Confessions of a SAS® PROC SQL Instructor

Course Notes

Confessions of a SAS® PROC SQL Instructor Course Notes was developed by Charu Shankar.

Confessions of a SAS® PROC SQL Instructor Course Notes

Book code E71361, course code UGCPSQL, prepared date 10May2019. UGCPSQL_001

Table of Contents

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 other SAS books that relate to the topics covered in this course notes, USA customers can contact the SAS Publishing Department at 1-800-727-3228 or send e-mail to sasbook@sas.com. Customers outside the USA, please contact your local SAS office.

Also, see the SAS Bookstore on the web at http://support.sas.com/publishing/ for a complete list of books and a convenient order form.

Chapter 1 PROC SQL Syntax Order: So Few Workers Go Home on Time

1.1 Overview of the SQL Procedure

SQL Procedure

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

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

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

2 s102d01

SQL Procedure

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

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

3

SELECT Statement

A *SELECT statement* is used to query one or more tables. The results of the SELECT statement are written to the default output destination.

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

4 s102d01

SELECT Statement

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

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

clauses

Note: Although it can contain multiple clauses, each SELECT statement begins with the SELECT keyword and ends with a semicolon.

5 s102d01

Viewing the Output

Partial PROC SQL Output

The SAS System

Employee ID	Employee Gender	Employee Annual Salary
120260	F	$207,885
120719	F	$87,420
120661	F	$85,495
121144	F	$83,505
120798	F	$80,755

6

SELECT Statement: Required Clauses

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

Here are two things that SQL always needs:

1. What do you want?
   The SELECT clause specifies the columns and column order.
2. Where do you want it from?
   The FROM clause specifies the data sources. You can query from 1 to 256 tables.

7

SELECT Statement: Syntax Order Mnemonic

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

- The WHERE clause specifies data that meets certain conditions.
- The GROUP BY clause groups data for processing.
- The HAVING clause specifies groups that meet certain conditions.
- The ORDER BY clause specifies an order for the data.

8

SELECT Statement Syntax

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

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

9

Discussion

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

Is this code correct?

10

Syntax Check with the NOEXEC Option

To explicitly check for syntax errors without submitting the code for execution, include the NOEXEC option in the PROC SQL statement. This option applies to all statements in a PROC SQL step.

PROC SQL<NOEXEC>;

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

s102d03

11

Viewing the Log

Partial SAS Log

```
proc sql noexec;
select Employee_ID, Employee_Gender, Salary
   from PHSUG.employee_information
   where Employee_Gender='F'
   order by Salary desc;
NOTE: Statement not executed due to NOEXEC option.
quit;
```

12

Chapter 2 Know Thy Data: DICTIONARY Tables

2.1 DICTIONARY Tables and Views

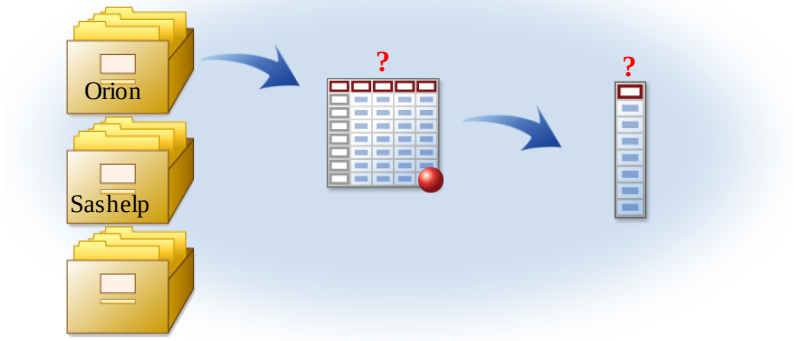
Objectives

- Use DICTIONARY tables and views to obtain information about SAS files.

2

Business Scenario

You have inherited many different data tables and want to become familiar with their content.



3

DICTIONARY Tables: Overview

DICTIONARY tables are Read-Only metadata views that contain session metadata, such as information about SAS libraries, data sets, and external files in use or available in the current SAS session.

DICTIONARY tables are

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

You can query DICTIONARY tables with PROC SQL.

4

Note: *Metadata* is data that provides information about other data.

Querying Metadata about SAS Libraries

There can be more than 30 DICTIONARY tables. We will focus on using data from three of the tables.

DICTIONARY.TABLES
- detailed information about tables

DICTIONARY.COLUMNS
- detailed information about all columns in all tables

DICTIONARY.MEMBERS
- general information about SAS library members

5

Note: SAS librefs are limited to eight characters. The libref **dictionary** is an automatically assigned reserved libref that is accessible only from within PROC SQL.

