The George Washington University
Department of Statistics

STAT 6197 – Spring 2020
Week 6 – February 21, 2020

Major Topic: Aggregating Data, and Combining SAS Data Sets (DATA/PROC Step)

Detailed Topics:

1. PROC SQL Basics
2. Aggregating/Summarizing by Groups - DATA Step vs. PROC SQL
3. Combining SAS Datasets - DATA Step vs. PROC SQL
4. Obtaining information from DICTIONARY tables Using PROC SQL
5. Getting Information from SASHELP Views – DATA Step vs. PROC SQL

Readings:

Relevant Chapters/Sections - Delwiche L, and Slaughter S. *The Little SAS Book: A Primer*, Fifth Edition Paperback – November 7, 2012

Exercises from Relevant Chapters/Sections - Ottesen RA, Delwiche LD, and Slaughter SJ. *Exercises and Projects for The Little SAS Book*, Fifth Edition Paperback – July 1, 2015

SET Statement, and By-Group Processing SET Statement

SAS(R) 9.4 SQL Procedure User's Guide Fourth Edition (PDF Document)

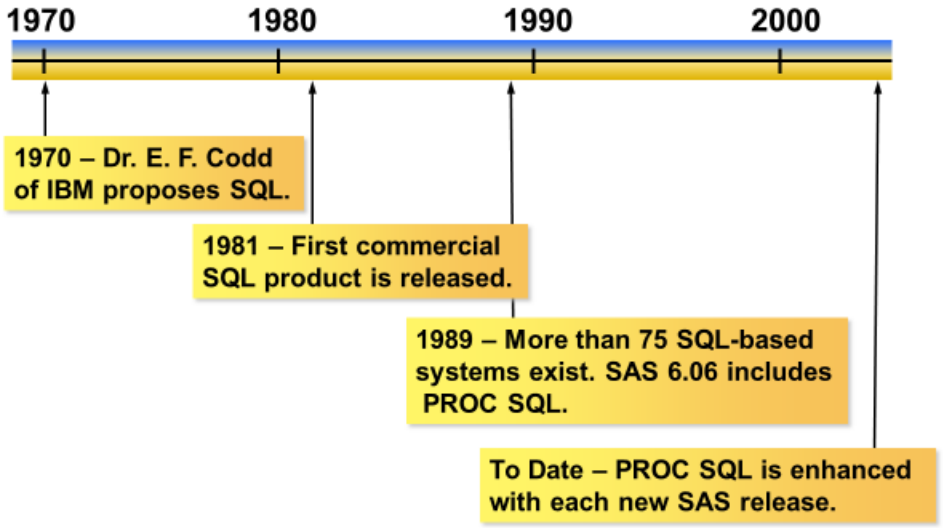
Dear Miss SASAnswers: A Guide to Efficient PROC SWQL Coding (Stroupe, and Jolley, 2009)

Many-to-Many Merges in the DATA Step (Heaton, 2008)

Merging Data Eight Different Ways (Franklin, 2009)

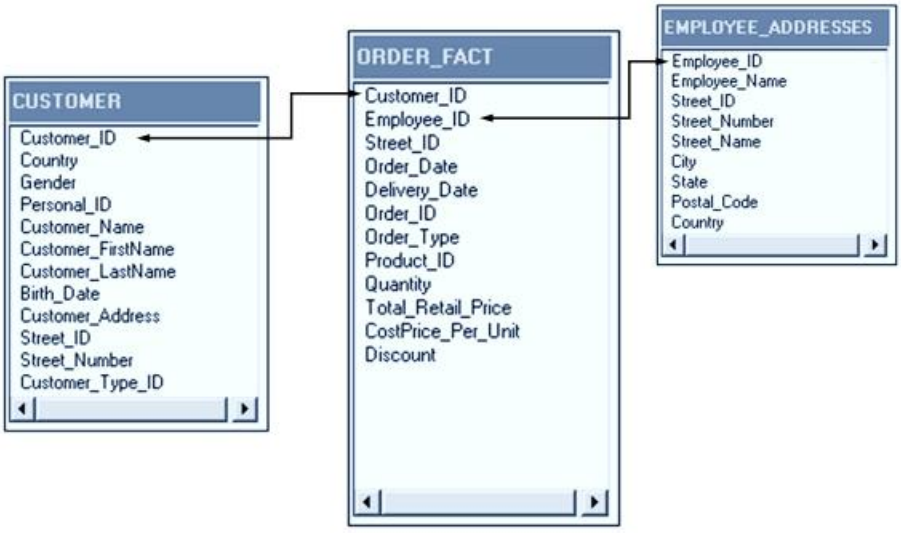
Exploring DICTIONARY Tables and SASHELP VIEWS (Lafler, 2009)

Structured Query Language: Timeline



5

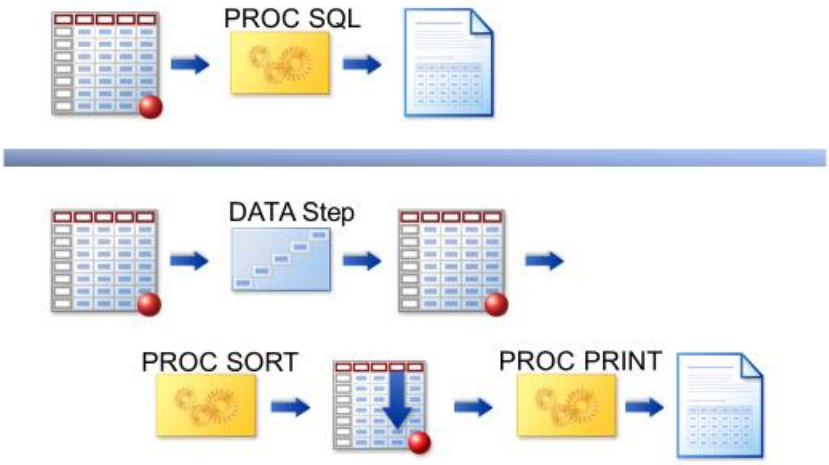
Relational Data



6

SQL Procedure versus Traditional SAS

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



9

Terminology

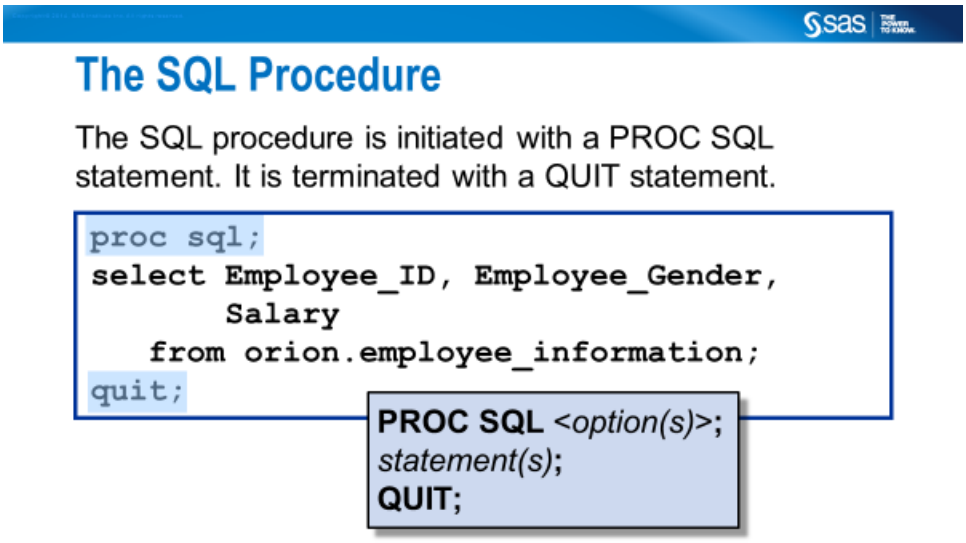
Data Processing	SAS	SQL
File	Data Set	Table
Record	Observation	Row
Field	Variable	Column

11

Uses for SQL: Read data from tables; add/modify/drop columns in tables, sort/filter rows, create tables/views, join tables/views, and create reports.

The Order of Clauses of the SELECT Statement within PROC SQL

- SELECT (So)
- FROM (Few)
- WHERE (Workers)
- GROUP BY (Go)
- HAVING (Home)
- Order BY (On-time)



- The SELECT statement has multiple clauses (e.g., SELECT, FROM, WHERE); PROC SQL, SELECT, QUIT statements are required
- The SELECT statement retrieves and displays data
- The SELECT clause specifies the columns and column order.
- The FROM clause specifies the data sources.
- You can query from 1 to 256 tables.
- Multiple statements can be included in a PROC SQL step.
- Each statement defines a process and is executed immediately.

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 orion.employee_information
   where Employee_Gender='F'
   order by Salary desc;
quit;
```

7 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

9

Calculating Columns Using SAS Dates: Step 3

Calculate **Age** based on today's date being 02JAN2013 and a **Birth_Date** value of 18AUG1980.

```
proc sql;
select Employee_ID, Employee_Gender,
       int((today()-Birth_Date)/365.25)
       as Age
  from orion.employee_information;
quit;
```

32

64

s102d09

Viewing the Output

Partial PROC SQL Output

```
                The SAS System

                 Employee
Employee ID      Gender                  Age
-------------------------------------------

     120101      M                        32
     120102      M                        39
     120103      M                        59
     120104      F                        54
     120105      F                        34
     120106      M                        64
     120107      F                        59
```

The values of **Age** vary based on the date that the program is executed.

65

Creating and Populating a Table

To define the structure of the **work.birth_months** table, use the SELECT list.

CREATE TABLE *table-name* **AS**
query-expression;

```
proc sql;
create table work.birth_months as
select Employee_ID, Birth_Date,
       month(Birth_Date) as
       Birth_Month,
       Employee_Gender
   from orion.employee_information;
describe table work.birth_months;
select * from work.birth_months;
quit;
```

column alias

69 s102d11

Displaying All Rows

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

Partial PROC SQL Output

```
          The SAS System
Department
----------------------------------------
Sales Management
Sales Management
Sales Management
Administration
Administration
Administration
Administration
Administration
Administration
Administration
```

Remove duplicates.

77 s102d12

Eliminating Duplicate Rows

Use the *DISTINCT* keyword to eliminate duplicate rows.

