Stacking Up - Horizontal or Vertical with PROC SQL or DATA Step

Wisconsin Illinois SAS Users Group

Milwaukee

28 June 2017

Charu Shankar

Technical Training Specialist

SAS Institute Canada Inc.

THE POWER TO KNOW.

Agenda

1. Vertical Stacking
   - Data & Proc Steps
   - PROC SQL Set Operators
   - Compare and contrast?
2. Horizontal Stacking
   - Data Step Merge
   - PROC SQL Joins
   - Compare and contrast?
3. Questions
4. Useful Links
5. Contact

About your instructor

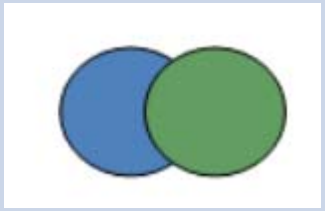
Charu Shankar trains and develops curriculum in SAS data access, analysis, reporting and big data solutions.

For over 20 years Charu has delivered computer language training to SAS customers globally and at the United Nations field office in New Delhi.

Skilled in customer needs analysis, Charu recommends the right SAS training to customers.

Work blog http://blogs.sas.com/content/author/charushankar/
My yoga & cooking www.charuyoga.com

Wide or long? Broad or narrow? How do you want to go?

Visual	Stacking	PROC SQL	Data Step	PROC	PROC
	Horizontal Stacking stack columns and align rows.	Joins	Merge		
	Vertical stacking rows and align columns.	Set operators	Concatenate	Append	Datasets

Now some language

	Data Step	PROC SQL
Vertical Stacking	*Concatenate*	*Set Operations*
	Data Step Concatenate	Union
	Proc Append	Outer Union
		Except
		Intersect
Horizontal Stacking	*Merge*	*Joins*
	Match Merge	Inner Join
	Non Matches	Outer Join
		Left Join
		Right Join
		Full Join

1. Vertical Stacking

1. Vertical Stacking - Proc Append Concepts

Business Scenario

emps is a master data set that contains employees hired in 2006 and 2007.

emps

First	Gender	HireYear
Stacey	F	2006
Gloria	F	2007
James	M	2007

The employees hired in 2008, 2009, and 2010 need to be appended.

emps2008

First	Gender	HireYear
Brett	M	2008
Renee	F	2008

emps2009

First	HireYear
Sara	2009
Dennis	2009

emps2010

First	HireYear	Country
Rose	2010	Spain
Eric	2010	Spain

1. Vertical Stacking - Proc Append Syntax

```
PROC APPEND BASE=SAS-data-set
            DATA=SAS-data-set;
RUN;
```

Like-Structured Data Sets

emps

First	Gender	HireYear
Stacey	F	2006
Gloria	F	2007
James	M	2007

emps2008

First	Gender	HireYear
Brett	M	2008
Renee	F	2008

The data sets contain the same variables.

```
proc append base=emps
            data=emps2008;
run;
```

1. Vertical Stacking - Proc Append Output

Like-Structured Data Sets

```
154  proc append base=emps data=emps2008;
155  run;

NOTE: Appending WORK.EMPS2008 to WORK.EMPS.
NOTE: There were 2 observations read from the data set WORK.EMPS2008.
NOTE: 2 observations added.
NOTE: The data set WORK.EMPS has 5 observations and 3 variables.
NOTE: PROCEDURE APPEND used (Total process time):
      real time               0.03 seconds
      cpu time                0.01 seconds
```

emps

First	Gender	HireYear
Stacey	F	2006
Gloria	F	2007
James	M	2007
Brett	M	2008
Renee	F	2008

1. Vertical Stacking - Proc Datasets Syntax

PROC DATASETS LIBRARY= *libref*;
APPEND BASE = *SAS-data-set* **DATA** =*SAS-data-set;*
RUN;

Like-Structured Data Sets

emps

First	Gender	HireYear
Stacey	F	2006
Gloria	F	2007
James	M	2007

emps2008

First	Gender	HireYear
Brett	M	2008
Renee	F	2008

The data sets contain the same variables.

```
Proc datasets library=work;
Append base=emps data=emps2008;
Quit;
```

1. Vertical Stacking - Proc Datasets Output

Like-Structured Data Sets

```
131   proc datasets library=work ;
132   append base=emps data=emps2008;
133   quit;

NOTE: Appending WORK.EMPS2008 to WORK.EMPS.
NOTE: There were 2 observations read from the data set WORK.EMPS2008.
NOTE: 2 observations added.
NOTE: The data set WORK.EMPS has 5 observations and 3 variables.
NOTE: PROCEDURE DATASETS used (Total process time):
      real time           0.07 seconds
      cpu time            0.04 seconds
```

emps

First	Gender	HireYear
Stacey	F	2006
Gloria	F	2007
James	M	2007
Brett	M	2008
Renee	F	2008