Exploring DICTIONARY Tables

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

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

Partial Log

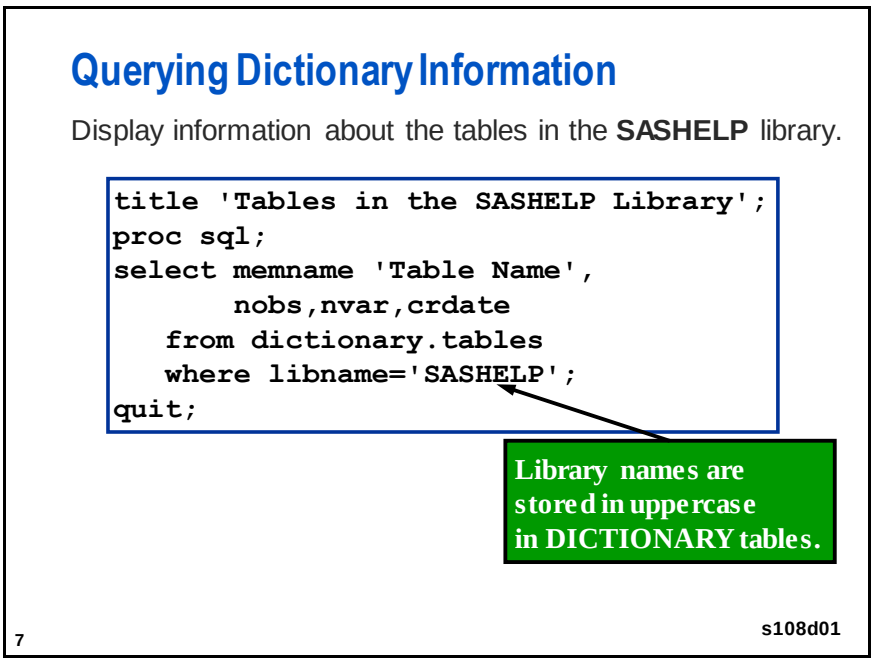
```
NOTE: SQL table DICTIONARY.TABLES was created like:

create table DICTIONARY.TABLES
  (
   libname char(8) label='Library Name',
   memname char(32) label='Member Name',
   ...
   crdate num format=DATETIME informat=DATETIME label='Date Created',
   modate num format=DATETIME informat=DATETIME label='Date Modified',
   nobs num label='Number of Physical Observations',
   obslen num label='Observation Length',
   nvar num label='Number of Variables', ...);
```

6 s108d01

The DESCRIBE TABLE statement is a good tool for exploring DICTIONARY tables. The complete log notes from the DESCRIBE statement are shown below:

```
create table DICTIONARY.TABLES
  (
   libname char(8) label='Library Name',
   memname char(32) label='Member Name',
   memtype char(8) label='Member Type',
   dbms_memtype char(32) label='DBMS Member Type',
   memlabel char(256) label='Dataset Label',
   typemem char(8) label='Dataset Type',
   crdate num format=DATETIME informat=DATETIME label='Date Created',
   modate num format=DATETIME informat=DATETIME label='Date Modified',
   nobs num label='Number of Physical Observations',
   obslen num label='Observation Length',
   nvar num label='Number of Variables',
   protect char(3) label='Type of Password Protection',
   compress char(8) label='Compression Routine',
   encrypt char(8) label='Encryption',
   npage num label='Number of Pages',
   filesize num label='Size of File',
   pcompress num label='Percent Compression',
   reuse char(3) label='Reuse Space',
   bufsize num label='Bufsize',
   delobs num label='Number of Deleted Observations',
   nlobs num label='Number of Logical Observations',
   maxvar num label='Longest variable name',
   maxlabel num label='Longest label',
   maxgen num label='Maximum number of generations',
   gen num label='Generation number',
   attr char(3) label='Dataset Attributes',
   indxtype char(9) label='Type of Indexes',
   datarep char(32) label='Data Representation',
   sortname char(8) label='Name of Collating Sequence',
   sorttype char(4) label='Sorting Type',
```

```
   sortchar char(8) label='Charset Sorted By',
   reqvector char(24) format=$HEX48 informat=$HEX48 label='Requirements Vector',
   datarepname char(170) label='Data Representation Name',
   encoding char(256) label='Data Encoding',
   audit char(3) label='Audit Trail Active?',
   audit_before char(3) label='Audit Before Image?',
   audit_admin char(3) label='Audit Admin Image?',
   audit_error char(3) label='Audit Error Image?',
   audit_data char(3) label='Audit Data Image?'
);
```



Note: SAS library and table names are stored in uppercase in the DICTIONARY tables. Using SAS functions, such as UPCASE() or LOWCASE(), when querying DICTIONARY tables dramatically degrades query performance. For example, using **upcase(libname)='SASHELP'** causes ***all*** librefs assigned to the SAS session to be opened to return this information. This prevents the PROC SQL Query Optimizer from seeing many conditions that could be optimized. This is especially apparent when librefs are assigned to a DBMS using a SAS/ACCESS engine.

When you query DICTIONARY tables, you supply values to the WHERE clause in the appropriate case, and match the known case for library and table names (uppercase) and for column names (mixed case). (Know your data!)

The PRESERVE_TAB_NAMES=YES and PRESERVE_COL_NAMES=YES options change how some table and column names are seen by SAS in the DICTIONARY tables. These options might require further investigation to maximize the efficiency of your queries.

Viewing the Output

Partial PROC SQL Output

Tables in the SASHELP Library

Table Name	Number of Physical Observations	Number of Variables	Date Created
AACOMP	2020	4	25JUN15:01:05:47
AARFM	61	4	25JUN15:01:07:08
ADSMSG	426	6	25JUN15:01:09:46
AFMSG	1090	6	25JUN15:01:06:18
AIR	144	2	25JUN15:01:12:52
APPLIANC	156	25	25JUN15:01:12:54
ASSCMGR	402	19	25JUN15:01:19:20
AUTHLIB	4	7	25JUN15:01:24:40

8

Note: This report varies based on the tables created during the SAS session.

Querying Dictionary Information

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

```
title 'Columns in the sashelp.cars Table';
proc sql;
select Name,Type,Length
   from dictionary.columns
   where libname='SASHELP'
         and memname='CARS';
quit;
```

Table names (*memnames*) are also stored in uppercase in DICTIONARY tables.

9 s108d01

Viewing the Output

PROC SQL Output

Columns in the sashelp.cars Table

Column Name	Column Type	Column Length
Make	char	13
Model	char	40
Type	char	8
Origin	char	6
DriveTrain	char	5
MSRP	num	8
Invoice	num	8
EngineSize	num	8
Cylinders	num	8
Horsepower	num	8
MPG_City	num	8
MPG_Highway	num	8
Weight	num	8
Wheelbase	num	8
Length	num	8

Column names are stored in mixed case.