```
proc sql;
select distinct Department
   from orion.employee_information;
quit;
```

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

78 s102d13

Viewing the Output

Partial PROC SQL Output

```
                The SAS System

Department
-------------------------------
Accounts
Accounts Management
Administration
Concession Management
Engineering
Executives
Group Financials
Group HR Management
IS
Logistics Management
Marketing
Purchasing
Sales
```

one row per department

79

Subsetting with the WHERE Clause

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

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

WHERE *sql-expression*

83 s102d14

Viewing the Output

PROC SQL Output

The SAS System

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

84

Subsetting with Calculated Values

One solution is to repeat the calculation in the WHERE clause.

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

ANSI standard

92 s102d15

Subsetting with Calculated Values

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

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

SAS enhancement

93 s102d15

Viewing the Output

Partial PROC SQL Output

The SAS System

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

94

Using the Calculated Keyword

You can also use the CALCULATED keyword in other parts of a query.

```
proc sql;
select Employee_ID, Employee_Gender,
       Salary, Salary*.10 as Bonus,
       calculated Bonus/2 as Half
   from orion.employee_information
   where calculated Bonus<3000;
quit;
```

95

s102d15

Ordering Rows

Report 2: Order the results in descending order of maximum quarterly donation and then by employee ID in ascending order.

```
proc sql;
select Employee_ID,
       max(Qtr1,Qtr2,Qtr3,Qtr4)
   from orion.employee_donations
   where Paid_By="Cash or Check"
   order by 2 desc, Employee_ID;
quit;
```

Mix and match!

10 s103d02

Viewing the Output

Partial PROC SQL Output

ascending Employee_ID

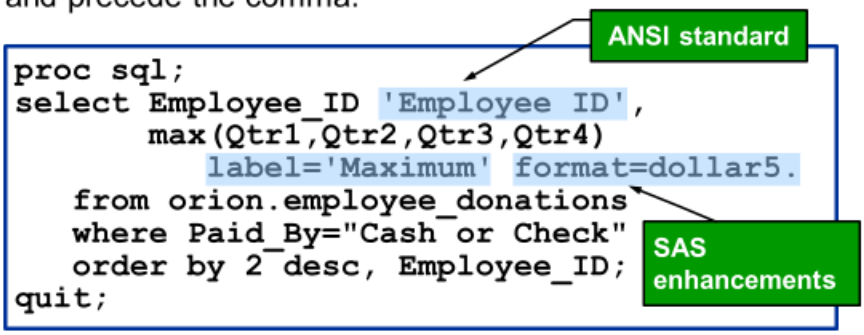
descending donation amounts

Employee ID	
120265	25
120736	25
120270	20
120679	20
120759	20
120760	20
120681	15
120734	15

11

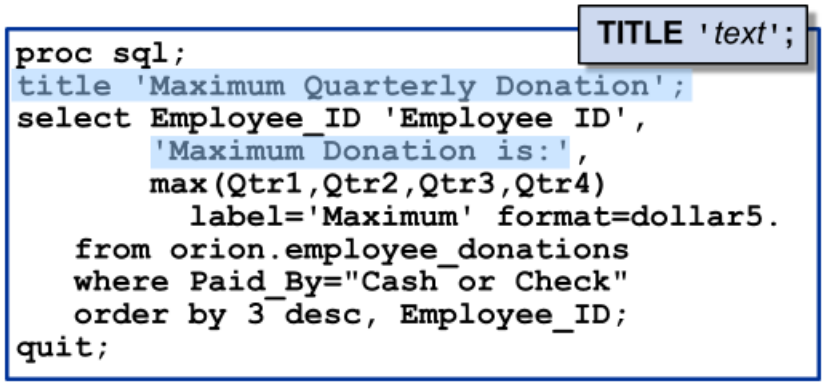
Adding Formats and Column Labels

Column labels and formats must follow the column name and precede the comma.



13 s103d03

Adding Titles and Constant Text



'*constant text*' <**AS** *alias*> <'column *label*'>

15 s103d04

Viewing the Output

Partial PROC SQL Output

```
        Maximum Quarterly Donation

Employee ID                              Maximum
-------------------------------------------------
     120265  Maximum Donation is:          $25
     120736  Maximum Donation is:          $25
     120270  Maximum Donation is:          $20
     120679  Maximum Donation is:          $20
     120759  Maximum Donation is:          $20
     120760  Maximum Donation is:          $20
     120681  Maximum Donation is:          $15
     120734  Maximum Donation is:          $15
```

title

constant text

16

Adding a Row Number

PROC SQL <*option(s)*>;

```
proc sql number;
title 'Maximum Quarterly Donation';
select Employee_ID 'Employee ID',
       'Maximum Donation is:',
       max(Qtr1,Qtr2,Qtr3,Qtr4)
         label='Maximum' format=dollar5.
   from orion.employee_donations
   where Paid_By="Cash or Check"
   order by 3 desc, Employee_ID;
quit;
```

s103d05

17

Summary Functions: Across a Row

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

```
sum(Qtr1,Qtr2,Qtr3,Qtr4)
```

Partial **orion.employee_donations**

SAS Data Set

Employee_ID	Qtr1	Qtr2	Qtr3	Qtr4
120736	25	.	.	20
120759	15	20	5	.
120681	10	10	5	15
120679	.	20	5	15
120777	5	15	5	15

28

Summary Functions: Across a Row

Total each employee's annual cash donations.

SUM(*col1*, …, *coln*)

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

29

s103d06

Viewing the Output

Partial PROC SQL Output

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

30

Summary Functions: Down a Column

Calculate the total of all charitable donations in quarter 1.

SUM(*argument*)

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

PROC SQL Output

Total Quarter 1 Donations
1515

35

s103d07

Alternative Method: MEANS Procedure

Calculate the total all charitable donations in quarter 1.

```
proc means data=orion.employee_donations
           sum maxdec=0;
   var Qtr1;
run;
```

PROC MEANS Output

```
Analysis Variable : Qtr1

        Sum
------------
       1515
------------
```

36 s103d07

Summary Functions: Recap

How a summary function works in SQL depends on the number of columns specified in the argument list.

- If the summary function specifies more than one column, the statistic is calculated for the row (using values from the listed columns).

```
sum(Qtr1,Qtr2,Qtr3,Qtr4)
```

- If the summary function specifies only one column, the statistic is calculated for the column (using values from one or more rows).

```
sum(Qtr1)
```

37

Commonly Used Summary Functions

Both ANSI SQL and SAS functions can be used in PROC SQL.

SQL	SAS	Description
AVG	MEAN	Returns the mean (average) value.
COUNT	FREQ, N	Returns the number of nonmissing values.
MAX	MAX	Returns the largest value.
MIN	MIN	Returns the smallest nonmissing value.
SUM	SUM	Returns the sum of nonmissing values.
	NMISS	Counts the number of missing values.
	STD	Returns the standard deviation.
	VAR	Returns the variance.

38

Summary Functions: COUNT Function

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

COUNT(*argument*)

```
proc sql;
select count(*) as Count
   from orion.employee_information
   where Employee_Term_Date is missing;
quit;
```

Argument value	Counts
* (asterisk)	All rows in a table or group
A column name	The number of nonmissing values in that column

40 s103d08a

Viewing the Output

The COUNT function returns the number of distinct nonmissing **Manager_ID** values returned by a query.

```
proc sql;
select count(distinct Manager_ID) as Count
   from orion.employee_information
   where Employee_Term_Date is missing;
quit;
```

PROC SQL Output

Count
52

41 s103d08b

Objectives

- Define First. and Last. processing.
- Calculate an accumulating total for groups of data.
- Use a subsetting IF statement to output selected observations.

32

Business Scenario

The **Salary** variable represents the portion of the employee's salary that is allocated to a project. An analyst would like to create a new data set and listing report that has the salary totals for each department.

Partial
orion.specialsals

Employee_ ID	Salary	Dept
110004	42000	HUMRES
110009	34000	ENGINR
110011	27000	FINANC
110036	20000	ENGINR
110037	19000	ENGINR

deptsals

Dept	DeptSal
ADMIN	410000
ENGINR	73000
FINANC	318000
HUMRES	42000
SALES	373000

38

Processing Needed

Dept	Salary
ADMIN	20000
ADMIN	100000
ADMIN	50000
ENGINR	25000
ENGINR	20000
ENGINR	23000
ENGINR	27000
FINANC	10000
FINANC	12000

DeptSal

Step 1 Sort the data by **Dept**.

40

Processing Needed

Dept	Salary
ADMIN	20000
ADMIN	100000
ADMIN	50000
ENGINR	25000
ENGINR	20000
ENGINR	23000
ENGINR	27000
FINANC	10000
FINANC	12000

DeptSal
170000
95000
22000

Step 2 Summarize the observations by department groups.

41

Processing Needed

Dept	Salary	DeptSal
ADMIN	20000	
ADMIN	100000	
ADMIN	50000	170000
ENGINR	25000	
ENGINR	20000	
ENGINR	23000	
ENGINR	27000	95000
FINANC	10000	
FINANC	12000	22000

Step 3 Write only the last observation of each BY group.

42

BY-Group Processing (Review)

The BY statement in the DATA step enables SAS to process data in groups.