1. Vertical Stacking - Proc Append /Proc Datasets unique cases

Unlike-Structured Data Sets

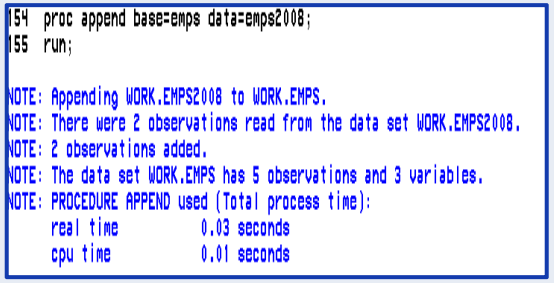
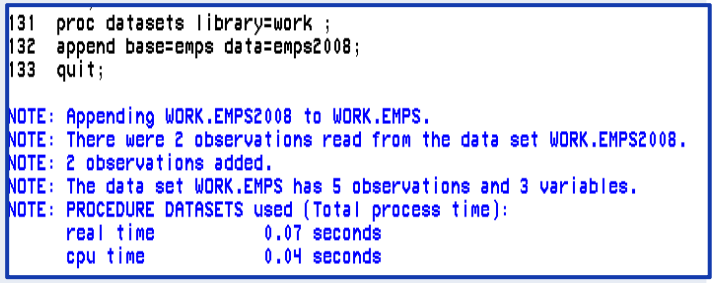
Situation	Action
The BASE= data set contains a variable that is not in the DATA= data set.	The observations are appended, but the observations from the DATA= data set have a missing value for the variable that was not present in the DATA= data set. The FORCE option is not necessary in this case.
The DATA= data set contains a variable that is not in the BASE= data set.	Use the FORCE option in the PROC APPEND statement to force the concatenation of the two data sets. The statement drops the extra variable and issues a warning message.

1. Vertical Stacking - Proc Append Tips

- Ensure that both datasets have the same exact variables and variable attributes to avoid a 'messy' log (i.e. avoid using the FORCE option if at all possible).
- Explicitly define the dataset name using the 'DATA=' option in the PROC APPEND statement. If 'DATA=xxxx' is not specified, SAS uses the dataset most recently created.
- Use Proc Append with WORK datasets. Avoid using with permanent datasets.
- Designate the larger of the two datasets as the BASE dataset to improve processing efficiency

1. Vertical Stacking

Proc Append vs. Proc Datasets

WHAT	Proc Append	Proc Datasets
RUN group processing	No	Supports Run-group processing-to submit multiple run groups within the same procedure without ending the procedure. Ends with a quit.
Default library	Work or User	None
Data Management	No	Yes
Data Building	No for both. Use FORCE option if the data= dataset has a different structure from the Base= dataset	
Processing time	Behind the scenes same code is used so little performance gain should be expected.	
		

```
154  proc append base=emps data=emps2008;
155  run;

NOTE: Appending WORK.EMPS2008 to WORK.EMPS.
NOTE: There were 2 observations read from the data set WORK.EMPS2008.
NOTE: 2 observations added.
NOTE: The data set WORK.EMPS has 5 observations and 3 variables.
NOTE: PROCEDURE APPEND used (Total process time):
      real time          0.03 seconds
      cpu time           0.01 seconds
```

```
131  proc datasets library=work ;
132  append base=emps data=emps2008;
133  quit;

NOTE: Appending WORK.EMPS2008 to WORK.EMPS.
NOTE: There were 2 observations read from the data set WORK.EMPS2008.
NOTE: 2 observations added.
NOTE: The data set WORK.EMPS has 5 observations and 3 variables.
NOTE: PROCEDURE DATASETS used (Total process time):
      real time          0.07 seconds
      cpu time           0.04 seconds
```

1. Vertical Stacking - Data Step Concepts

Concatenate like-structured data sets **empsdk** and **empsfr** to create a new data set named **empsall1**.

empsdk

First	Gender	Country
Lars	M	Denmark
Kari	F	Denmark
Jonas	M	Denmark

empsfr

First	Gender	Country
Pierre	M	France
Sophie	F	France

empsall1

First	Gender	Country
Lars	M	Denmark
Kari	F	Denmark
Jonas	M	Denmark
Pierre	M	France
Sophie	F	France

Both data sets contain the same variables.

