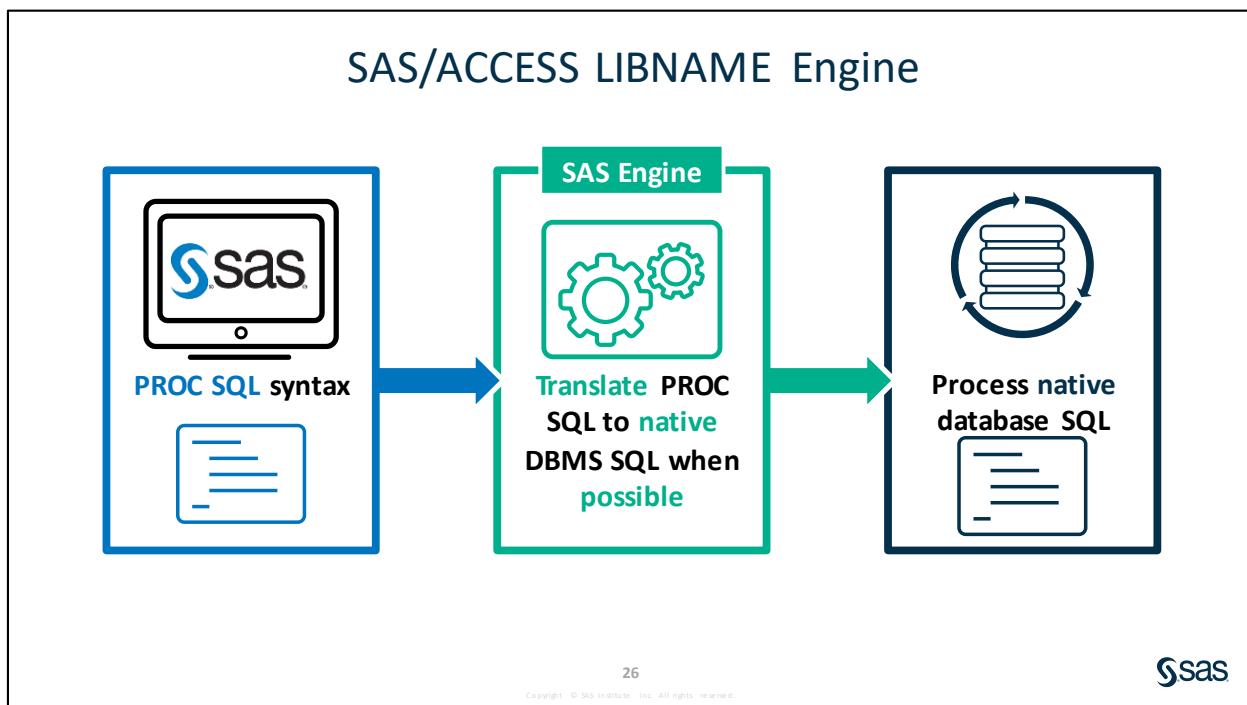
SQL Pass-Through

24

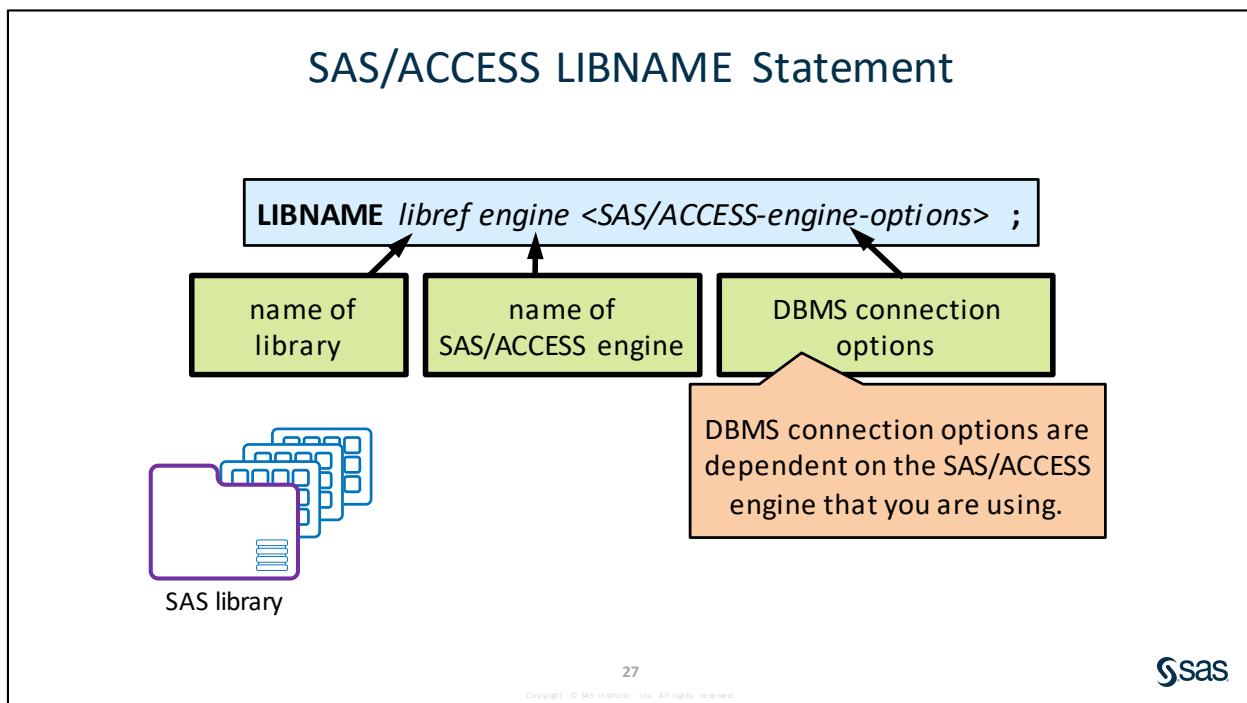
Copyright © SAS Institute Inc. All rights reserved.



7.3 SAS/ACCESS LIBNAME Statement



The SAS/ACCESS LIBNAME engine translates your PROC SQL syntax to native DBMS SQL when possible. This is important because SAS SQL and native DBMS SQL can differ. So if you are working with multiple DBMSs, you can learn SAS SQL instead of learning native DBMS SQL implementations, and let the engine do the work for you.