```
proc sort data=orion.specialsals
          out=salsort;
   by Dept;
run;

data deptsals(keep=Dept DeptSal);
   set salsort;
   by Dept;
   <additional SAS statements>
run;
```

Step 1: Sort by Dept.

Step 2: Process by Dept groups.

First.*BY-variable*
Last.*BY-variable*

A BY statement in a DATA step creates two temporary variables for each variable listed in the BY statement.

43

First. / Last. Values: First DATA Step Iteration

Dept	Salary
ADMIN	20000
ADMIN	100000
ADMIN	50000
ENGINR	25000
ENGINR	20000
ENGINR	23000
ENGINR	27000
FINANC	10000
FINANC	12000

First.Dept
1

Last.Dept
?

How can SAS determine the value for Last.Dept?

37

...

First. / Last. Values: First DATA Step Iteration

Dept	Salary
ADMIN	20000
ADMIN	100000
ADMIN	50000
ENGINR	25000
ENGINR	20000
ENGINR	23000
ENGINR	27000
FINANC	10000
FINANC	

First.Dept
1

Last.Dept
0

SAS looks ahead at the next observation to determine Last.Dept value.

38

...

First. / Last. Values: Second DATA Step Iteration

Dept	Salary
ADMIN	20000
ADMIN	100000
ADMIN	50000
ENGINR	25000
ENGINR	20000
ENGINR	23000
ENGINR	27000
FINANC	10000
FINANC	12000

First.Dept
0

Last.Dept
0

39

...

First. / Last. Values: Third DATA Step Iteration

Dept	Salary
ADMIN	20000
ADMIN	100000
ADMIN	50000
ENGINR	25000
ENGINR	20000
ENGINR	23000
ENGINR	27000
FINANC	10000
FINANC	12000

First.Dept
0

Last.Dept
1

40

...

First. / Last. Values: Fourth DATA Step Iteration

Dept	Salary
ADMIN	20000
ADMIN	100000
ADMIN	50000
ENGINR	25000
ENGINR	20000
ENGINR	23000
ENGINR	27000
FINANC	10000
FINANC	12000

First.Dept
1

Last.Dept
0

41

3.03 Short Answer Poll

What are the values for **First.Dept** and **Last.Dept** when the DATA step is processing the observation indicated by the arrow?

Dept	Salary
ADMIN	20000
ADMIN	100000
ADMIN	50000
ENGINR	25000
FINANC	10000
FINANC	12000

First.Dept
?

Last.Dept
?

42

3.03 Short Answer Poll – Correct Answer

What are the values for **First.Dept** and **Last.Dept** when the DATA step is processing the observation indicated by the arrow?

Dept	Salary
ADMIN	20000
ADMIN	100000
ADMIN	50000
ENGINR	25000
FINANC	10000
FINANC	12000

First.Dept
1

Last.Dept
1

First.Dept and Last.Dept are both 1. This happens when a group consists of a single observation.

43

First. / Last. Values: DATA Step Iterations

Dept	Salary	First.Dept	Last.Dept	_N_
ADMIN	20000	1	0	1
ADMIN	100000	0	0	2
ADMIN	50000	0	1	3
ENGINR	25000	1	0	4
ENGINR	20000	0	0	5
ENGINR	23000	0	0	6
ENGINR	27000	0	1	7
FINANC	10000	1	0	8
FINANC	12000	0	1	9

44

Summarizing Data by Groups

Step 1 Initialize

```
data deptsals(keep=Dept DeptSal);
   set SalSort;
   by Dept;
   if First.Dept then DeptSal=0;
   <additional SAS statements>
run;
```

- The condition is considered true when **First.Dept** has a value of *1*.

45

Summarizing Data by Groups

Step 2 Accumulate

```
data deptsals(keep=Dept DeptSal);
   set SalSort;
   by Dept;
   if First.Dept then DeptSal=0;
   DeptSal+Salary;
   <additional SAS statements>
run;
```

46

Summarizing Data by Groups

Step 3 Output

```
data deptsals(keep=Dept DeptSal);
   set SalSort;
   by Dept;
   if First.Dept then DeptSal=0;
   DeptSal+Salary;
   if Last.Dept;
run;
```

- The subsetting IF defines a condition that the observation must meet to be further processed by the DATA step.

47 psm04d04

Summarizing Data by Groups

```
proc print data=deptsals noobs;
run;
```

PROC PRINT Output

Dept	DeptSal
ADMIN	410000
ENGINR	163000
FINANC	318000
HUMRES	181000
SALES	373000

Partial SAS Log

```
NOTE: There were 39 observations read
      from the data set WORK.SALSORT.
NOTE: The data set WORK.DEPTSALS has 5
      observations and 2 variables.
```

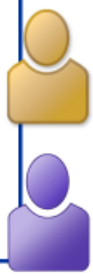
48

Business Scenario

Each employee listed in **orion.projsals** is assigned to a special project. A business analyst would like to see the salary totals from each department for each special project.

Partial **orion.projsals**

Employee_ ID	Salary	Proj	Dept
110004	42000	EZ	HUMRES
110009	34000	WIN	ENGINR
110011	27000	WIN	FINANC
110036	20000	WIN	ENGINR
110037	19000	EZ	ENGINR
110048	19000	EZ	FINANC
110077	27000	CAP1	ADMIN
110097	20000	EZ	ADMIN
110107	31000	EZ	ENGINR



51

Accumulator Variable for BY Groups

In order to create an accumulator variable for BY groups, the beginning and end of each BY group must be determined.

County	Town	Population	Total Population
Androscoggin	Auburn	23203	23203
Androscoggin	Lewiston	35690	58893
Cumberland	Brunswick	21172	21172
Cumberland	Portland	64249	85421
Cumberland	Scarborough	16970	102391
Cumberland	South Portland	23324	125715
Kennebec	Augusta	18560	18560
Penobscot	Bangor	31473	31473
York	Biddeford	20942	20942
York	Sanford	20806	41748

accumulator variable by County

59

BY-Group Processing

In the DATA step, SAS identifies the beginning and end of each BY group by creating two temporary variables for each BY variable: the **FIRST.** and **LAST.** variables.

```
data TopCounties;
   set Top10Town;
   by County;
run;
```

Data must be sorted or indexed by County.

First.County and Last.County are created.

These temporary variables are available for DATA step programming but are not added to the output data set.

60

BY-Group Processing

- The **FIRST.** variable is set to 1 when an observation is the first in a BY group. Otherwise, it equals 0.
- The **LAST.** variable is set to 1 when an observation is the last in a BY group. Otherwise, it equals 0.

County	Population	first.County	last.County
Androscoggin	23203	1	0
Androscoggin	35690	0	1
Cumberland	21172	1	0
Cumberland	64249	0	0
Cumberland	16970	0	0
Cumberland	23324	0	1
Kennebec	18560	1	1

61

BY-Group Processing

The following program resets the accumulator variable at the beginning of each BY group and outputs only at the end of each BY group.

```
data TopCounties;
   set Top10Town;
   by County;
   if first.County then TotalPopulation=0;
   TotalPopulation+Population;
   if last.County=1;
   keep County TotalPopulation;
run;
```

	County	TotalPopulation
1	Androscoggin	58893
2	Cumberland	125715
3	Kennebec	18560
4	Penobscot	31473
5	York	41748

64

Multiple BY-Group Variables

A **FIRST.** variable and a **LAST.** variable is created for each variable in the BY statement.

```
data CityDonate;
  set Donations;
  by State City;
run;
```

Data must be sorted or indexed by State and City.

First.State, Last.State, First.City, and Last.City are created.

When you use more than one variable in the BY statement, a change in the primary variable forces the **LAST.** variable=1 for the secondary variable.

65

Exercise 3 Solution

State	City	Donation	Total Donation	first. State	last. State	first. City	last. City
NC	Charlotte	2000	15000	1	0	1	0
NC	Charlotte	9000		0	0	0	0
NC	Charlotte	4000		0	0	0	1
NC	Greenville	6000	9000	0	0	1	0
NC	Greenville	3000		0	1	0	1
SC	Greenville	5000	7000	1	0	1	0
SC	Greenville	2000		0	0	0	1
SC	Pelzer	5000	5000	0	1	1	1

67

What Must Happen When?

There is a three-step process for using the DATA step to summarize grouped data.

Step 1	Initialize: Set the accumulating variable to zero at the start of each BY group.
Step 2	Accumulate: Increment the accumulating variable with a sum statement (automatically retains).
Step 3	Output: Write only the last observation of each BY group.

38

Multiple BY-Group Variables

```
data CityDonate;
   set Donations;
   by State City;
   if first.City=1 then TotalDonation=0;
   TotalDonation+Donation;
   if last.City=1;
run;
```

	state	city	donation	TotalDonation
1	NC	Charlotte	4000	15000
2	NC	Greenville	3000	9000
3	SC	Greenville	2000	7000
4	SC	Pelzer	5000	5000

68

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 orion.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.

49

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.
```

51

Grouping Data

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

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

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

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

s103d10

58

Viewing the Output

PROC SQL Output

```
      Average Salary by Gender

       Employee
       Gender          Average
       ------------------------
       F             37002.88
       M             43334.26
```

59

Analyzing Groups of Data

```
proc sql;
select Department, count(*) as Count
   from orion.employee_information
   group by Department;
quit;
```

64 s103d11

Step 2

Control the result to include only the departments that have at least 25 people, with the departments in decreasing order.

Department	Count
Accounts	17
Accounts Management	9
Administration	34
Concession Management	11
Engineering	9
Executives	4
Group Financials	3
Group HR Management	18
IS	25
Logistics Management	
Marketing	
Purchasing	
Sales	
...	

Department	Count
Sales	201
Administration	34
Stock & Shipping	26
IS	25

65

Selecting Groups with the HAVING Clause

The *HAVING clause* subsets groups based on the expression value.

```
proc sql;
select Department, count(*) as Count
   from orion.employee_information
   group by Department
   having Count ge 25
   order by Count desc;
quit;
```

GROUP BY *group-by-item <,…,group-by-item>*
HAVING *sql-expression*

66 s103d12

Viewing the Output

PROC SQL Output

Department	Count
Sales	201
Administration	34
Stock & Shipping	26
IS	25

67

WHERE Clause versus HAVING Clause

- The WHERE clause is evaluated ***before*** a row is available for processing and determines which individual rows are available for grouping.

  WHERE *sql-expression*

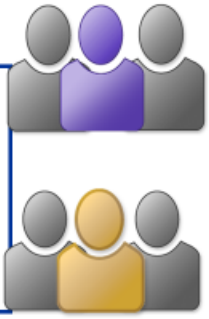
- The HAVING clause is processed ***after*** the GROUP BY clause and determines which groups are displayed.

  HAVING *sql-expression*

68

Business Scenario: Desired Output

Create a new data set, **pdsals**, using the SQL procedure. The data set results show the number of employees and salary totals from each department for each special project.



Proj	Dept	Dept Sal	Num Emps
CAP1	ADMIN	70000	2
EZ	ADMIN	83000	3
EZ	ENGINR	109000	4
EZ	FINANC	122000	3
EZ	HUMRES	178000	5
NGEN	ADMIN	37000	2

52

Creating a Data Set with the SQL Procedure

Use the CREATE TABLE statement to create a data set from the results of a query.

```
proc sql;
create table pdsals as
   select proj, dept,
          sum(Salary) as DeptSal,
          count(*) as Numemps
   from orion.projsals
   group by proj, dept
   order by proj, dept;
```

CREATE TABLE *table-name* **AS** *query-expression*;

53

psm04d05

continued...

Analyzing Multiple Groups of Data

```
proc sql;
create table pdsals as
   select proj, dept,
          sum(Salary) as DeptSal,
          count(*) as Numemps
   from orion.projsals
   group by proj, dept
   order by proj, dept;
```

Proj	Dept	Salary
CAP1	ADMIN	43000
CAP1	ADMIN	27000
EZ	ADMIN	32000
EZ	ADMIN	31000

70,000

54

psm04d05

continued...