1. Vertical Stacking - Data Step Syntax

Execution

empsdk

First	Gender	Country
Lars	M	Denmark
Kari	F	Denmark
Jonas	M	Denmark

empsfr

First	Gender	Country
Pierre	M	France
Sophie	F	France

EOF

```
data empsall1;
   set empsdk empsfr;
run;
```

PDV

First	Gender	Country
Sophie	F	France

empsall1

First	Gender	Country
Lars	M	Denmark
Kari	F	Denmark
Jonas	M	Denmark
Pierre	M	France
Sophie	F	France

1. Vertical Stacking - Data Step Output

Viewing the Log

Partial SAS Log

```
145  data empsall1;
146     set empsdk empsfr;
147  run;

NOTE: There were 3 observations read from the data set
WORK.EMPSDK.
NOTE: There were 2 observations read from the data set
WORK.EMPSFR.
NOTE: The data set WORK.EMPSALL1 has 5 observations and 3
variables.
```

1. Vertical Stacking - Data Step Advantage

Scenario 1 Standard data step concatenate	Scenario 2. Build new variables	Scenario 3 Determine which table contributed to final results
Create new table by reading every row in every table listed on the set statement	Build new variables in the new dataset	The data step can keep all uncommon columns, plus determine which table contributed to final results.
data dataconc; **set** ** wiilsu.prepsales** ** wiilsu.nonsales;** **run;**	**data dataconc;** **set** ** wiilsu.prepsales** ** wiilsu.nonsales;** ** sales='YS';** ** nonsales='YN';** **run;**	**data dataconc;** **set** ** wiilsu.prepsales(in=ins)** ** wiilsu.nonsales(in=inn);** ** sales=ins;** ** nonsales=inn;** **run;**

1. Vertical Stacking

Proc Append/Proc Datasets vs Data step

ACTION	PROC APPEND	Data Step Concatenate
Handing of different variables in datasets	cannot include variables found only in the DATA= dataset	Uses all variables and assigns missing values as necessary
Number of data sets	2	Any number
Speed	Proc Append much faster since it doesn't process observations from BASE= data set	Slower- has to read every data set listed on SET statement
Build new variables or Create new Table	Cannot build new variables as descriptor portion information in base SAS dataset cannot change	almost always the best method for creating a table from several input datasets or building new variables in the same data step
Space	Proc Append only reads in observations from dataset being appended. Use Proc Append over the SET statement to save work space.	SET statement reads in all observations from the datasets being concatenated.

1. Vertical Stacking - PROC SQL Concepts

Set Operator

PROC SQL provides traditional set operators from relational algebra:

OUTER UNION
    concatenates the query results. Similar to data step concatenate
UNION
    produces all unique rows from both queries.
EXCEPT
    produces rows that are part of the first query only.
INTERSECT
    produces rows that are common to both query results.

1. Vertical Stacking - PROC SQL Concepts

Set Operator

Data is stored in 2 tables.

Which employees have completed training A or B?

Partial **train_a**

ID	Name	Date
11	Bob	15JUN2012
16	Sam	5JUN2012
14	Pete	21JUN2012

Training class A is completed in a single session. Date represents the date of training.