The SAS/ACCESS LIBNAME statement

- uses the SAS/ACCESS engine to assign a libref to a DBMS
- enables you to reference a DBMS object directly in a DATA step or SAS procedure
- has many options to control the connection to the DBMS.

Closing the DBMS Connection

```
LIBNAME libref CLEAR;
```

Release the DBMS and associated resources.

It's good practice to clear the DBMS connection when complete.



sas

28

Copyright © SAS Institute Inc. All rights reserved.

Submit a LIBNAME statement with a CLEAR option to release the DBMS and associated resources.



Using the SAS/ACCESS LIBNAME Statement

Scenario

Use the SAS/ACCESS LIBNAME statement to read a table in Microsoft Access.

Files

- **s107d02.sas**
- **customer** – a Microsoft Access table that contains customer information

Syntax

```
LIBNAME libref engine <SAS/ACCESS-engine-options>;
LIBNAME libref CLEAR;
```

Notes

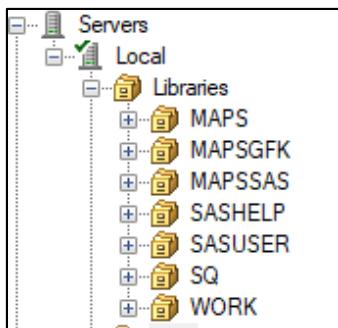
The SAS/ACCESS LIBNAME statement

- uses the SAS/ACCESS engine to assign a libref to a DBMS
- enables you to reference a DBMS object directly in a DATA step or SAS procedure
- has many options to control the connection to the DBMS.

Demo

1. Open the **s107d02.sas** program in the **demos** folder and find the **Demo** section. If you have not already done so, run the **libname.sas** program to define the **PATH** macro variable.
2. Begin by viewing all available libraries in your current session.

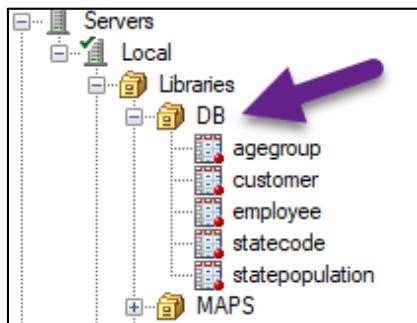
Note: Your libraries might differ. Notice that there is no library named **db**.



3. Complete the LIBNAME statement to define a library named **db** that uses the PCFILES engine. Add the PATH= option to connect to the **SQL_DB.accdb** Microsoft Access database, and add the DBPASSWORD= option using the password **sastest**. Highlight the LIBNAME statement and run the selected code. Use the navigation pane to expand the **db** library.

Note: In SAS Enterprise Guide, click **Libraries** and select **Refresh** to update the library list.

```
libname db pcfiles path=&path/database/SQL_DB.accdb"
          dbpassword=sastest;
```



4. Review the query below the LIBNAME statement. Add the library **db** to the beginning of the **customer** table name, and apply the DOLLAR16. format to the **Income** column.

```
libname db pcfiles path=&path/database/SQL_DB.accdb"
          dbpassword=sastest;

proc sql outobs=10;
select UserID, Income format=dollar16., State
  from db.customer
  order by Income desc;
quit;
```

5. Add a statement to clear the **db** library. Highlight the entire demo program and run the selected code.

```
libname db pcfiles path=&path/database/SQL_DB.accdb"
          dbpassword=sastest;

proc sql outobs=10;
select UserID, Income format=dollar16., State
  from db.customer
  order by Income desc;
quit;

libname db clear;
```

UserID	Income	State
marmartinez6531@ismissing.com	\$229,306	IL
kimlihuffman843@fakeemail.com	\$228,165	NY
heacamoredock827@invalid.com	\$220,092	NY
heliahonne8613@invalid.com	\$188,054	NY

End of Demonstration

SAS/ACCESS LIBNAME Statement Advantages



Fewer lines of SAS code are required.



DBMS-specific SQL knowledge is **not required**.



You have direct **access** to DBMS data.

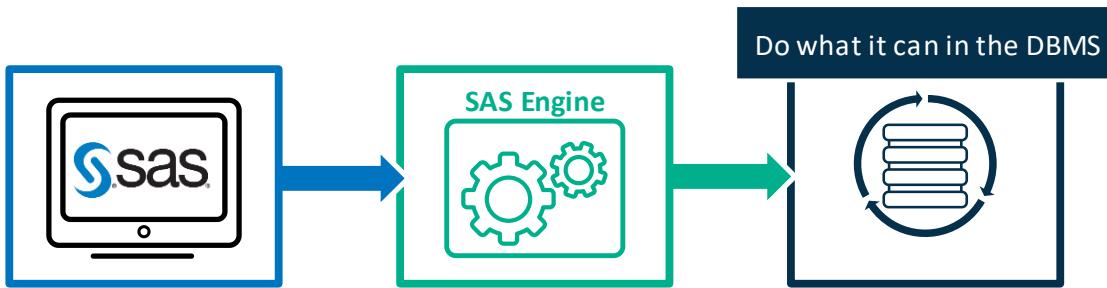


The **LIBNAME** engine **optimizes** processing.

The LIBNAME statement offers the following advantages:

- Significantly fewer lines of SAS code are required to perform operations in your DBMS. For example, a single LIBNAME statement establishes a connection to your DBMS, enables you to specify how your data is processed, and enables you to easily browse your DBMS tables in SAS.
- You do not need to know your DBMS's SQL language to access and manipulate your DBMS data.
- Direct and transparent access to DBMS data. You can use SAS procedures, such as PROC SQL, or DATA step programming on any libref that references DBMS data. You can read, insert, update, delete, and append data, as well as create and drop DBMS tables by using normal SAS syntax. The engine can also translate some SAS processes into the DBMS for processing, which reduces the data that needs to be returned to SAS.
- The LIBNAME engine optimizes the processing of joins and WHERE clauses by passing these operations directly to the DBMS to take advantage of the indexing and other processing capabilities of your DBMS.