Analyzing Multiple Groups of Data

```
proc sql;
create table pdsals as
   select proj, dept,
          sum(Salary) as DeptSal,
          count(*) as Numemps
   from orion.projsals
   group by proj, dept
   order by proj, dept;
```

COUNT(*argument*)

Argument value	Counts
* (asterisk)	All rows in a table or group
A column name	The number of nonmissing values in that column

55

psm04d05

Analyzing Multiple Groups of Data

```
proc sql;
create table pdsals as
   select proj, dept,
          sum(Salary) as DeptSal,
          count(*) as Numemps
   from orion.projsals
   group by proj, dept
   order by proj, dept;
```

COUNT(*)

Proj	Dept	Salary
CAP1	ADMIN	43000
CAP1	ADMIN	27000
EZ	ADMIN	32000
EZ	ADMIN	31000

2

56

psm04d05

Display Data Set Results

```
select * from pdsals;
quit;
```

Partial PROC SQL Output (14 Total Observations)

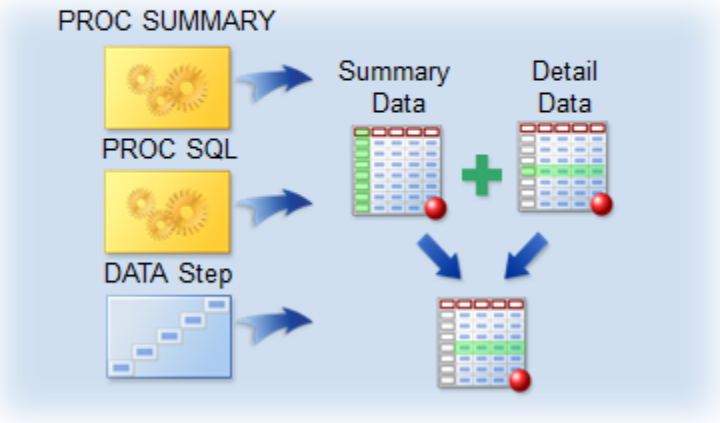
Proj	Dept	DeptSal	Numemps
CAP1	ADMIN	70000	2
EZ	ADMIN	83000	3
EZ	ENGINR	109000	4
EZ	FINANC	122000	3

57

psm04d05

Introduction

This section explores how to combine summary and detail data using PROC SUMMARY, PROC SQL, and the DATA step.



79

Business Scenario

Use **orion.totalsalaries** to calculate percentages of the total company payroll for each manager. This requires a few steps.

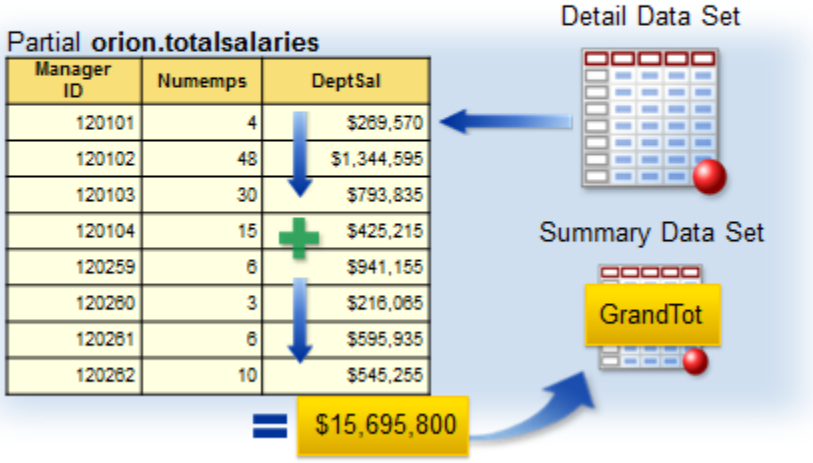
Partial **orion.totalsalaries**

Manager ID	Numemps	DeptSal	GrandTot	Percent
120101	4	$269,570	$15,695,800	1.72%
120102	48	$1,344,595	$15,695,800	8.57%
120103	30	$793,835	$15,695,800	5.06%
120104	15	$425,215	$15,695,800	2.71%
120259	6	$941,155	$15,695,800	6.00%
120260	3	$216,065	$15,695,800	1.38%
120261	6	$595,935	$15,695,800	3.80%
120262	10	$545,255	$15,695,800	3.47%

81

Business Scenario

Calculate the grand total of **DeptSal** and store it in a summary data set.



82

Business Scenario

Combine the summary data with each row of the detailed data. Divide each detail amount by the grand total to calculate the percent of payroll by manager.

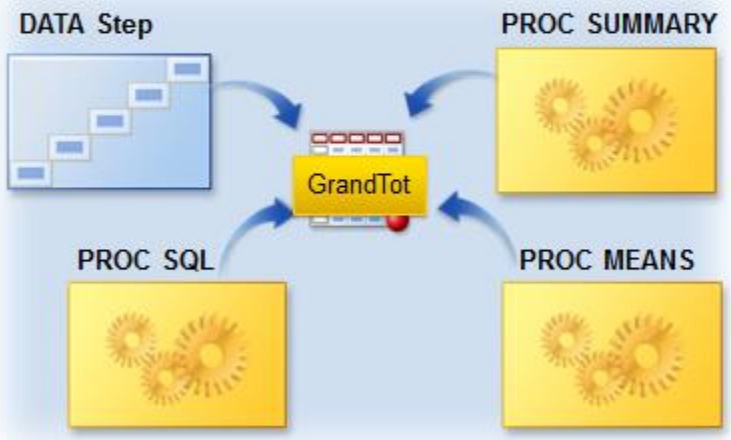
Partial **orion.totalsalaries**

Manager ID	Numemps	Deptsal		GrandTot		Percent
120101	4	$269,570	÷	$15,695,800	=	1.72%
120102	48	$1,344,595		$15,695,800		8.57%
120103	30	$793,835		$15,695,800		5.06%
120104	15	$425,215		$15,695,800		2.71%
120259	6	$941,155		$15,695,800		6.00%
120260	3	$216,065		$15,695,800		1.38%
120261	6	$595,935		$15,695,800		3.80%
120262	10	$545,255		$15,695,800		3.47%

83

Creating a Summary Data Set

There are several ways to create the summary data set.



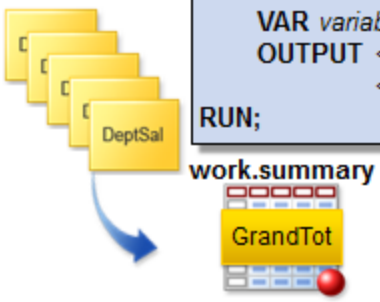
84

Creating a Summary Data Set

PROC SUMMARY generates descriptive statistics. The OUTPUT statement with the OUT= option creates a SAS data set with the summary statistics.

```
proc summary data=orion.totalsalaries;
   var DeptSal;
   output out=summary sum=GrandTot;
run;
```

```
PROC SUMMARY <option(s)><statistic-keyword(s)>;
    VAR variable(s);
    OUTPUT <OUT=SAS-data-set>
           <output-statistic-specification(s)>;
RUN;
```

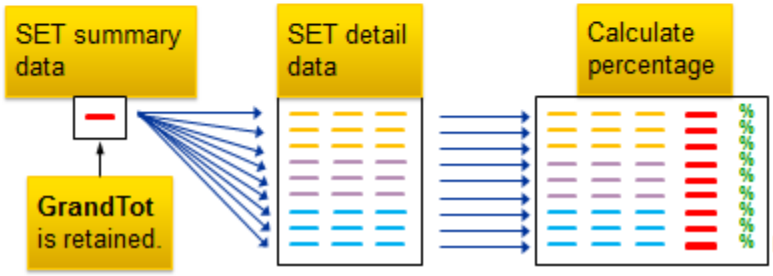


85 p307d10

Combining the Summary and Detail Data

Use two SET statements in the DATA step to combine the summary and detailed data.

```
data percent;
   if _N_=1 then
      set summary(keep=GrandTot);
   set orion.totalsalaries;
   Percent=DeptSal/GrandTot;
   format Percent percent8.2;
run;
```



86 p307d10

Creating a Summary Data Set

Partial PROC PRINT Output

Percentage of Total Salaries for Each Manager

GrandTot	ManagerID	Numemps	DeptSal	Percent
$15,695,800	120101	4	$269,570	1.72%
$15,695,800	120102	48	$1,344,595	8.57%
$15,695,800	120103	30	$793,835	5.06%
$15,695,800	120104	15	$425,215	2.71%
$15,695,800	120259	6	$941,155	6.00%

p307d10

97

Combining Summary and Detail Data Using the SQL Procedure

Calculate the summary and merge it with the detail data in a single PROC SQL step.

```
proc sql;
   create table percentsql as
   select ManagerID,
          DeptSal,
          sum(DeptSal) as GrandTot 'Grand Total',
          DeptSal / calculated GrandTot
                  as Percent format=percent8.2
   from orion.totalsalaries;
quit;
```

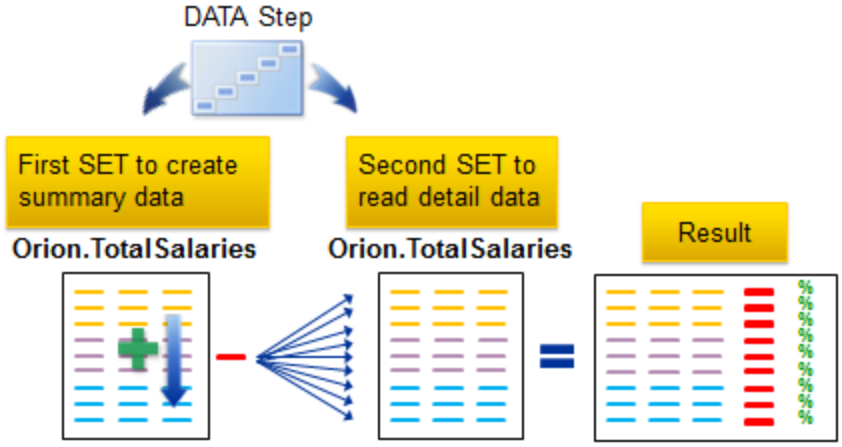
```
NOTE: The query requires remerging summary statistics back with the original data.
```

p307d12

103

Combining the Summary and Detail Data Using SET Statements

Use two SET statements to calculate the **GrandTot** variable and calculate the percentages.