10

Using Dictionary Information

Which tables contain an **ID** column?

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

Because different tables might use different cases for same-named columns, you can use the UPCASE function for comparisons. However, this significantly degrades the performance of the query.

11 s108d01

Viewing the Output

Tables Containing an ID Column

Table Names	Column Name
ADSMSG	MSGID
AFMSG	MSGID
ASSCMGR	ID
BURROWS	ID
CLNMSG	MSGID
COLUMN	TABLEID
COLUMN	ID
DEMOGRAPHICS	ID
DFTDICT	ID
DYNATTR	SOURCEID
DYNATTR	ID
EISMKCN	ID

All **ID** column names are stored in uniform uppercase, so the UPCASE function is not needed the next time that a query such as this is executed.

12

Note: The tables identified in this report vary depending on the files created in your SAS session.

Finding Common Column Names Dynamically

All of the previous techniques to explore DICTIONARY tables work when you know the names of columns.

What happens if you do not know your data, and you want SAS to retrieve all same-named columns in a library.

Use the following code

```
title 'Common columns in SASHELP';
proc sql;
select name, type, length, memname
   from dictionary.columns
   where libname='SASHELP'
   group by name
   having count(name) > 1;
```

s108d01

13

Viewing the Output

Common columns in SASHELP

Column Name	Member Name	Column Type	Column Length
ACTUAL	PRDSAL2	num	8
ACTUAL	PRDSAL3	num	8
ACTUAL	PRDSALE	num	8
ALIAS_CITY	ZIPCODE	char	300
ALIAS_CITY	ZIPMIL	char	300
ALIAS_CITYN	ZIPCODE	char	300
ALIAS_CITYN	ZIPMIL	char	300
AMOUNT	BUY	num	8
AMOUNT	NVST1	num	8
AMOUNT	NVST2	num	8
AMOUNT	NVST3	num	8
AMOUNT	NVST4	num	8
AMOUNT	NVST5	num	8
AMOUNT	RENT	num	8
AMOUNT	ROCKPIT	num	8

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

14

Using DICTIONARY Tables in Other SAS Code

SAS provides views based on the DICTIONARY tables in the **SASHELP** library.

Most of the **SASHELP** library DICTIONARY view names are similar to DICTIONARY table names, but they are shortened to eight characters or less. They begin with the letter **v** and do not end in **s**. For example:

dictionary.tables = sashelp.vtable

The following code executes successfully:

```
title 'Tables in the SASHELP Library';
proc print data=sashelp.vtable NOOBS ;
   var memname nobs nvar;
   where libname='SASHELP';
run;
```

s108d01

15

An Efficiency Question: PROC SQL or PRINT?

```
options fullstimer;
proc sql;
   select libname, memname, name, type, length
   from dictionary.columns
   where upcase(name) contains 'ID'
   and libname='SASHELP' and type='num';
quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time               0.73 seconds
      user cpu time           0.42 seconds
      system cpu time         0.29 seconds
      memory                  5584.18k
      OS Memory               24672.00k
      Timestamp               05/22/2018 01:52:52 PM
      Step Count              4  Switch Count  36
```

16

s108d01

An Efficiency Question: PROC SQL or PRINT?

What do these statistics mean?

Statistic	Description
Real Time	The amount of real time (clock time) spent to process the SAS job. Real time is also referred to as *elapsed time*.
User CPU Time	The CPU time that is spent in the user program.
System CPU Time	CPU time is spent to perform operating system tasks (system overhead tasks) that support the execution of your SAS code.
Memory	The amount of memory required to run a step.
OS Memory	the largest amount of operating system memory that is available to SAS during the step.
Timestamp	The date and time that a step was executed.
Step Count	Count of DATA steps or procedures that run in a SAS program.
Switch Count	A count of task switches within a step—that is, within a DATA step or procedure—in a SAS program. A task switch occurs when a step requests service from another process. Another task switch occurs when the step resumes. The number reported is for the last step that runs.

17

An Efficiency Question: PROC SQL or PRINT?

Can I use PROC PRINT instead?

```
options fullstimer;
proc print data=sashelp.vcolumn;
   var libname memname name type length;
   where upcase(name) contains 'ID' and
libname='SASHELP' and type='num';
run;
NOTE: There were 34 observations read from the data set
SASHELP.VCOLUMN. WHERE UPCASE(name) contains 'ID' and
(libname='SASHELP') and (type='num');
NOTE: PROCEDURE PRINT used (Total process time):
      real time           2.19 seconds
      user cpu time       0.92 seconds
      system cpu time     1.18 seconds
      memory              6738.81k
      OS Memory           25440.00k
      Timestamp           05/22/2018 02:22:29 PM
      Step Count                10  Switch Count  44
```

18

An Efficiency Question: PROC SQL or PRINT? Why Is PROC SQL More Efficient?

While querying a DICTIONARY table, SAS launches a discovery process. Depending on the DICTIONARY table being queried, this discovery process can search libraries, open tables, and execute views.

The PROC SQL step runs much faster than other SAS procedures and the DATA step. This is because PROC SQL can optimize the query before the discovery process is launched. It has to do with the processing order. The PROC SQL step runs much faster because the WHERE clause is processed before the tables referenced by the SASHELP.VCOLUMN view are opened. Therefore, it is more efficient to use PROC SQL instead of the DATA step or SAS procedures to query DICTIONARY tables.