Partial **train_b**

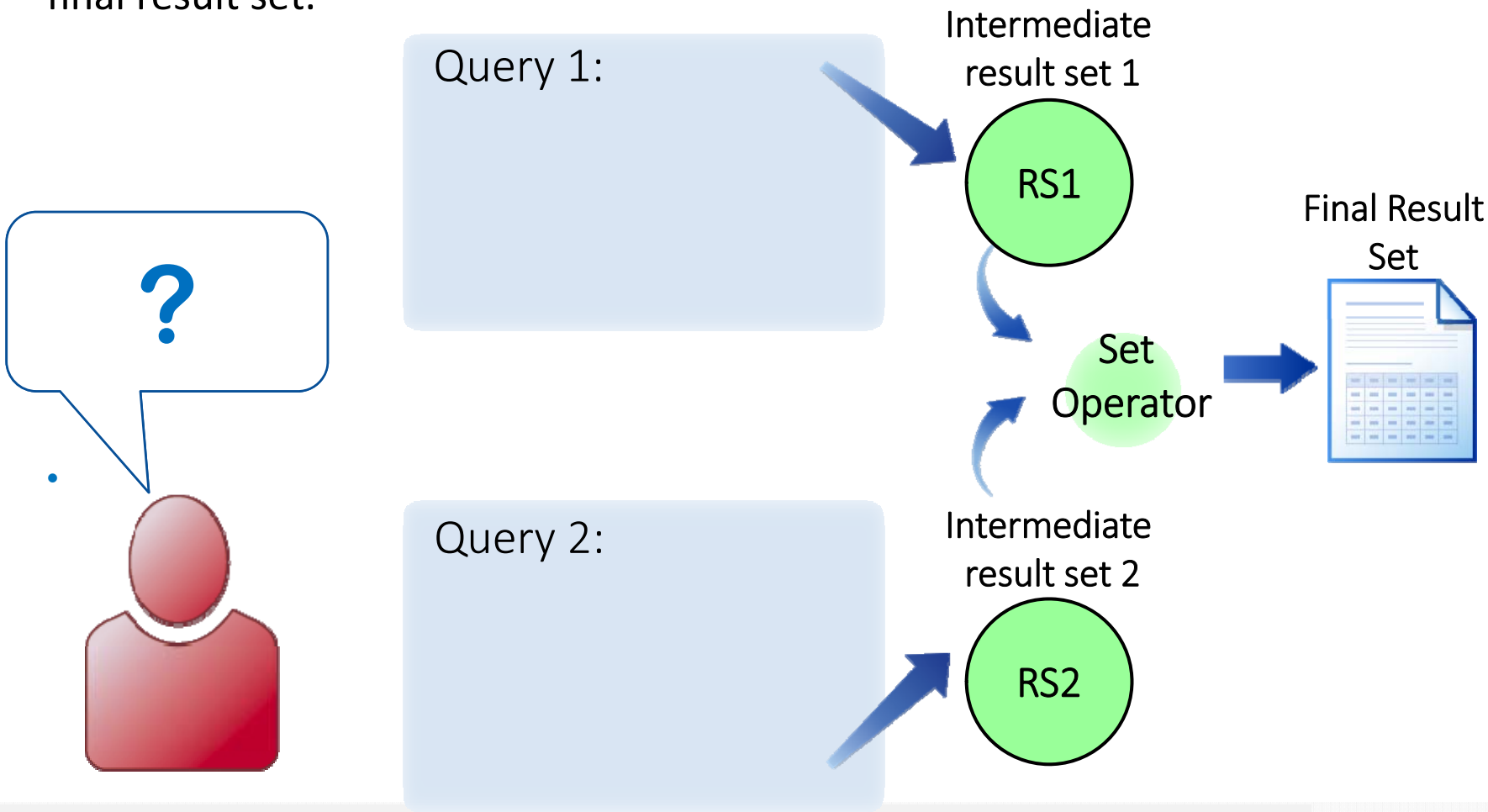
Name	ID	SDate	EDate
Bob	11	9JUL2012	13JUL2012
Pam	15	25JUL2012	27JUL2012
Kyle	19	12JUL2012	20JUL2012
Chris	21	29JUL2012	.

Training class B is a multi-session class. SDate is recorded on the first training day. EDate is recorded when the course is complete.

1. Vertical Stacking - PROC SQL Concepts

Set Operator

Set operators use the intermediate result sets from two queries to create a final result set.