104

Combining the Summary and Detail Data Using SET Statements

Create summary data.

```
data percent(drop=i);
   if _N_=1 then do i=1 to TotObs;
   set orion.totalsalaries(keep=DeptSal) nobs=TotObs;
      GrandTot+DeptSal;
   end;
   set orion.totalsalaries;
   Percent=DeptSal/GrandTot;
   format Percent percent8.2;
run;
```

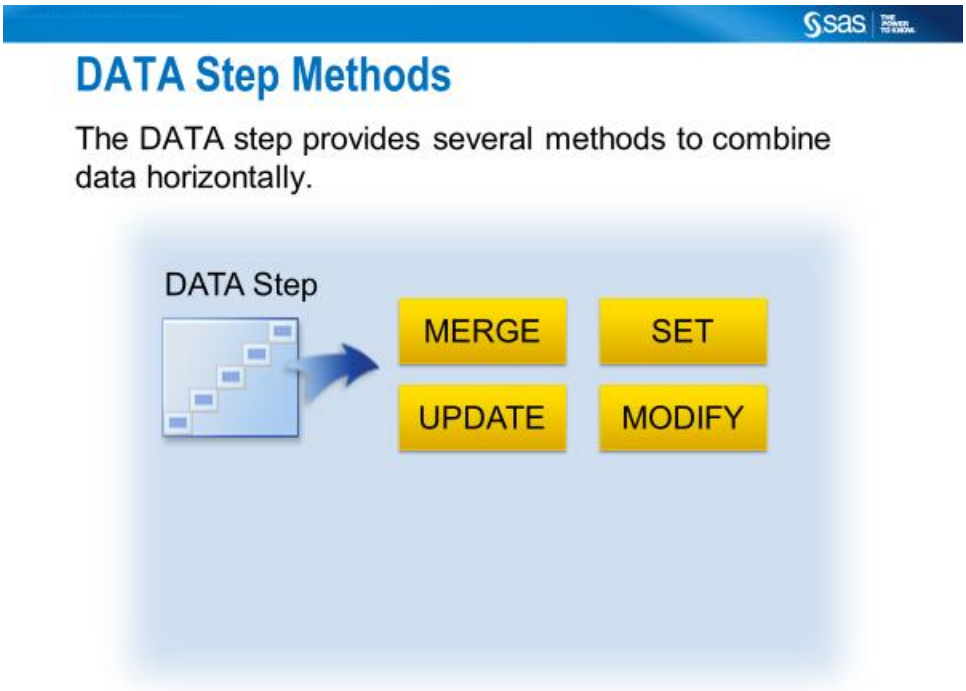
Read detail data and calculate Percent.

Partial SAS Log

```
NOTE: There were 53 observations read from the data set ORION.TOTALSALARIES.
NOTE: There were 53 observations read from the data set ORION.TOTALSALARIES.
NOTE: The data set WORK.PERCENT has 53 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time          0.07 seconds
      cpu time           0.03 seconds
```

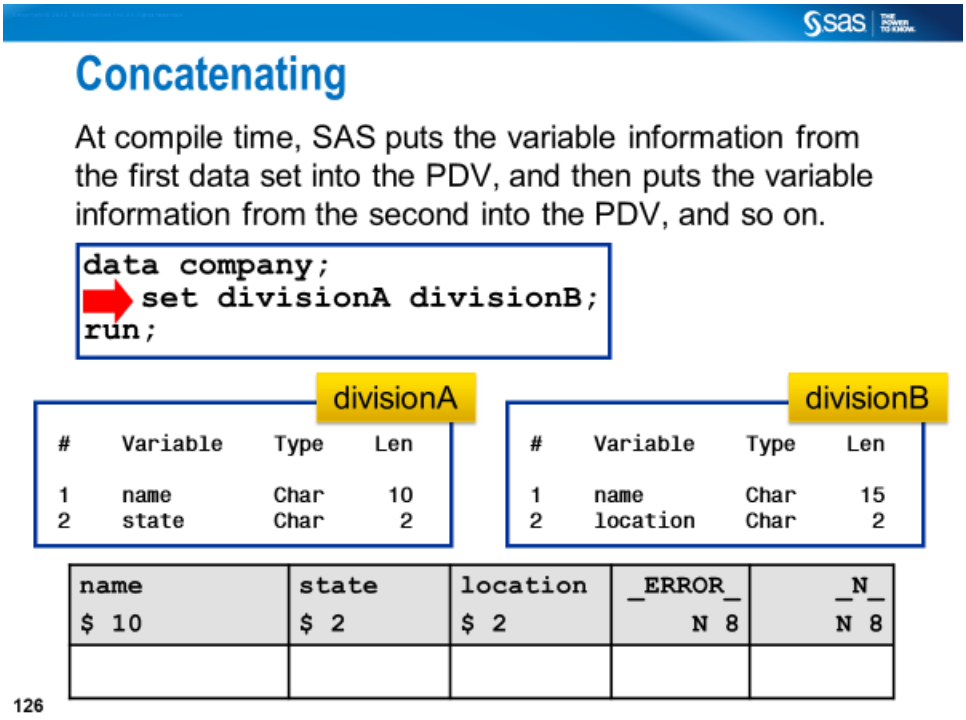
p307d14

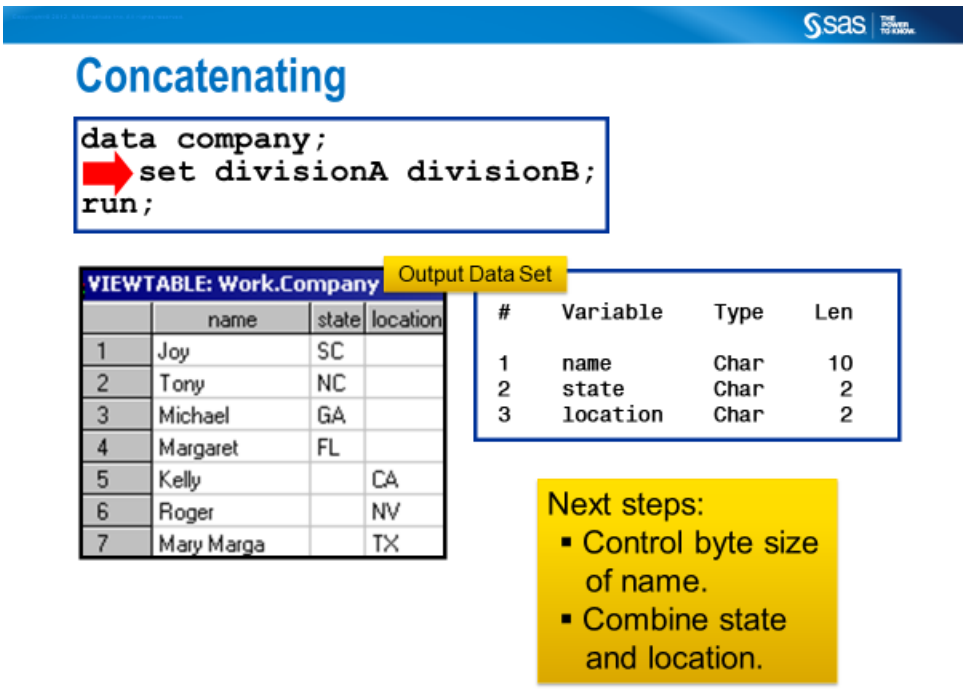
105



You can read observations sequentially using the SET, MERGE, UPDATE, or MODIFY statements.

Combining Data Using the SET Statement (Concatenation)





The length of the variables in the first input data set will determine the length of the variables in the output data set.

LENGTH Statement

The LENGTH statement specifies the number of bytes for storing variables.

```
data company;
   length name $ 15;
   set divisionA divisionB;
run;
```

VIEWTABLE: Work.Company

	name	state	location
1	Joy	SC	
2	Tony	NC	
3	Michael	GA	
4	Margaret	FL	
5	Kelly		CA
6	Roger		NV
7	Mary Margaret		TX

Output Data Set

#	Variable	Type	Len
1	name	Char	15
2	state	Char	2
3	location	Char	2

128

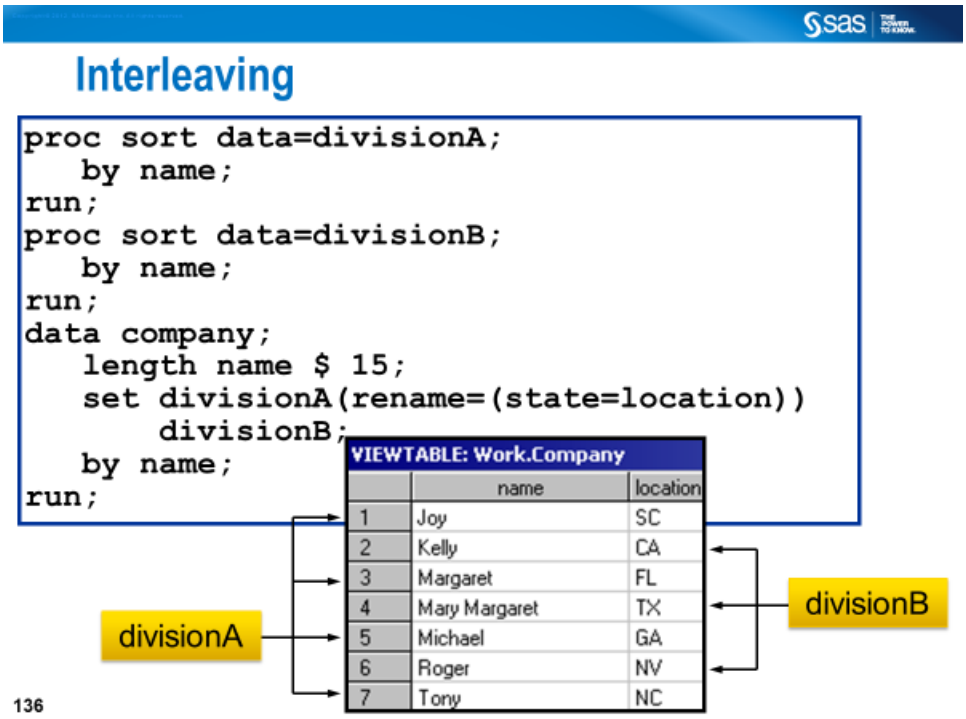
RENAME= Option

The RENAME= data set option changes the names of variables.

```
data company;
   length name $ 15;
   set divisionA(rename=(state=location))
       divisionB;
run;
```

- The RENAME= option specifies the variable that you want to rename equal to the new name of the variable.
- The list of variables to rename must be enclosed in parentheses.