19

Try it out for yourself. Both programs above produce the same result, ***But*** the SAS PROC step might have you wringing your hands because it can seem to take forever to execute. By now you know why: it has to search libraries, open tables, and so on. On the other hand, PROC SQL optimized the query due to the WHERE clause and returns you results quickly.

Chapter 3 Two Ways to Stack Data Horizontally

3.1 SQL Joins: When Is a Cartesian Product Useful?

Objectives

- Identify different ways to combine data horizontally from multiple tables.
- Understand the Cartesian product.

3

Business Scenario

Management has requested multiple reports. You have to understand how to combine the data from the tables to complete these requests.

4

Exploring the Data

customers

ID	Name
101	Smith
104	Jones
102	Blank

transactions

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

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

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

5

Combining Data from Multiple Tables

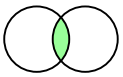
SQL uses *joins* to combine tables horizontally. Requesting a join involves matching data from one row in one table with a corresponding row in a second table. Matching is typically performed on one or more columns in the two tables.

6

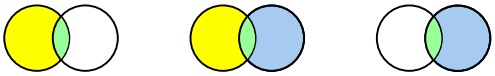
Types of Joins

PROC SQL supports two types of joins:

Inner joins return only matching rows.



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



Left Full Right

7

Cartesian Product

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

```
proc sql;
select *
   from customers, transactions;
quit;
```

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

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

s104d01

8

Building the Cartesian Product

customers

ID	Name
101	Smith
104	Jones
102	Blank

transactions

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

Result Set

ID	Name	ID	Action	Amount
101	Smith	102	Purchase	$100

9 ...

Building the Cartesian Product

customers

ID	Name
101	Smith
104	Jones
102	Blank

transactions

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

Result Set

ID	Name	ID	Action	Amount
101	Smith	102	Purchase	$100
101	Smith	103	Return	$52

10 ...

Building the Cartesian Product

customers

ID	Name
101	Smith
104	Jones
102	Blank

transactions

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

Result Set

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

11 ...

Building the Cartesian Product

customers

ID	Name
101	Smith
104	Jones
102	Blank

transactions

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

Result Set

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

12 ...

Building the Cartesian Product

customers

ID	Name
101	Smith
104	Jones
102	Blank

transactions

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

Result Set

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

13 ...

Building the Cartesian Product

customers

ID	Name
101	Smith
104	Jones
102	Blank

transactions

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

Result Set

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

14 ...

Building the Cartesian Product

customers

ID	Name
101	Smith
104	Jones
102	Blank

transactions

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

Result Set

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

15 ...

Building the Cartesian Product

customers

ID	Name
101	Smith
104	Jones
102	Blank

transactions

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

Result Set

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

16 ...

Building the Cartesian Product

customers

ID	Name
101	Smith
104	Jones
102	Blank

transactions

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

Result Set

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

17 ...

Building the Cartesian Product

customers

ID	Name
101	Smith
104	Jones
102	Blank

transactions

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

Result Set

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

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

18

Nonmatching Data in the Cartesian Product

customers

ID	Name
101	Smith
104	Jones
102	Blank

transactions

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

Result Set

Non-matching IDs

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

19

Size of the Cartesian Product

customers

ID	Name
101	Smith
104	Jones
102	Blank

transactions

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

Result Set

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

9 rows

20

Size of the Cartesian Product

The number of rows in a Cartesian product is the product of the number of rows in the contributing tables.

3 x 3 = 9

1,000 x 1,000 = 1,000,000

100,000 x 100,000 = 10,000,000,000

Partial SAS Log

```
NOTE: The execution of this query involves performing one or more
      Cartesian product joins that cannot be optimized.
```

21

3.01 Short Answer Poll

How many rows and columns are returned from this query?

```
select *
   from customer2, transaction2;
```

customer2

ID	Name
101	Jones
101	Jones
102	Kent
102	Kent
104	Avery

transaction2

ID	Action	Amount
102	Purchase	$376
102	Return	$119
103	Purchase	$57
105	Purchase	$98

22

When Is a Cartesian Product Useful?

Consider a table with as single summary row. If you want to combine with detail rows in another table to get percentages, a Cartesian product might be useful there.

23

Business Scenario

Calculate each male employee's salary as a percentage of all male employees' salaries.

PHSUG.employee_information

PROC SQL

Partial Results

Male Employee Salaries

Employee_ID	Salary	
120259	433,800	5.9%
120262	268,455	3.7%
120261	243,190	3.3%

24

Using Remerged Summary Statistics

Calculate each male employee's salary as a percentage of all male employees' salaries. Display **Employee_ID**, **Salary**, and percentage in decreasing order of percentage.

```
proc sql;
title "Male Employee Salaries";
select Employee_ID, Salary format=comma12.,
       Salary / sum(Salary)
       'PCT of Total' format=percent6.2
   from PHSUG.employee_information
   where Employee_Gender="M"
         and Employee_Term_Date is missing
   order by 3 desc;
quit;
title;
```

Select only the group of rows that you want to analyze.

25 ...

Using Remerged Summary Statistics

Calculate each male employee's salary as a percentage of all male employees' salaries. Display **Employee_ID**, **Salary**, and percentage in decreasing order of percentage.

```
proc sql;
title "Male Employee Salaries";
select Employee_ID, Salary format=comma12.,
       Salary / sum(Salary)
       'PCT of Total' format=percent6.2
   from PHSUG.employee_information
   where Employee_Gender="M"
         and Employee_Term_Date is missing
   order by 3 desc;
quit;
title;
```

individual salary value for each row

divided by a remerged summary value (sum of all salaries)

s103d09

26

Viewing the Output

Partial PROC SQL Output

Male Employee Salaries

Employee_ID	Employee Annual Salary	PCT of Total
120259	433,800	5.9%
120262	268,455	3.7%
120261	243,190	3.3%
121141	194,885	2.7%
120101	163,040	2.2%

Partial SAS Log

```
NOTE: The query requires remerging summary statistics back
      with the original data.
```

27

3.2 Subqueries: Best Practices, Dangers of Correlated

Objectives

- Define PROC SQL subqueries.
- Differentiate between correlated and noncorrelated subqueries.
- Subset data based on values returned from a subquery.