SAS/ACCESS LIBNAME Statement Considerations

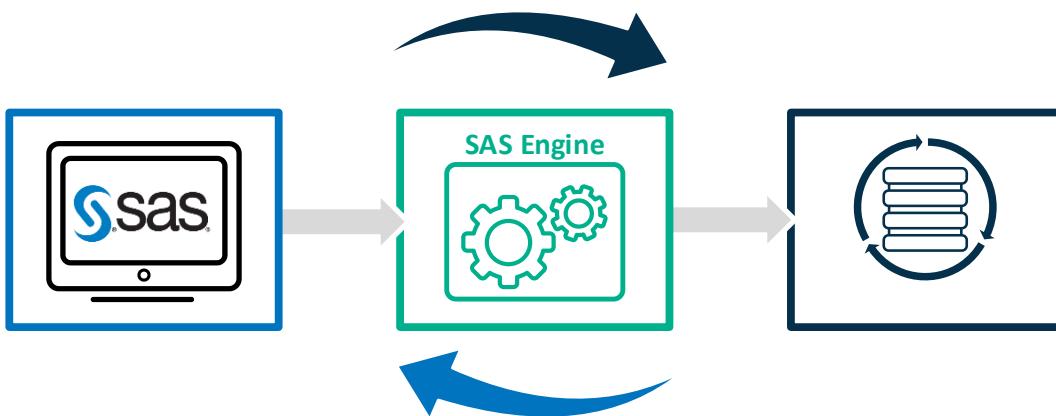


31



Behind the scenes, the SAS/ACCESS engine attempts to convert your SAS code to an SQL query that is passed to the DBMS. This enables as much work to be done in the DBMS as possible.

SAS/ACCESS LIBNAME Statement Considerations



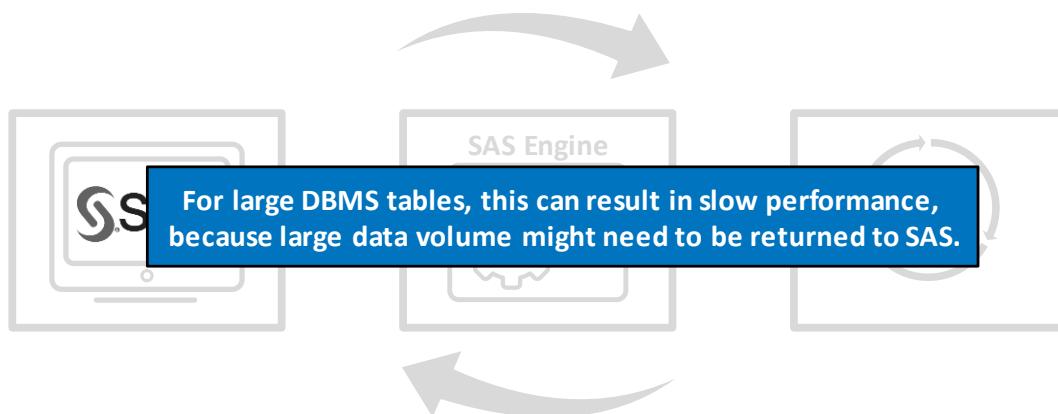
For operations that cannot be converted to DBMS SQL, the SAS/ACCESS engine generates a DBMS query to bring the data into SAS.

32



If SAS fails to pass the query (or parts of the query) to the DBMS, then SAS processes the code. Things like DATA step merge, specific functions, specific data set options, or combining different data sources must be done in SAS. The data is transferred between the two.

SAS/ACCESS LIBNAME Statement Considerations



33



In order for PROC SQL code to be converted to SQL, the code must meet these basic requirements:

- refer to a single SAS/ACCESS LIBNAME engine
- use SAS functions that can be translated into DBMS functions

See the documentation for your specific DBMS for more information.

To examine the SQL that SAS/ACCESS submits, use the following option:

```
OPTIONS SASTRACE=',,d' SASTRACELOC=saslog NOSTSUFFIX;
```

The SASTRACE= option enables you to examine the SQL that the SAS/ACCESS engine submits to the DBMS.

- ',,d' – Specifies that all SQL statements sent to the DBMS are sent to the log.
- ',,s' – Specifies that a summary of timing information for calls made to the DBMS is sent to the log.

The SASTRACELOC= option defines where to send the SASTRACE information.

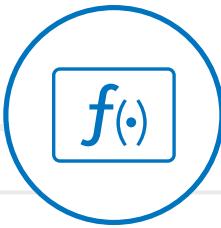
The NOSTSUFFIX option limits the amount of information displayed in the log.

For more information about how to determine where the processing is occurring, visit the SAS documentation or the ELP.

Working with DBMS



SAS **connects**
differently to DBMSs.



Functions are **not consistent** across
DBMSs.



Combining different
data sources must be
done in SAS.

Visit the SAS documentation.

34

Copyright © SAS Institute Inc. All rights reserved.



For more information about DBMS-specific references, see the *SAS/ACCESS® 9.4 for Relational Databases: Reference, Ninth Edition* documentation. You can also find the direct link in the Course Links section on the ELP.

7.02 Activity

Open a browser and perform the following tasks to view SAS DBMS-Specific Reference documentation:

1. In a web browser, access SAS Help at <http://support.sas.com/documentation>.
2. Under **Popular Documentation**, select **Programming: SAS 9.4 and Viya**.
3. Scroll down until you find the **Accessing Data** section. Inside the section, find **SAS/ACCESS** and select **Relational Databases**.
4. Scroll down and find the **DBMS-Specific Reference** section and select a reference of your choice.
5. Review the topics in your DBMS-specific reference. Click the **Passing SAS Functions to** link and review the SAS functions that pass to your DBMS for processing.

35

Copyright © SAS Institute Inc. All rights reserved.