131

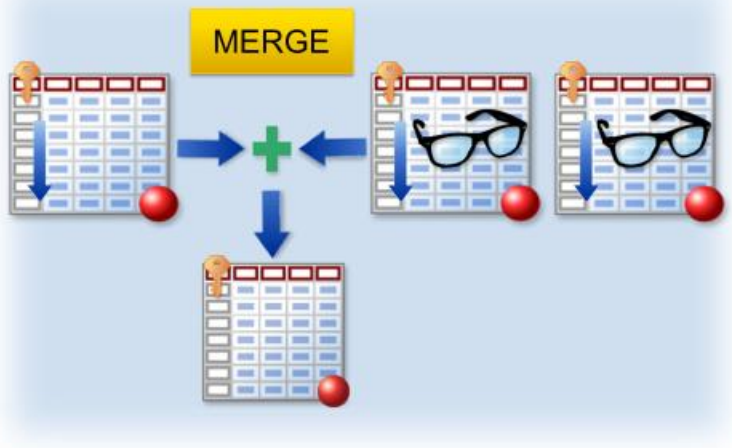


The observations in the new data set are arranged by the values of the BY variable or variables. Then, within each BY group, they are arranged by the order of the data sets in which they occur. In another words, when two or more data sets are concatenated using a SET statement followed by a BY statement, as shown below, files are interleaved.

Combining Data Using the MERGE Statement

DATA Step Merge

In a DATA step merge, the data sets must be sorted or indexed by the key columns.



9

DATA Step Merge

Merge **orion.country** with **orion.continent** by **ContinentID**.

```
data countryinfo;
   merge orion.country
         orion.continent;
   by ContinentID;
run;
```

Partial **orion.country**

Country	...	Continent ID	...
CA	...	91	...
US	...	91	...
DE	...	93	...
ZA	...	94	...

Partial **orion.continent**

Continent ID	Continent Name
91	North America
93	Europe
94	Africa
95	Asia

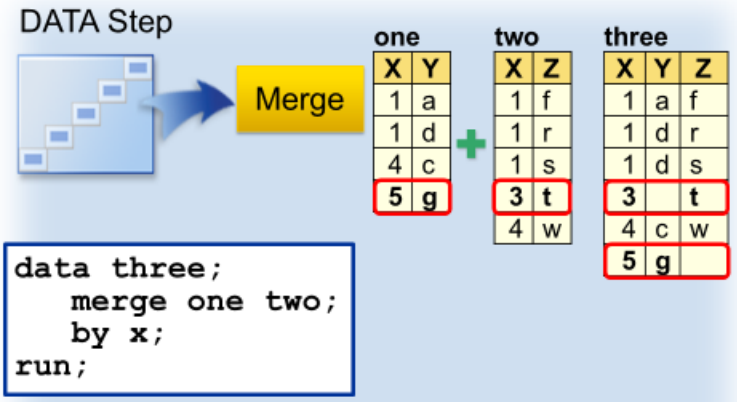
Partial results

Country	Continent ID	Continent Name
CA	91	North America
US	91	North America
DE	93	Europe
ZA	94	Africa

10

DATA Step Merge: A Sequential Process

Two observations represent the nonmatches.




```
data three;
   merge one two;
   by x;
run;
```

16

p307d02

Default Action with a DATA Step Merge

Variable name, type, and length attributes are established by the first data set.

Example:

```
data c;
   merge a b;
   by X;
run;
```

a

X	y
N 8	$ 4
1	ABCD
1	ZZZZ

b

X	Y
N 8	$ 6
1	EFGHIJ

c

X	y
N 8	$ 4
1	EFGH
1	ZZZZ

33

p307d06

Default Action with a DATA Step Merge

In a one-to-one or one-to-many merge, variable values might come from the last data set.

Example:

```
data c;
   merge a b;
   by X;
run;
```

a

X N 8	y $ 4
1	ABCD
1	ZZZZ

b

X N 8	Y $ 6
1	EFGHIJ

c

X N 8	y $ 4
1	EFGH
1	ZZZZ

```
WARNING: Multiple lengths were specified for the variable y by input data set(s).
         This can cause truncation of data.
```

p307d06

34

One-to-Many Matches

The DATA step and PROC SQL sometimes produce the same results.

```
data dsthree;
   merge one two;
   by X;
run;
```

one

X	Y
1	a
2	b

two

X	Z
1	f
1	r
2	g

```
proc sql;
   create table sqlthree as
   select one.X, Y, Z
      from one, two
      where one.X=two.X;
quit;
```

One-to-many match

dsthree

X	Y	Z
1	a	f
1	a	r
2	b	g

=

sqlthree

X	Y	Z
1	a	f
1	a	r
2	b	g

21

P307d03

Merging

- The MERGE statement joins observations from two or more SAS data sets into single observations.
- The BY statement specifies the common variables to match-merge observations. The variables in the BY statement must be common to all data sets.

```
data combine;
   merge revenue expense;
   by name;
   profit=revenue-expense;
run;
```

- The data sets listed in the MERGE statement must be sorted in the order of the values of the variables that are listed in the BY statement, or they must have an appropriate index.

139

IN= Option

The IN= option creates a variable that indicates whether the data set contributed data to the current observation.

```
data combine1;
   merge revenue1(in=rev)
         expense1(in=exp);
   by name;
   profit=revenue-expense;
run;
```

Within the DATA step, the value of the variable is 1 if the data set contributed to the current observation, and 0 if the data set did not contribute to the current observation.

144

When you combine two data sets, you can use IN= data set options to track which of the original data sets contributes to each observation in the new data set.

IN= Option

```
data combine1;
   merge revenue1(in=rev)
         expense1(in=exp);
   by name;
   profit=revenue-expense;
run;
```

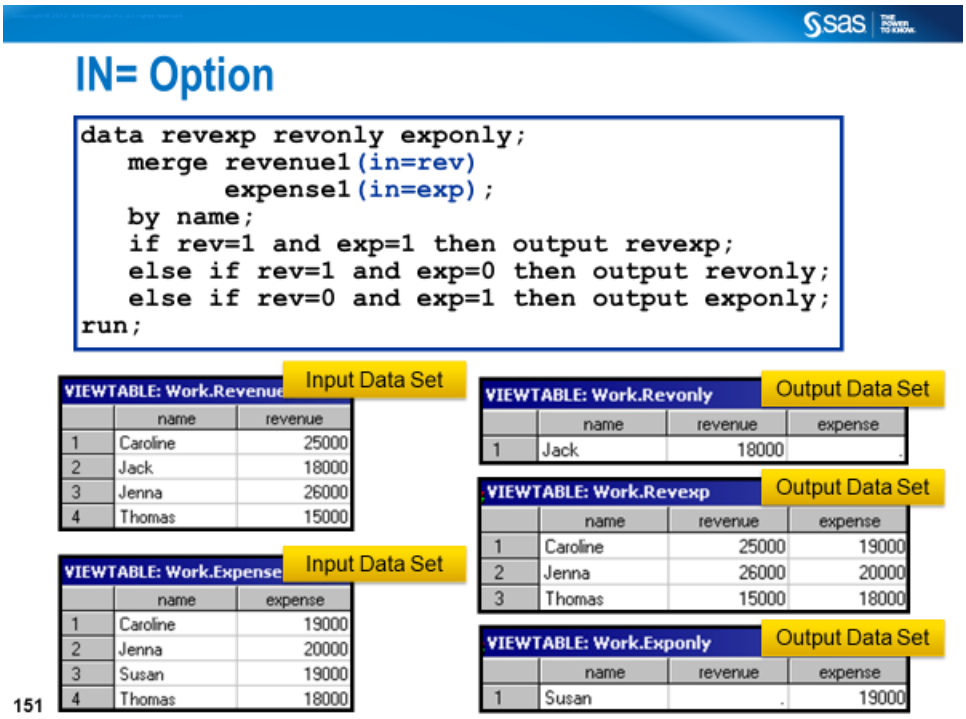
name $ 10	revenue N 8	expense N 8	profit N 8	rev N 8	exp N 8	_ERROR_ N 8	_N_ N8
Caroline	25000	19000	6000	1	1	0	1
Jack	18000	.	.	1	0	0	2
Jenna	26000	20000	6000	1	1	0	3
Susan	.	19000	.	0	1	0	4
Thomas	15000	18000	-3000	1	1	0	5

147

IN= Option

Possible Scenarios	
`if rev=1 and exp=1;` `if rev and exp;`	Matches only
`if rev=1 and exp=0;` `if rev and not exp;`	Nonmatches from **revenue1**
`if rev=0 and exp=1;` `if not rev and exp;`	Nonmatches from **expenses1**
`if rev=1;` `if rev;`	All observations from **revenue1** (matches and nonmatches)
`if exp=1;` `if exp;`	All observations from **expenses1** (matches and nonmatches)
`if rev=0 or exp=0;` `if not rev or not exp;`	Nonmatches from **revenue1** and **expenses1**

148



Match-Merging (Review)

This program writes observations for matches only. The IN= data set option creates a variable that can be used to identify matches and non-matches.

```
data CustOrd;
   merge orion.customer(in=cust)
         work.order_fact(in=order);
   by Customer_ID;
   if cust=1 and order=1;
run;
```

```
NOTE: There were 77 observations read from the data set ORION.CUSTOMER.
NOTE: There were 128 observations read from the data set WORK.ORDER_FACT.
NOTE: The data set WORK.CUSTORD has 128 observations and 22 variables.
```

9

psm03d01

3.02 Short Answer Poll – Correct Answer

Write the appropriate IF statement to create the desired data set that contains only non-matches.

```
data combine;
   merge products(in=InProd) costs(in=InCost);
   by ID;
   if InProd=0 or InCost=0;
run;
```

Products

Product	ID
XYZ Shoe	A123
ABC Coat	B456

+

Costs

ID	Cost
B456	59.99
C789	35.75

=

Combine

Product	ID	Cost
XYZ Shoe	A123	
	C789	35.75

11

3. If you execute the following DATA step, which observations does the data set **bonuses** contain?

```
data bonuses;
   merge managers(in=M)
         staff(in=S);
   by EmpID;
   if M=0 and S=1;
run;
```

a. the observations from **staff** that have no match in **managers**
b. the observations from **managers** that have no match in **staff**
c. all observations from both **managers** and **staff**
d. none of the above

If M=0 and S=1, then only observations from the staff file that do not have a match in the managers file are selected.