30

Business Scenario

HR and Payroll managers requested a report that displays **Job_Title** for job groups with an average salary greater than the average salary of the company as a whole.

Employee Job Title	MeanSalary
Account Manager	46090
Administration Manager	47415
Applications Developer I	42760
...	

31

Step 1

Calculate the company's average salary.

```
proc sql;
select avg(Salary) as CompanyMeanSalary
   from PHSUG.staff;
quit;
```

Company MeanSalary
38041.51

32 s105d01

Step 2

Determine the job titles whose average salary exceeds the company's average salary.

```
proc sql;
select Job_Title,
       avg(Salary) as MeanSalary
   from PHSUG.staff
   group by Job_Title
   having MeanSalary>38041.51;
quit;
```

Partial PROC SQL Output

Employee Job Title	MeanSalary
Account Manager	46090
Administration Manager	47415
Applications Developer I	42760

33 s105d01

Step 3

Write the program as a single step using a subquery.

A *subquery* is a query that resides within an outer query.

outer query

```
proc sql;
select Job_Title,
       avg(Salary) as MeanSalary
   from PHSUG.staff
   group by Job_Title
   having MeanSalary >
          (select avg(salary)
              from PHSUG.staff);
quit;
```

subquery

Note: The subquery must be resolved before the outer query can be resolved

34 s105do1

Subqueries

A subquery

- returns values to be used in the outer query's WHERE or HAVING clause

```
SELECT ...>
    FROM ...
    <WHERE ...>

    <GROUP BY ...>
    <HAVING ...>

    <ORDER BY ...>;
```

```
...(select Employee_ID
    from PHSUG.staff
    where ...  )...
```

35 ...

Subqueries: Noncorrelated

There are two types of subqueries:

- A *noncorrelated subquery* is a self-contained query. It executes independently of the outer query.

```
proc sql;
select Job_Title, avg(Salary) as MeanSalary
   from PHSUG.staff
   group by Job_Title
   having avg(Salary) >
      (select avg(Salary)
         from PHSUG.staff);
quit;
```

This query is a stand-alone query.

36

Subqueries: Correlated

- A *correlated subquery* requires a value or values to be passed to it by the outer (main) query before it can be successfully resolved.

```
proc sql;
select Employee_ID, avg(Salary) as MeanSalary
   from PHSUG.employee_addresses
   where 'AU'=
      (select Country
          from work.supervisors
          where employee_addresses.Employee_ID=
                supervisors.Employee_ID);
quit;
```

37

Correlated Subqueries

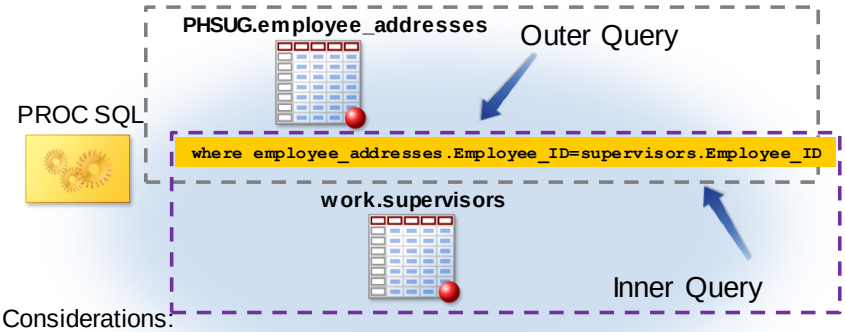
Correlated subqueries

- cannot be evaluated independently
- require values to be passed to the inner query from the outer query
- are evaluated for each row in the outer query.

38

Business Scenario

Use a correlated subquery to create a report listing the employee identifier and name for all managers in Australia.



Considerations:

- You have a temporary table, **Supervisors**, containing **Employee_ID** and **Country** for all managers.
- The table **wuss.Employee_Addresses** contains **Employee_Name** for all employees

39

Correlated Subqueries

In a correlated subquery, the outer query provides information so that the subquery resolves successfully.

```
proc sql;
  select Employee_ID,
         Employee_name
    from PHSUG.Employee_Addresses
    where 'AU'=
      (select Country
       from Work.Supervisors
      where Employee_Addresses.Employee_ID=
            Supervisors.Employee_ID);
quit;
```

This query is not stand-alone. It needs additional information from the main query.

You must qualify each column with a table name.

s1aad01

40

Correlated Subqueries

```
proc sql;
  select
Employee_ID,Employee_name
 from PHSUG.Employee_Addresses
  where 'AU'=
 (select Country
  from Work.Supervisors
where
Employee_Addresses.Employee_ID
=
Supervisors.Employee_ID) ;
quit;
```

Step 1: The outer query takes the first row in **wuss.employee_addresses** and finds **Employee_ID** and **Employee_Name**.

Partial Listing of Wuss.employee_addresses.