Syntax Summary

LIBNAME *librefengine* <SAS/ACCESS-engine-options> ;

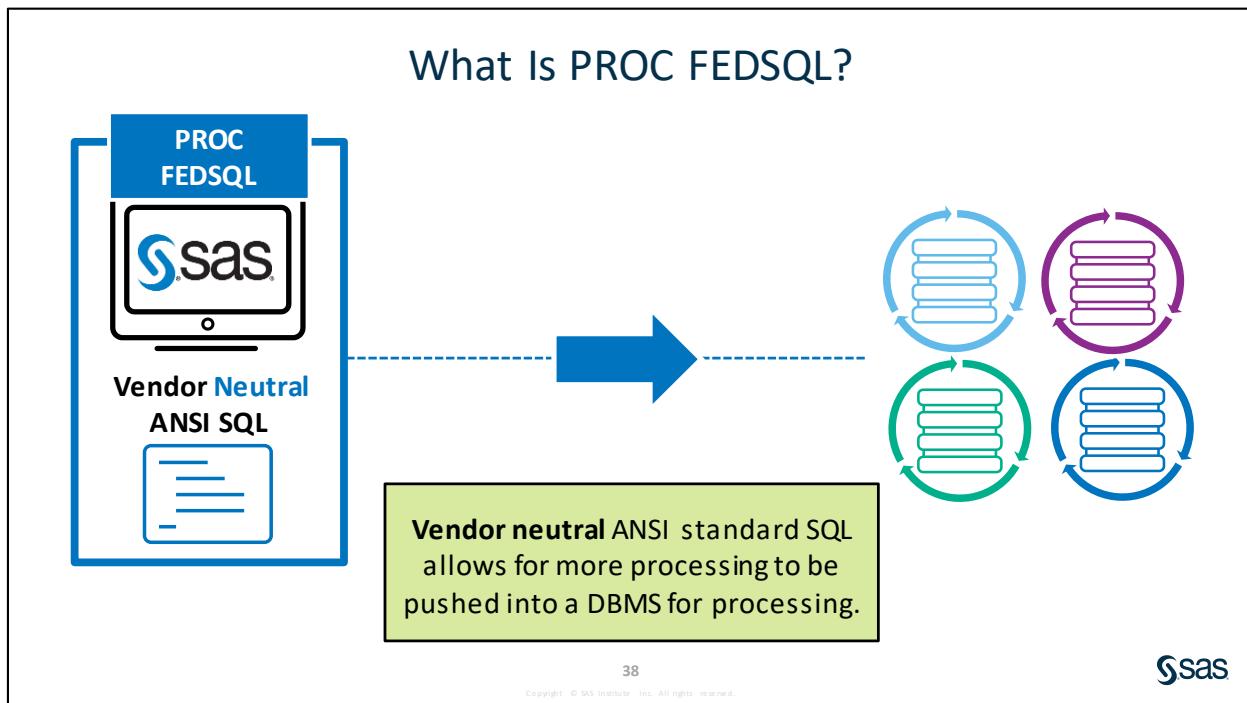
LIBNAME Statement



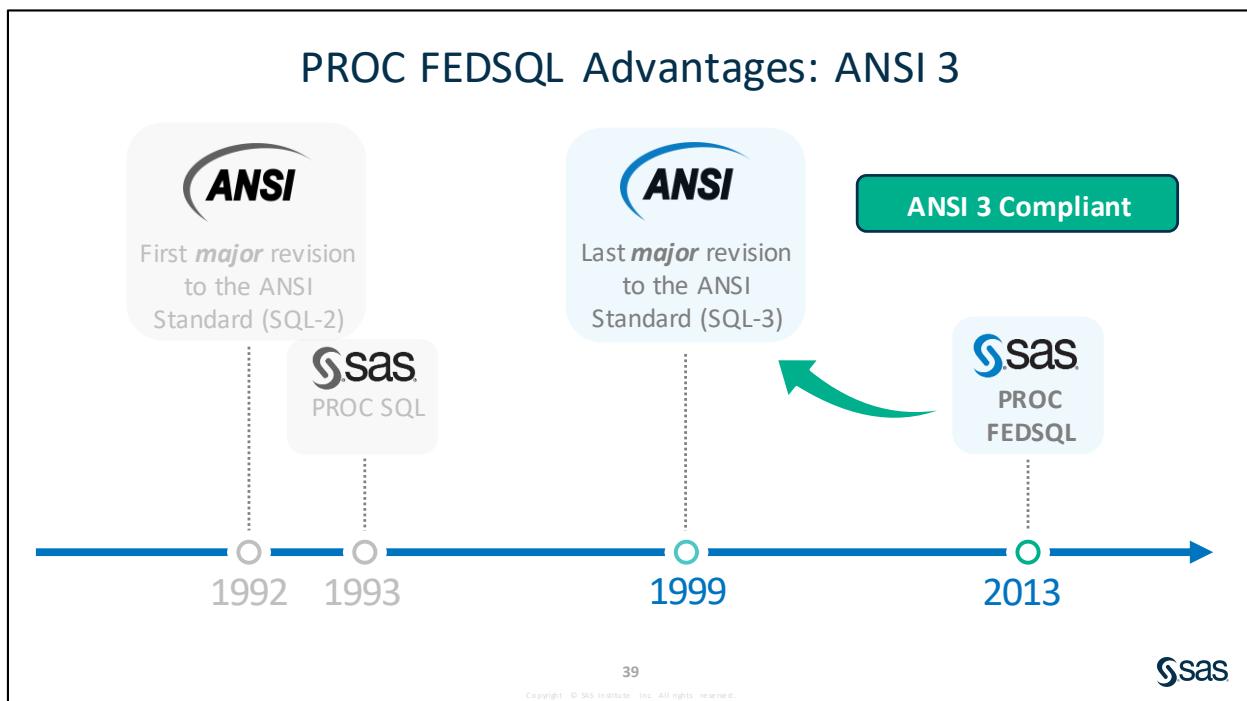
LIBNAME *libref* CLEAR;