50

5. The following program will execute without errors:

○ True
○ False

```
data work.merged;
   merge blood.donors1 blood.donors2;
   by ID;
run;
```

blood.donors1

ID	Type	Units
1104	O	16
1129	A	48
1127	A	50
1129	A	57
2486	B	63

blood.donors2

ID	Code	Rate
1104	65	27.00
1105	63	32.25
1129	62	39.15
2486	61	45.50
2525	64	67.00

The values of the BY variable must be sorted or indexed for a match-merge to execute successfully. The ID value 1127 in the donors1 file is out of sequence.

54

Multiple SET Statements with Duplicate Key Values

In this example, data set **one** has contiguous duplicates of the **CustomerID** value. Each has a match in data set **two**.

```
data three;
   set one;
   set two key=CustomerID;
run;
```

one

CustomerID	X
A	1
A	2
A	3

two

CustomerID	Y
A	1
A	2
A	3

One-to-one matches

70 p307d08

Multiple SET Statements with Duplicate Key Values

Data set **one** has contiguous duplicates. Some do not have a separately occurring match in data set **two**.

```
data three;
   set one;
   set two key=CustomerID;
run;
```

one

CustomerID	X
A	1
A	2
A	3

two

CustomerID	Y
A	1
A	2

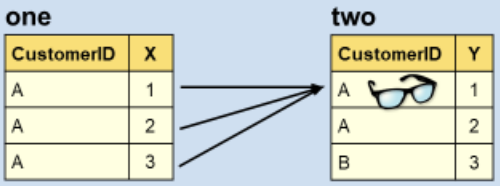
```
customerid=A x=3 y=2 _ERROR_=1 _IORC_=1230015 _N_=3
NOTE: There were 3 observations read from the data set WORK.ONE.
NOTE: The data set WORK.THREE has 3 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

71 p307d08

Multiple SET Statements with Duplicate Key Values

The UNIQUE option forces SAS to ignore duplicates.

```
data three;
   set one;
   set two key=CustomerID/unique;
run;
```



72

p307d08

Combining Data using the UPDATE Statement

You can update selected values of certain variables in a master file (e.g., MASTER) using a transaction file (e.g., TRANSECT below).

***Ex15_update_modify.sas;**

Master

Id	item	sub_6_inch	footlong
1	Cold Cut Combo	3.5	5.00
2	Pizza Sub	3.5	5.00
3	Spicy Italian	3.5	5.00
4	Veggie Delite	3.5	5.00
5	Turkey Breast	4.0	6.00
6	Tuna	4.0	6.00
7	Veggie Patty	4.0	6.00
8	Subway Club	4.5	7.00
9	Subway Melt	4.5	7.00
10	Steak & Cheese	4.5	7.25
11	Roast Beef	4.5	7.25

Transact

Id	item	sub_6_inch	footlong
9	Subway Melt	5.5	8.00
10	Steak & Cheese	5.5	9.25
11	Roast Beef	5.5	8.25

```
28 PROC SORT DATA=master;
29   BY id;
30 run;
31 PROC SORT DATA=Transact;
32   BY id;
33 run;
34 DATA updated;
35  UPDATE master Transact; BY id;
36 run;
37 PROC PRINT DATA=updated noobs;
38 run;
```

Id	item	sub_6_inch	footlong
1	Cold Cut Combo	3.5	5.00
2	Pizza Sub	3.5	5.00
3	Spicy Italian	3.5	5.00
4	Veggie Delite	3.5	5.00
5	Turkey Breast	4.0	6.00
6	Tuna	4.0	6.00
7	Veggie Patty	4.0	6.00
8	Subway Club	4.5	7.00
9	Subway Melt	5.5	8.00
10	Steak & Cheese	5.5	9.25
11	Roast Beef	5.5	8.25

Combining Data using the MODIFY Statement

You can modify selected values of certain variables in a master file (e.g., MASTER) using a transaction file (e.g., TRANSECT below).

***Ex15_update_modify.sas;**

Master

Id	item	sub_6_inch	footlong
1	Cold Cut Combo	3.5	5.00
2	Pizza Sub	3.5	5.00
3	Spicy Italian	3.5	5.00
4	Veggie Delite	3.5	5.00
5	Turkey Breast	4.0	6.00
6	Tuna	4.0	6.00
7	Veggie Patty	4.0	6.00
8	Subway Club	4.5	7.00
9	Subway Melt	4.5	7.00
10	Steak & Cheese	4.5	7.25
11	Roast Beef	4.5	7.25

Transact

Id	item	sub_6_inch	footlong
9	Subway Melt	5.5	8.00
10	Steak & Cheese	5.5	9.25
11	Roast Beef	5.5	8.25

```
40 DATA master_x;
41   SET master;
42 run;
43 DATA master_x;
44   MODIFY master_x Transact;
45 BY id;
46 run;
47 PROC PRINT DATA=Master_x noobs;
48 run;
```

Id	item	sub_6_inch	footlong
1	Cold Cut Combo	3.5	5.00
2	Pizza Sub	3.5	5.00
3	Spicy Italian	3.5	5.00
4	Veggie Delite	3.5	5.00
5	Turkey Breast	4.0	6.00
6	Tuna	4.0	6.00
7	Veggie Patty	4.0	6.00
8	Subway Club	4.5	7.00
9	Subway Melt	5.5	8.00
10	Steak & Cheese	5.5	9.25
11	Roast Beef	5.5	8.25

UPDATE replaces an existing file with a new file, allowing you to add, delete, or rename columns. MODIFY performs an update in place by rewriting only those records that have changed, or by appending new records to the end of the file.

Processing Missing Values with UPDATE and MERGE Statements

*Ex4_merge_update_missing.sas;

LISTNG - WORK.MASTER

Obs	pt_id	Name	visit_date
1	P001	Ann-Mary	07/23/2016

LISTNG - WORK.TRANS

Obs	pt_id	Name	visit_date
1	P001		09/24/2016

```
31 data work.Merged_NEW;
32   MERGE work.MASTER
33         work.TRANS; by pt_id;
34  run;
35  title 'WORK.MERGED_NEW';
36 proc print data=
37        work.Merged_NEW;
38  run;
```

WORK.MERGED_NEW

Obs	pt_id	Name	visit_date
1	P001		09/24/2016

```
40 data work.Updated_NEW;
41   UPDATE work.MASTER (IN=O)
42          work.TRANS (IN=T);
43          by pt_id;
44  run;
45  title 'WORK.Updated_NEW';
46 proc print data=
47   work.Updated_NEW;
48  run;
```

WORK.Updated_NEW

Obs	pt_id	Name	visit_date
1	P001	Ann-Mary	09/24/2016

Lines 31-34: When merging, SAS overwrites in the PDV with values from the rightmost data file specified in the statement.

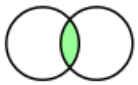
Lines 40:44: During an update, SAS uses the value for the NAME variable from the MASTER data set to replace the missing value for the same variable in the TRANS data set when it writes the observation in data set work.UPDATED_NEW.

Note that by default, UPDATE and MODIFY do not replace nonmissing values in a master data set with missing values from a transaction data set.

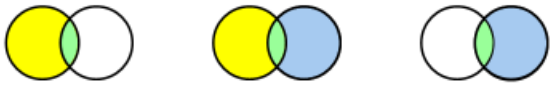
Types of Joins

PROC SQL supports two types of joins:

Inner joins return only matching rows.



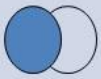
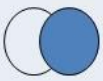
Outer joins return all matching rows, plus nonmatching rows from one or both tables.



Left Full Right

9

Merging -SQL vs. Data Step

VISUAL	SQL	DATA STEP
All rows from both tables 	**Full Outer Join** Select * from tableA Full outer Join tableB On tableA.id=tableB.id;	**Match Merge** Data tableC; Merge tableA tableB; By id; Run;
All rows from left table & matching rows from right table 	**Left Join** Select * from tableA Left Join tableB On tableA.id=tableB.id;	**Data Step Merge** Use IN=data set option Data tableD; Merge tableA(in=INA) tableB; By id; If INA; Run;
All rows from right table & matching rows from left table 	**Right Join** Select * from tableA Right Join tableB On tableA.id=tableB.id;	**Data Step Merge** Data tableE; Merge tableA tableB(in=INB); By id; If INB; Run;

VISUAL	SQL	DATA STEP
Matching rows only from both tables 	**Inner Join** Select * from tableA , INNER JOIN tableB On tableA.id=tableB.id: or Select * from tableA, tableB where tableA.id=tableB.id;	**Use IN= data set options** Data tableF; Merge tableA(in=INA) tableB(in=INB); By id; If INA and INB; Run;
All rows only in tableA but not in tableB 	Perform **left join**, then exclude unwanted records on right side Via a WHERE clause Select * from tableA LEFT JOIN tableB On tableA.id = tableB.id Where tableA.id ne tableB.id;	**Use IN= data set options** Data tableG; Merge tableA(in=INA) tableB(in=INB); By id; If INA and not INB; Run;
All rows unique to tableA and tableB 	Perform **full outer join**, then exclude records we don't want from both sides via a WHERE clause Select * from tableA FULL OUTER JOIN tableB ON tableA.id=tableB.id Where tableA.id ne tableB.id;	**Use IN= data set options** Data tableH; Merge tableA(in=INA) tableB(in=INB); By id; If not INA or not INB; Run;

Source: https://blogs.sas.com/content/sastraining/2015/05/27/life-saver-tip-for-comparing-proc-sql-join-with-sas-data-step-merge/

Blog written by Charu Shankar, 2015

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.

46 psm03d03

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

49

If you join tables that have multiple records per matching ids, your output table may be a Cartesian product. For example, with PROC SQL, 2 rows joining 5 rows of the same id variable will produce 10 rows. The DATA Step MERGE will create only 5 observations.

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.
```

50

Inner Join

Specify the matching criteria in the WHERE clause.

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

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

PROC SQL Output

ID	Name	ID	Action	Amount
102	Blank	102	Purchase	$100

30

s104d02

Abbreviating the Code with a Table Alias

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

PROC SQL Output

ID	Name	Action	Amount
102	Blank	Purchase	$100

39 s104d04

SQL Inner Join versus DATA Step Merge

A PROC SQL inner join and the DATA step match merge can return the same results.

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

```
data orders;
   merge customers(in=c)
         transactions(in=t);
   by ID;
   if c=1 and t=1;
run;
proc print data=orders;
run;
```

40 s104d04

SQL Inner Join versus DATA Step Merge

A PROC SQL inner join and the DATA step match merge will not always return the same results.