Employee ID	Employee_Name
120145	Aisbitt, Sandy
120798	Ardskin, Elizabeth
120656	Amos, Salley
120104	Billington, Kareen
121035	Blackley, James
121141	Bleu, Henri Le
120679	Cutucache, Chrisy
120103	Dawes, Wilson
120672	Guscott, Verne

Work.Supervisors

Employee ID	Country
120798	US
120800	US
120104	AU
120735	US
121141	US
...	...
120262	US
120679	US
120103	AU
120668	US
121143	US
120260	US
120672	AU

41

Correlated Subqueries

```
proc sql;
  select Employee_ID,
Employee_name from
PHSUG.Employee_Addresses
 where 'AU'=
(select Country from
Work.Supervisors
  where
Employee_Addresses.Employee_ID=
Supervisors.Employee_ID) ;
quit;
```

NO MATCH

Partial Listing of Wuss.employee_addresses.

Employee_ID	Employee_Name
120145	Aisbitt, Sandy
120798	Ardskin, Elizabeth
120656	Amos, Salley
120104	Billington, Kareen
121035	Blackley, James
121141	Bleu, Henri Le
120679	Cutucache, Chrisy
120103	Dawes, Wilson
120672	Guscott, Verne

Work.Supervisors

Employee_ID	Country
120798	US
120800	US
120104	AU
120735	US
121141	US
…	…
120262	US
120679	US
120103	AU
120668	US
121143	US
120260	US
120672	AU

Step 2: In the subquery, try to match an **employee_addresses.Employee_ID** value of **120145** with the value of **supervisors.Employee_ID** to find a qualifying row in **work.supervisors**.

42

Correlated Subqueries

```
proc sql;
select Employee_ID,Employee_Name
from PHSUG.Employee_Addresses
where 'AU'=
(select Country from
Work.Supervisors
Where
Employee_Addresses.Employee_ID=S
upervisors.Employee_ID) ;
quit;
```

MATCH

Partial Listing of Wuss.employee_addresses.

Employee_ID	Employee_Name
120145	Aisbitt, Sandy
120798	Ardskin, Elizabeth
120656	Amos, Salley
120104	Billington, Kareen
121035	Blackley, James
121141	Bleu, Henri Le
120679	Cutucache, Chrisy
120103	Dawes, Wilson
120672	Guscott, Verne

Work.Supervisors

Employee ID	Country
120798	US
120800	US
120104	AU
120735	US
121141	US
…	…
120262	US
120679	US
120103	AU
120668	US
121143	US
120260	US
120672	AU …

Steps 1 and 2 (Repeat): Read the next row from **wuss.employee_addresses** and pass the corresponding employee ID to the subquery to look for a matching employee ID in **work.supervisors**. There is a match.

43

Correlated Subqueries

```
proc sql;
select Employee_ID,Employee_Name
  from PHSUG.Employee_Addresses
  where 'AU'=
 (select Country
         from Work.Supervisors
Where Employee_Addresses.Employee_ID
=Supervisors.Employee_ID) ;
quit;
```

Subquery returns 'US'

FALSE

Partial Listing of Wuss.employee_addresses.

Employee_ ID	Employee_Name
120145	Aisbitt, Sandy
120798	Ardskin, Elizabeth
120656	Amos, Salley
120104	Billington, Kareen
121035	Blackley, James
121141	Bleu, Henri Le
120679	Cutucache, Chrisy
120103	Dawes, Wilson
120672	Guscott, Verne

Work.Supervisors

Employee ID	Country
120798	US
120800	US
120104	AU
120735	US
121141	US
…	…
120262	US
120679	US
120103	AU
120668	US
121143	US
120260	US
120672	AU

Step 3: The subquery passes the value of **Country** from the selected row in **work.supervisors** back to the outer query, where the = operator compares this value to **'AU'** for selection in the main query. In this case, the main query WHERE expression (where **'AU'='US'**) resolves to FALSE.

44

3.02 Quiz

Given the following query, subquery, and data in **Work.Supervisors**, what is the maximum number of rows that will be selected by the outer query?

```
proc sql;
select Employee_ID, Employee_Name
from PHSUG.Employee_Addresses
  where 'AU'=
  (select Country
  from Work.Supervisors
where Employee_Addresses.Employee_ID=
Supervisors.Employee_ID) ;
quit;
```

Work.Supervisors

Employee_ID	Country
120798	US
120800	US
120104	AU
120735	US
121141	US
…	…
120262	US
120679	US
120103	AU
120668	US
121143	US
120260	US
120672	AU

45

3.02 Quiz – Correct Answer

Given the following query, subquery, and data in **Work.Supervisors**, what is the maximum number of rows that will be selected by the outer query?

Only the three managers where Country='AU' would be selected.

```
proc sql;
   select Employee_ID,Employee_Name
from PHSUG.Employee_Addresses
where 'AU'=
(select Country
from Work.Supervisors
where Employee_Addresses.Employee_ID=
Supervisors.Employee_ID) ;
quit;
```

Work.Supervisors

Employee_ID	Country
120798	US
120800	US
120104	AU
120735	US
121141	US
…	…
120262	US
120679	US
120103	AU
120668	US
121143	US
120260	US
120672	AU

46

The Outer Query Controls the Result Set

The outer query determines which rows cause the inner query to resolve successfully.

```
proc sql;
select Employee_ID,Employee_Name
from PHSUG.Employee_Addresses
where 'AU'=
(select Country
 from Work.Supervisors
where
Employee_Addresses.Employee_ID=
Supervisors.Employee_ID) ;
quit;
```

Work.Supervisors

Employee ID	Country
120798	US
120800	US
120104	AU
120735	US
121141	US
…	…
120262	US
120679	US
120103	AU
120668	US
121143	US
120260	US
120672	AU

47

Correlated Subqueries

Build the first row of the report:

```
Employee_ID   Manager_Name
_____________________________________
     120104   Kareen Billington
```

48

Noncorrelated Subquery

```
proc sql;
select Job_Title,
       avg(Salary) as MeanSalary
   from PHSUG.staff
   group by Job_Title
   having avg(Salary) >
      (select avg(Salary)
          from PHSUG.staff);
quit;
```

Evaluate the subquery first.