Clear the libref

7.4 FEDSQL Procedure

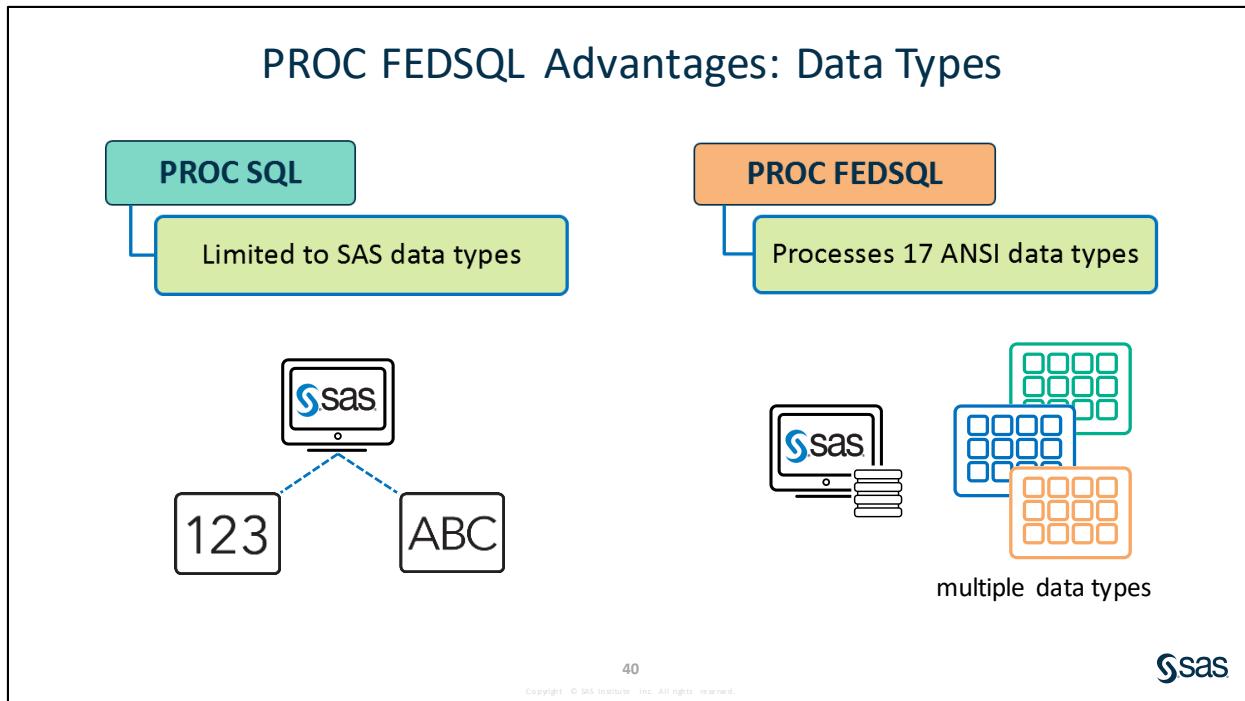


PROC FEDSQL is not a replacement for PROC SQL. On the contrary, both are tools that are used for specific scenarios. They offer different strengths for different situations. But in the end, they are both an implementation of SQL and have some slight differences in syntax. But when you know one, you can easily transition to another.



The FEDSQL procedure is a SAS proprietary implementation of ANSI SQL:1999 (SQL-3) core standard. It provides ANSI 1999 core compliance features and proprietary extensions.

For more information about PROC FEDSQL and the ANSI Standard, see "FedSQL and the ANSI Standard" in the *SAS® 9.4 FedSQL Language Reference, Fifth Edition* documentation. You can also find the direct link in the Course Links section on the ELP.



PROC FEDSQL supports many more data types than previous SAS SQL implementations. Traditional DBMS access through SAS/ACCESS LIBNAME engines translates target DBMS data types to and from two legacy SAS data types: SAS numeric and SAS character. PROC FEDSQL processes ANSI data types at native precision using a threaded driver instead of the typical data access and translation by the LIBNAME engine.

Large Data Types

ORACLE®

```
proc sql;
select *
from orcbigint;
quit;
```

BIGINT
8273652738098450432
7392946582819160064
1029658292748290048

```
proc fedsql;
select *
from orcbigint;
quit;
```

BIGINT
8273652738098453789
7392946582819163536
1029658292748292037

PROC SQL loses precision after 15 significant digits.

PROC FEDSQL allows for larger data types.

Copyright © SAS Institute Inc. All rights reserved.

41

Sas

PROC SQL

ANSI Type	Resulting SAS Type	Default Length
CHAR(n)	Character	8
VARCHAR(n)	Character	8
INTEGER	Numeric	8
SMALLINT	Numeric	8
DECIMAL	Numeric	8
NUMERIC	Numeric	8
FLOAT	Numeric	8
REAL	Numeric	8
DOUBLE PRECISION	Numeric	8
DATE	Numeric	8

Traditional DBMS access translates target DBMS data types to SAS data types.

Copyright © SAS Institute Inc. All rights reserved.

42

Sas

PROC SQL can work only with the data types defined in SAS: numeric and character. When working with DBMSs, SAS translates ANSI data types to SAS data types.

For example, when working with a DBMS that has data defined as integer or float, that is not a problem for PROC SQL it converts the value to a SAS numeric. The precision values of SAS numeric values are accurate to approximately 15 digits.

For more information about data types for SAS data, see "Data Types for SAS Data Sets" in the *SAS® 9.4 FedSQL Language Reference, Fifth Edition* documentation. You can also find the direct link in the Course Links section on the ELP.

For more information about the numerical accuracy in SAS software, see "Numerical Accuracy in SAS Software" in the *SAS® 9.4 FedSQL Language Reference, Fifth Edition* documentation. You can also find the direct link in the Course Links section on the ELP.