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

```
select *
   from customer2 as c2, transaction2 as t2
   where c2.ID=t2.ID;
```

ID	Name	ID	Action	Amount
102	Kent	102	Purchase	$376
102	Kent	102	Purchase	$376
102	Kent	102	Return	$119
102	Kent	102	Return	$119

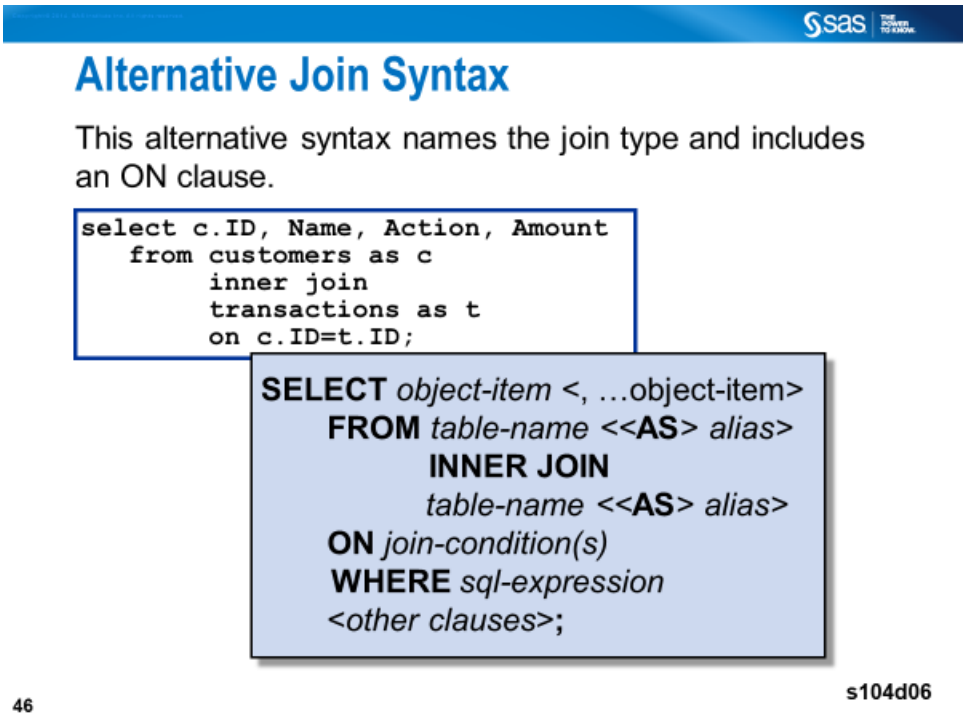
41

s104d05

Compare SQL Join and DATA Step Merge

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

45



Outer Joins

Outer join syntax is similar to the alternate inner join syntax.

```
proc sql;
title 'All Customers';
select *
   from customers as c
     left join
     transactions as t
   on c.ID=t.ID;
quit;
```

SELECT *object-item* <, …object-item>
    FROM *table-name* <<**AS**> *alias*>
        LEFT|RIGHT|FULL JOIN
        table-name <<**AS**> *alias*>
    ON *join-condition(s)*
    <*other clauses*>;

The ON clause specifies the join criteria in outer joins.

55 s104d07

Viewing the Output

PROC SQL Output

All Customers

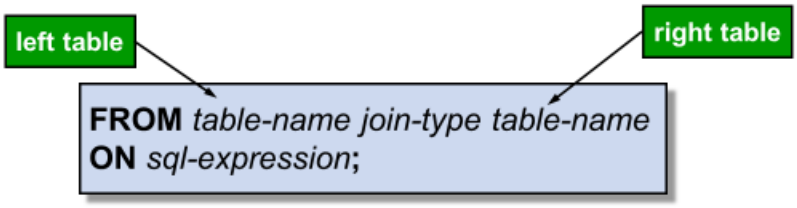
ID	Name	ID	Action	Amount
101	Smith	.		.
102	Blank	102	Purchase	$100
104	Jones	.		.

56

Determining Left and Right

Consider the position of the tables in the FROM clause.

- Left joins return all matching and non-matching rows from the ***left*** table and the matching rows from the ***right*** table.
- Right joins return all matching and non-matching rows from the ***right*** table and the matching rows from the ***left*** table.
- Full joins return all matching and non-matching rows from all of the tables.



57

Left Join

customers

ID	Name
101	Smith
104	Jones
102	Blank

transactions

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

```
select *
   from customers c left join transactions t
   on c.ID = t.ID;
```

ID	Name	ID	Action	Amount
101	Smith	.		.
102	Blank	102	Purchase	$100
104	Jones	.		.

Includes all rows from the left table, even if there are no matching rows in the right table.

s104d08

58

Right Join

customers

ID	Name
101	Smith
104	Jones
102	Blank

transactions

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

```
select *
   from customers c right join transactions t
   on c.ID = t.ID;
```

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

Includes all rows from the right table, even if there are no matching rows in the left table.

59 s104d09

Full Join

customers

ID	Name
101	Smith
104	Jones
102	Blank

transactions

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

```
select *
   from customers c full join transactions t
   on c.ID = t.ID;
```

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

Includes all rows from both tables, even if there are no matching rows in either table

60 s104d10

SQL Join versus DATA Step Merge

SQL joins do not overlay same-name columns.

customers

ID	Name
101	Smith
104	Jones
102	Blank

transactions

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

```
proc sql;
select *
   from customers c full join transactions t
   on c.ID=t.ID;
quit;
```

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

s104d13

COALESCE Function

You can use the *COALESCE function* to overlay columns. The COALESCE function returns the value of the first nonmissing argument.

```
proc sql;
select coalesce(c.ID,t.ID) as ID,
       Name, Action, Amount
   from customers c full join transactions t
   on c.ID=t.ID;
quit;
```

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

s104d13

Comparing Inner Joins and Outer Joins

Key Points	Inner Join	Outer Join
Table Limit	256	256
Join Behavior	Returns matching rows only	Returns matching and nonmatching rows
Join Options	Matching rows only	LEFT, FULL, RIGHT
Syntax changes	▪ Multiple tables, separated by commas, in the FROM clause ▪ WHERE clause that specifies join criteria	ON clause that specifies join criteria

77

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.

5

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

6

At initialization, SAS creates special Read-Only dictionary tables and views that contain data or metadata about each SAS session or batch job. You can access the tables using PROC SQL or you can use traditional SAS programming to access the view.

Querying Dictionary Information

Display information about the tables in the **orion** library.

```
title 'Tables in the ORION Library';
proc sql;
select memname 'Table Name',
       nobs,nvar,crdate
   from dictionary.tables
   where libname='ORION';
quit;
```

Library names are stored in uppercase in dictionary tables.

8 s108d01

Viewing the Output

Partial PROC SQL Output

Tables in the ORION Library

Table Name	Number of Physical Observations	Number of Variables	Date Created
CUSTOMER	77	12	14DEC07:08:05:44
CUSTOMER_TYPE	8	4	14DEC07:08:05:42
EMPLOYEE_ADDRESSES	424	9	23OCT07:10:12:23
EMPLOYEE_DONATIONS	124	7	14JAN08:10:34:21
EMPLOYEE_INFORMATION	424	11	05JUL12:11:25:08
EMPLOYEE_ORGANIZATION	424	4	12OCT07:14:06:32
EMPLOYEE_PAYROLL	424	8	14DEC07:08:08:06

9

Querying Dictionary Information

Display information about the columns in **orion.employee_addresses**.

```
title 'Columns in the orion.employee_addresses '
      'Table';
proc sql;
select Name,Type,Length
   from dictionary.columns
   where libname='ORION'
         and memname='EMPLOYEE_ADDRESSES';
quit;
```

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

10 s108d01

Viewing the Output

PROC SQL Output

Columns in the orion.employee_addresses Table

Column Name	Column Type	Column Length
Employee_ID	num	8
Employee_Name	char	40
Street_ID	num	8
Street_Number	num	8
Street_Name	char	40
City	char	30
State	char	2
Postal_Code	char	10
Country	char	2

Column names are stored in mixed case.

11

Using Dictionary Information

Which tables contain the **Employee_ID** column?

```
title 'Tables Containing an Employee_ID '
      'Column';
proc sql;
select memname 'Table Names', name
   from dictionary.columns
   where libname='ORION' and
         upcase(name)='EMPLOYEE_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.

12 s108d01

Viewing the Output

Partial PROC SQL Output

Tables Containing an Employee_ID Column

Table Names	Column Name
EMPLOYEE_ADDRESSES	Employee_ID
EMPLOYEE_DONATIONS	Employee_ID
EMPLOYEE_INFORMATION	Employee_ID
EMPLOYEE_ORGANIZATION	Employee_ID
EMPLOYEE_PAYROLL	Employee_ID
EMPLOYEE_PHONES	Employee_ID
ORDER_FACT	Employee_ID
SALESSTAFF	Employee_ID
STAFF	Employee_ID

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

13

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 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 ORION Library';
proc print data=sashelp.vtable label;
   var memname nobs nvar;
   where libname='ORION';
run;
```

16 s108d02

Review Questions

1. Suppose you have listed a variable named ID in the BY statement along with the SET statement in a DATA step, what variables are created?
2. What is the purpose of using the IN= option in the MERGE statement in a DATA step?
3. You have merged two SAS data sets. From which data set (left or right), common non-BY variable values will be kept in the output data set?
4. You have concatenated two SAS data sets. From which data set (left or right), common non-BY variable values will be kept in the output data set?
5. Imagine that you are merging two data sets based on a selected variable. Which SQL join will produce the same results as "match-merging" when only some of the values of that variable match?
6. How would you eliminate the rows containing duplicate values in PROC SQL?
7. Explain the significance of the following operators in PROC SQL.
    - CONTAINS
    - IN
    - BETWEEN-AND
    - IS MISSING
    - IS NULL
    - Wild card operator underscore (_) with LIKE operator
    - Wild card operator percent (%) with LIKE operator
    - SOUNDS-LIKE operator
8. In PROC SQL, how is the calculation performed when the summary function (e.g., AVG or COUNT) with one argument is used?
9. In PROC SQL, how is the calculation performed when the summary function (e.g., SUM) with multiple arguments is used?
10. Write PRCO SQL codes to calculate the
    - Total number of rows in a data table or group
    - Number of non-missing values for a specific column
    - Number of unique values for a specific column
11. Use SASHELP.CLASS and write a single-value subquery (i.e., inner query) in PROC SQL such that the outer query lists students by SEX whose weights are greater than the average weight of all students.