49 s105d02

Noncorrelated Subquery

```
proc sql;
select Job_Title,
       avg(Salary) as MeanSalary
   from PHSUG.staff
   group by Job_Title
   having avg(Salary) >
      (38041.51);
quit;
```

Then pass the results to the outer query.

Partial PROC SQL Output

Employee Job Title	MeanSalary
Account Manager	46090
Administration Manager	47415
Applications Developer I	42760

50 s105d02

Chapter 4 Where ANSI SQL Falls Short and PROC SQL Steps In

4.1 Making a View Portable

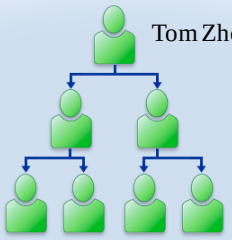
Objectives

- Create a PROC SQL view.
- Use PROC SQL views in SQL queries.
- Make a PROC SQL view portable.

2

Business Scenario

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



3

Business Data

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

PHSUG.employee_addresses

PHSUG.employee_payroll

PHSUG.employee_organization

4

Considerations

What is the best way to help Tom, given the following requirements:

- He should not be allowed access to personnel data for any employee that is not his direct report.
- He can write simple PROC SQL queries and use basic SAS procedures, but cannot write complex joins.

A PROC SQL view accessing data for Tom Zhou's direct reports can provide the information that Tom needs in a secure manner.

5

What Is a PROC SQL View?

A *PROC SQL view*

- is a stored query
- contains no actual data
- can be derived from one or more tables, PROC SQL views, DATA step views, or SAS/ACCESS views
- extracts underlying data each time it is used and accesses the most current data
- can be referenced in SAS programs in the same way as a data table
- cannot have the same name as a data table stored in the same SAS library.

6

Creating a PROC SQL View

To create a PROC SQL view, use the CREATE VIEW statement.

CREATE VIEW *view-name* **AS SELECT …;**

```
proc sql;
create view PHSUG.tom_zhou as
   select Employee_Name as Name format=$25.0,
          Job_Title as Title format=$15.0,
          Salary 'Annual Salary' format=comma10.2,
          int((today()-Employee_Hire_Date)/365.25)
             as YOS 'Years of Service'
      from employee_addresses as a,
           employee_payroll as p,
           employee_organization as o
      where a.Employee_ID=p.Employee_ID and
            o.Employee_ID=p.Employee_ID and
            Manager_ID=120102;
quit;
```

s107d07

7

View the Log

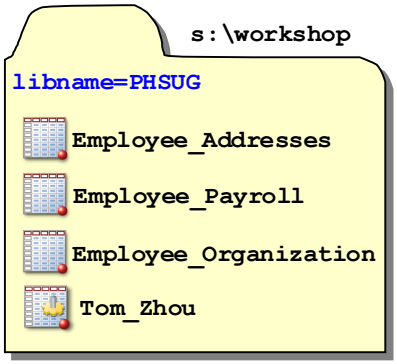
Partial SAS Log

```
5      proc sql;
46   create view PHSUG.tom_zhou as
47    select Employee_Name as Name format=$25.0,
48         Job_Title as Title format=$15.0,
49       Salary 'Annual Salary' format=comma10.2,
50              int((today()-Employee_Hire_Date)/365.25)
51                      as YOS 'Years of Service'
52               from employee_addresses as a,
53                    employee_payroll as p,
54                    employee_organization as o
55               where a.Employee_ID=p.Employee_ID and
56                     o.Employee_ID=p.Employee_ID and
57                     Manager_ID=120102;
NOTE: SQL view PHSUG.TOM_ZHOU has been defined.
```

8

Location of a PROC SQL View

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



9

Location of the Source Tables: ANSI

In PROC SQL, the default libref for the table (or tables) in the FROM clause is the libref of the library that contains the view. When the view and data source are in the same location, you specify a one-level name for the table (or tables) in the FROM clause.

```
create view PHSUG.tom_zhou as
   ...
   from employee_addresses as a,
        employee_payroll as p,
        employee_organization as o
```

10

Using a View

Tom can use the view to produce simple reports.

```
title "Tom Zhou's Direct Reports";
title2 "By Title and Years of Service";
select *
   from PHSUG.tom_zhou
   order by Title desc, YOS desc;
```

Partial PROC SQL Output (executed 27Aug2018)

Tom Zhou's Direct Reports

By Title and Years of Service

Name	Title	Annual Salary	Years of Service
Nowd, Fadi	Sales Rep. IV	30,660.00	40
Hofmeister, Fong	Sales Rep. IV	32,040.00	35
Phoumirath, Lynelle	Sales Rep. IV	30,765.00	28
Platts, Alexei	Sales Rep. IV	32,490.00	16
Kletschkus, Monica	Sales Rep. IV	30,890.00	7
Hayawardhana, Caterina	Sales Rep. III	30,490.00	40
Comber, Edwin	Sales Rep. III	28,345.00	40
Kaiser, Fancine	Sales Rep. III	28,525.00	35

11 s107d08

Business Scenario

You created a PROC SQL view to provide Tom Zhou access to personnel data for his direct reports.

Tom copied his view to a folder on his hard drive. Now Tom reports that the view does not work anymore, and he asked for your help to resolve the problem.