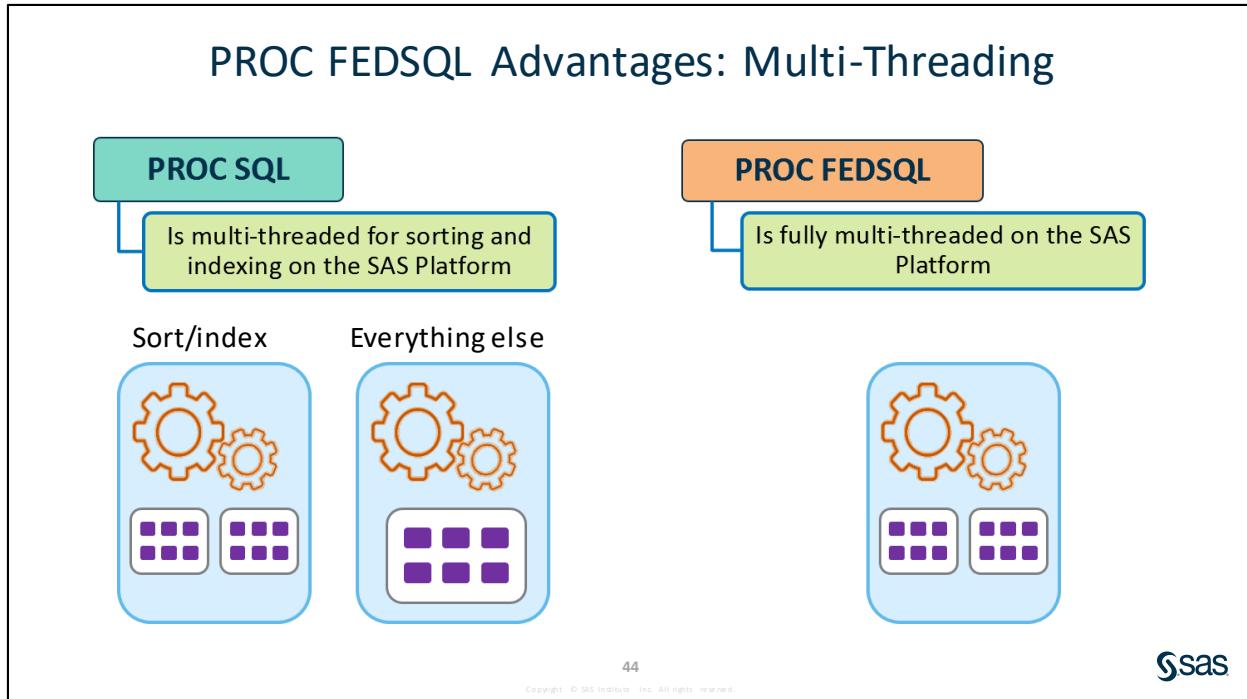
PROC FEDSQL Data Types

DATA TYPES		
BIGINT	FLOAT(p)	TIME(p)
BINARY(n)	INTEGER	TIMESTAMP(p)
CHAR(n)	NCHAR(n)	TINYINT
DATE	NVARCHAR(n)	VARBINARY(n)
DOUBLE	REAL	VARCHAR(n)
DECIMAL NUMERIC(p,s)	SMALLINT	

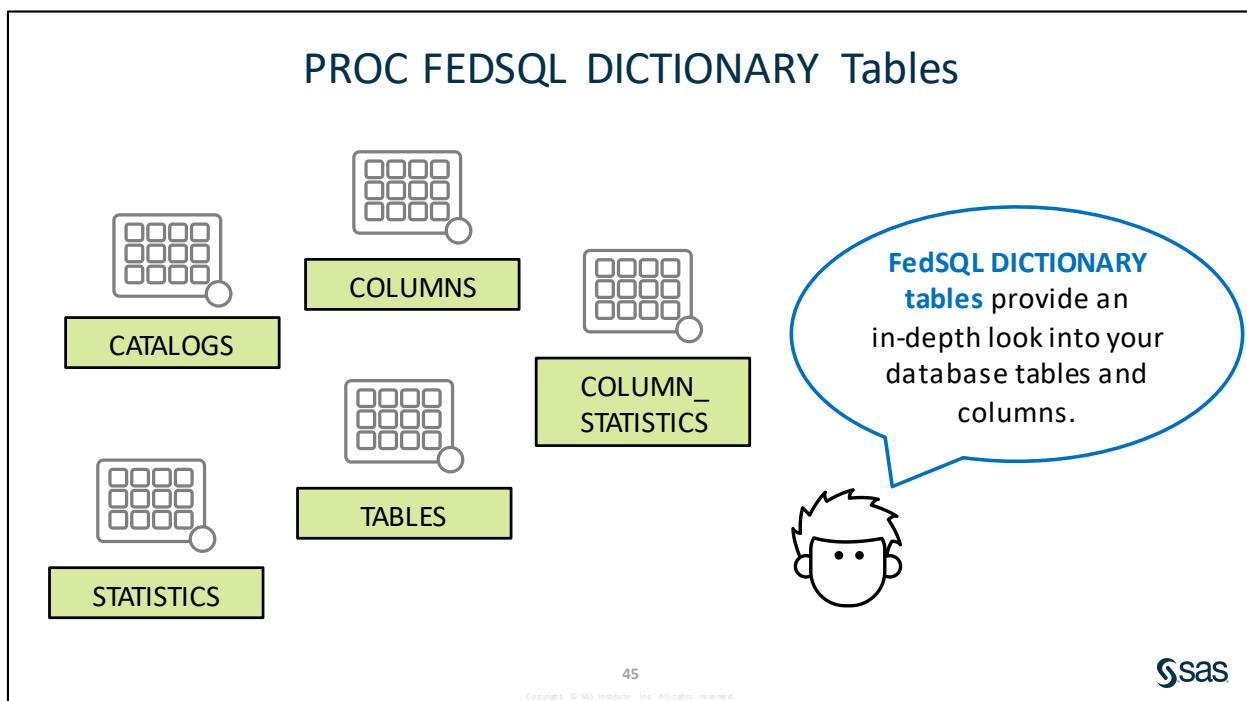
PROC FEDSQL allows for 17 ANSI data types.

For more information about PROC FEDSQL data types, see "Data Types" in the *SAS® 9.4 FedSQL Language Reference, Fifth Edition* documentation. You can also find the direct link in the Course Links section on the ELP.

For more information about PROC FEDSQL data types for specific DBMSs, see "Data Type Reference" in the *SAS® 9.4 FedSQL Language Reference, Fifth Edition* documentation. You can also find the direct link in the Course Links section on the ELP.



PROC FEDSQL provides a scalable, threaded, high-performance way to access, manage, and share relational data in multiple data sources. When possible, PROC FEDSQL queries are optimized with multi-threaded algorithms in order to resolve large-scale operations.



A PROC FEDSQL DICTIONARY table is a Read-only table that contains information about columns, tables, and catalogs, as well as statistics about tables and their associated indexes.

PROC FEDSQL DICTIONARY tables can give you a more in-depth look into your database tables and columns.

For more information about PROC FEDSQL DICTIONARY tables, see "FedSQL DICTIONARY Tables" in the SAS® 9.4 FedSQL Language Reference, Fifth Edition documentation. You can also find the direct link in the Course Links section on the ELP.

PROC FEDSQL Syntax

```
PROC FEDSQL <options>;
```

```
SELECT col-name, col-name  
      FROM input-table  
      <WHERE clause>  
      <GROUP BY clause>  
      <HAVING clause>  
      <ORDER BY clause>;
```

```
QUIT;
```

The foundation of the **PROC FedSQL** syntax is similar to **PROC SQL**.



When using PROC FEDSQL, ANSI standard syntax is required.

PROC FEDSQL LIBNAME Statement

```
LIBNAME libref engine <SAS/ACCESS-engine-options>;
```

```
PROC FEDSQL <options>;  
      SELECT col-name, col-name  
      FROM libref.table;  
      QUIT;
```

PROC FEDSQL needs the correct DBMS information through the SAS **LIBNAME statement**.



PROC FEDSQL LIBNAME Statement

```

libname mkt oracle user=sas_user password=sastest
          path=localhost;

proc fedsql;
  select State,
         count(*) as TotalCustomer
    from mkt.customer ←
  where CreditScore > 700
  group by State
  order by TotalCustomer desc;
quit;

libname mkt clear;

```

Reference the table.

48


Copyright © SAS Institute Inc. All rights reserved.

7.03 Activity

Open **s107a03.sas** from the **activities** folder and perform the following tasks to perform a PROC FEDSQL query:

1. Examine and run the query. Did it produce an error?
2. In the WHERE clause, replace the **double quotation marks** around NC with **single quotation marks**. Run the query. Did it run successfully?

49


Copyright © SAS Institute Inc. All rights reserved.

Examining the SQL That PROC FEDSQL Submits

```
PROC FEDSQL IPTRACE;
```

```
libname mkt oracle user= path=
proc fedsql iptrace;
select State,
       count(*) as TotalCustomer
  from mkt.customer
 where CreditScore > 700
 group by State
 order by TotalCustomer desc;
quit;

libname mkt clear;
```

The IPTRACE option prints a text description of the PROC FEDSQL query plan.

```
IPTRACE: Query:
select State, count(*) as TotalCustomer from
IPTRACE: FULL pushdown to ORACLE SUCCESS!
IPTRACE: Retextualized child query:
select "SAS_USER"."CUSTOMER"."STATE", COUNT (
("SAS_USER"."CUSTOMER"."CREDITSCORE">700) )
IPTRACE: END
```

For debugging purposes, you can see that PROC FEDSQL submits to the DBMS by specifying the undocumented IPTRACE option. This option causes PROC FEDSQL to write the SQL sent to the RDBMS in the log as the "Retextualized child query." If the query is not accepted by the DBMS, IPTRACE writes the error message generated by the RDBMS in the log. PROC FEDSQL continues to reformulate the query and try again until the DBMS executes the query and returns a result set. IPTRACE then reports how much of the original query was pushed down into the DBMS.

Another method that prints the query plan is the _METHOD option. For more information about _METHOD, see "_METHOD FedSQL Option" in the *Base SAS® 9.4 Procedures Guide, Seventh Edition* documentation. You can also find the direct link in the Course Links section on the ELP.

Working with SAS Viya

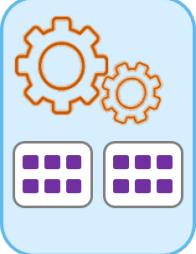
PROC SQL

Does not execute in CAS

PROC FEDSQL

Executes in CAS


SAS® Viya®





Copyright © SAS Institute Inc. All rights reserved.

SAS Viya on the SAS Platform

SAS Platform

SAS Viya

- an open, cloud-enabled, analytic run-time environment

SAS Cloud Analytic Services (CAS)



in-memory engine



fast processing



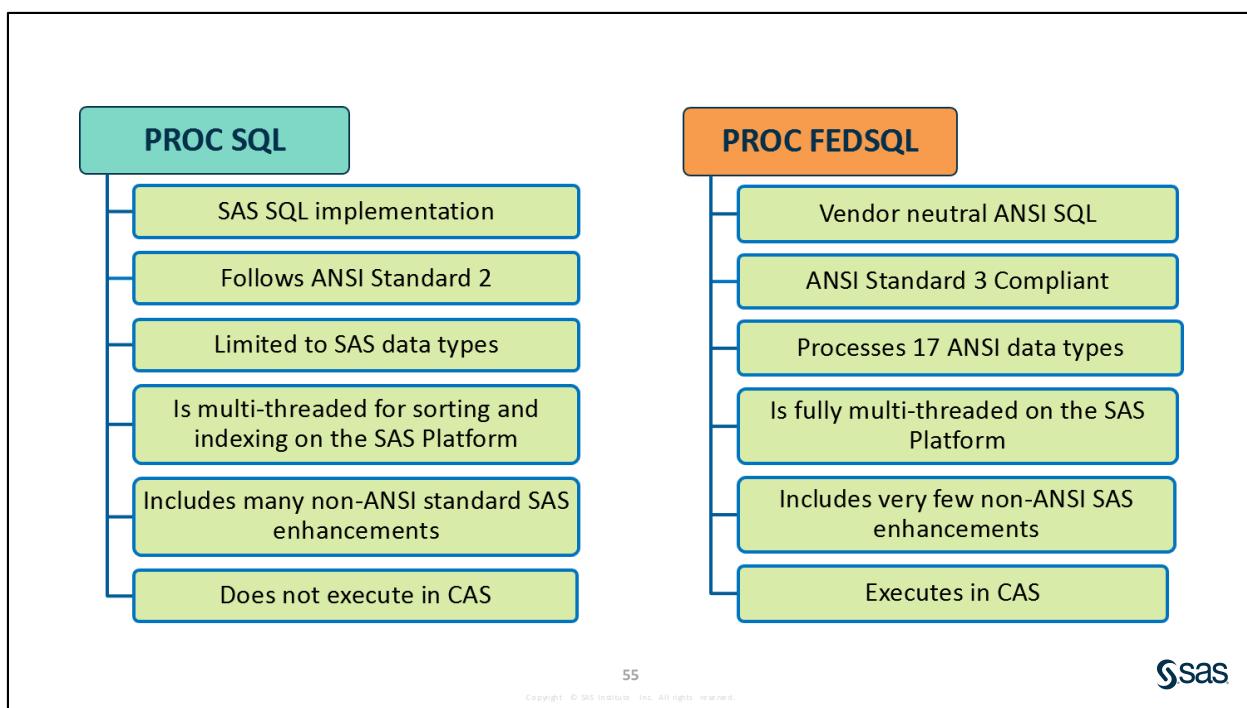
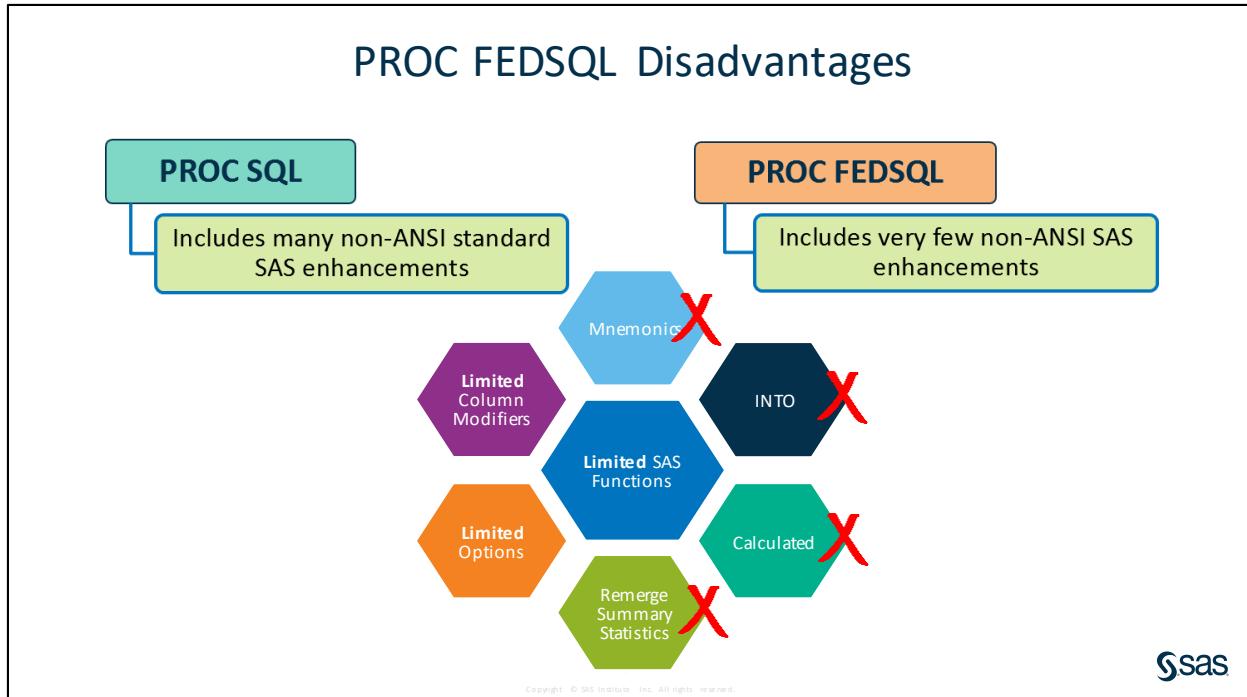
data of any size