12

Exploring the Problem

Tom submitted the following:

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

13 s107d09

Viewing the Log

Partial SAS Log

```
   libname PHSUG 'c:\workshop';
NOTE: Libref WUSS was successfully assigned as follows:
      Engine:        V9
      Physical Name: c:\workshop
   proc sql;
   title "Tom Zhou's Direct Reports";
   title2 "By Title and Years of Service";
   select *
      from PHSUG.tom_zhou
      order by Title desc, YOS desc;
ERROR: File PHSUG.EMPLOYEE_ADDRESSES.DATA does not exist.
ERROR: File PHSUG.EMPLOYEE_PAYROLL.DATA does not exist.
ERROR: File PHSUG.EMPLOYEE_ORGANIZATION.DATA does not
exist.
   quit;
   title;
NOTE: The SAS System stopped processing this step because
of errors.
```

14

Considerations

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

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

15

Making a View Portable

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

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

two-level data set names

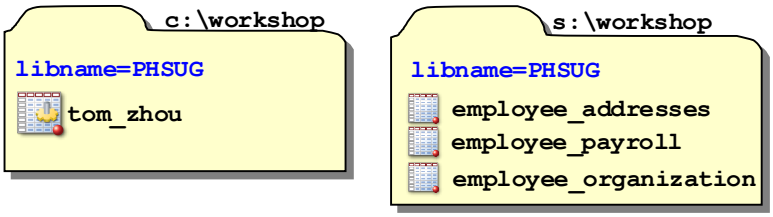
A USING clause names the location of the tables.

16 s107d10

Two-Level Table Names in Permanent Views

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

- The USING clause libref is local to the view, and it will not conflict with an identically named libref in the SAS session.
- When the query finishes, the libref is disassociated.



17

Views: Advantages

You can use views to do the following:

- 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.
- pull together data from multiple database tables and multiple libraries or databases.
- simplify complex queries.
- prevent other users from inadvertently altering the query code.

18

Views: Disadvantages

- Because views access the most current data in changing tables, the results might be different each time you access the view.
- Views can require significant resources each time they execute. With a view, you save disk storage space at the cost of extra CPU and memory usage.
- When accessing the same data several times in a program, use a table instead of a view. This ensures consistent results from one step to the next and can significantly reduce the resources that are required.

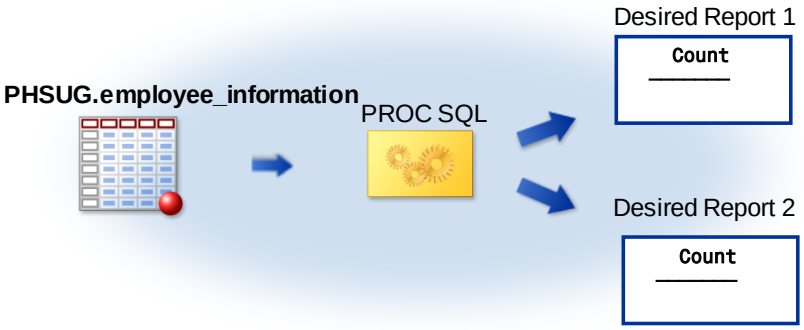
19

Chapter 5 Summarizing Data Using the Boolean Gate

5.1 Summarizing Data

Business Scenario

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



2

Business Scenario

Create a report that lists the following for each department:

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

Below is a rough sketch of the desired report.

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

3

Business Data

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

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

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

4

Counting Rows That Meet a Specified Criterion

How do you determine the rows that ***do*** have *Manager* in **Job_Title**, as well as rows that ***do not***? You cannot use a WHERE clause to exclude either group.

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

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

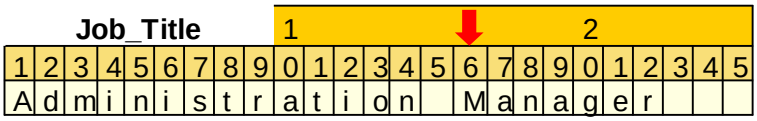
5

FIND Function

The *FIND function* returns the starting position of the first occurrence of a substring within a string (character value).

Find the starting position of the substring *Manager* in the character variable **Job_Title**.

```
find(Job_Title,"manager","i")
```



The value returned by the FIND function is 16.

FIND(*string*, *substring*<,*modifier(s)*><,*startpos*>)

6

Using Boolean Expressions

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

```
proc sql;
select Department, Job_Title,
       (find(Job_Title,"manager","i")>0)
       "Manager"
   from PHSUG.employee_information;
quit;
```

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

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

7

s103d13

Viewing the Output

Partial PROC SQL Output

Department	Job_Title	Manager
Administration	Administration Manager	1
Administration	Secretary I	0
Administration	Office Assistant II	0
Administration	Office Assistant III	0
Administration	Warehouse Assistant II	0
Administration	Warehouse Assistant I	0
Administration	Warehouse Assistant III	0
Administration	Security Guard II	0
Administration	Security Guard I	0
Administration	Security Guard II	0
Administration	Security Manager	1

8

Using Boolean Expressions

Part 2: Calculate the statistics requested.

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

9 s103d14

Viewing the Output

PROC SQL Output

Manager-to-Employee Ratios

Department	Managers	Employees	M/E Ratio
Accounts	3	14	21.4%
Accounts Management	1	8	12.5%
Administration	5	29	17.2%
Concession Management	1	10	10.0%
Engineering	1	8	12.5%
Executives	0	4	0.0%
Group Financials	0	3	0.0%
Group HR Management	3	15	20.0%
IS	2	23	8.7%
Logistics Management	6	8	75.0%
Marketing	6	14	42.9%
Purchasing	3	15	20.0%
Sales	0	201	0.0%
Sales Management	5	6	83.3%
Secretary of the Board	0	2	0.0%
Stock & Shipping	5	21	23.8%
Strategy	0	2	0.0%

10