53

Copyright © SAS Institute Inc. All rights reserved.

SAS Viya is the latest enhancement of the SAS Platform. It is an open, cloud-enabled, analytic run-time environment with a number of supporting services.

One of those supporting services is SAS Cloud Analytic Services, or CAS. CAS provides a powerful in-memory engine that delivers blazing speed to accurately process your big data. It uses scalable, high-performance, multi-threaded algorithms to rapidly perform analytical processing on in-memory data of any size.



For a summary document about PROC SQL versus PROC FEDSQL, see "PROC SQL vs. PROC FEDSQL Summary" on the ELP.

Syntax Summary

```
PROC FEDSQL <options>;
SELECT col-name, col-name
  FROM input-table
  <WHERE clause>
  <GROUP BY clause>
  <HAVING clause>
  <ORDER BY clause>;
QUIT;
```

PROC FEDSQL



PROC FEDSQL IPTRACE;

PROC FEDSQL Options

56



Beyond SQL Programming

What if you want to ...

... learn more about working with DBMSs by viewing the SAS documentation or taking a SAS course

- View the [SAS/ACCESS 9.4 for Relational Databases: Reference, Ninth Edition](#).
- View the [SAS 9.4 FedSQL Language Reference, Fifth Edition](#).
- Take the [Processing Database and PC File Data with SAS/ACCESS Software](#) course.
- Take the [Introduction to SAS/ACCESS Interface to Teradata](#) course.
- Take the [Introduction to SAS and Hadoop](#) course.

... see the ELP for additional information about working with DBMS

- View the **Working with Database** section in the ELP for additional SAS papers and blogs.

57



Put It All Together!

Extended Learning - SAS® SQL 1: Essentials

Dashboard / All courses / Extended Learning - SAS® SQL 1: Essentials



Practice all your new SAS skills by completing a comprehensive case study. Visit the Extended Learning page to access the case study materials.



58

sas

Copyright © SAS Institute Inc. All rights reserved.

Join the Discussion

<https://communities.sas.com/sas-training>

Visit the SAS Training Community to ask questions of our experts and exchange ideas with other SAS users.



Communities SAS Training

Discuss SAS courses and test your SAS skills

[Join Now >](#)

59

sas

Copyright © SAS Institute Inc. All rights reserved.

7.5 Solutions

Solutions to Activities and Questions

7.01 Activity – Correct Answer

1. Did it produce an error? Yes
2. Did it run successfully? Yes. IS MISSING is a SAS enhancement. IS NULL is ANSI standard.

```
connect to pcfiles(path="&path\database\SQL_DB.accdb"
                     dbpassword=sastest);
select UserID, Income format=dollar16., State
  from connection to pcfiles
    (select top 10, UserID, Income, State
     from customer
      where BankID is null
      order by Income desc);
disconnect from pcfiles;
```

A native ACCESS SQL query requires IS NULL.

21



Copyright © SAS Institute Inc. All rights reserved.

7.03 Activity – Correct Answer

1. Did it produce an error? Yes
2. Did it run successfully? Yes. Double quotation marks around a character string is a SAS enhancement, and it produces an error when using PROC FEDSQL. Single quotation marks around a character string is ANSI standard and required in FEDSQL.

```
proc fedsql;
select UserID, Income, State
  from sq.customer
    where State='NC'
    order by Income desc
      limit 10;
quit;
```

Single quotation marks are ANSI standard.

The LIMIT statement limits the number of records returned in FedSQL.

50



Copyright © SAS Institute Inc. All rights reserved.

To use a macro variable in a character string in PROC FEDSQL, you must use %TSLIT(&*macro-variable*).

```
%LET StateVar=NC;  
  
PROC FEDSQL;  
SELECT UserID, Income, State  
  FROM sq.customer  
 WHERE State=%t$lit(&StateVar)  
 ORDER BY Income DESC  
 LIMIT 10;  
QUIT;
```