1. Vertical Stacking - PROC SQL Syntax

Set Operator general form

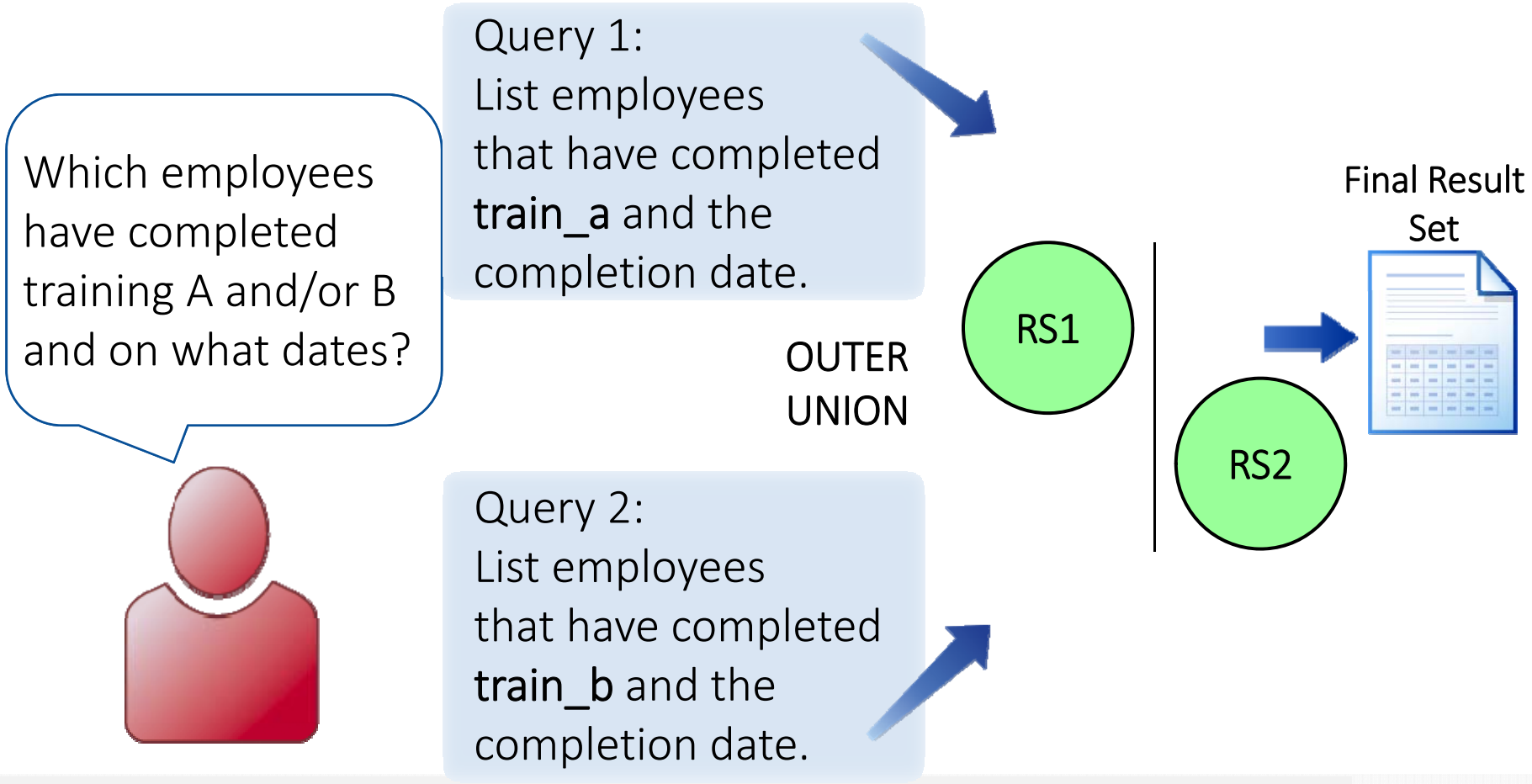
SELECT …
UNION | OUTER UNION | EXCEPT | **INTERSECT**
<ALL><CORR>
SELECT …**;**

The modifiers ALL and CORR change the default behavior of the set operators.

- ALL modifies the default behavior for rows.
- CORR modifies the default behavior for columns.

1. Vertical Stacking - PROC SQL Concepts

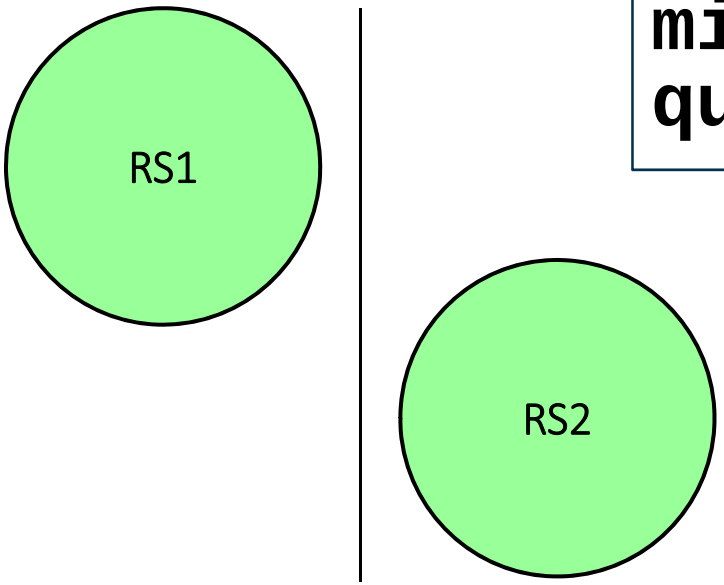
Outer Union Set Operator



1. Vertical Stacking - PROC SQL Syntax

Outer Union Set Operator

keep all rows and all columns from the two intermediate result sets with The OUTER UNION operator




```
proc sql;
select * from train_a
outer union
select * from train_b
   where EDate is not
missing;
quit;
```

SELECT …
OUTER UNION <*CORR*>
SELECT …

s106d04

1. Vertical Stacking - PROC SQL Output

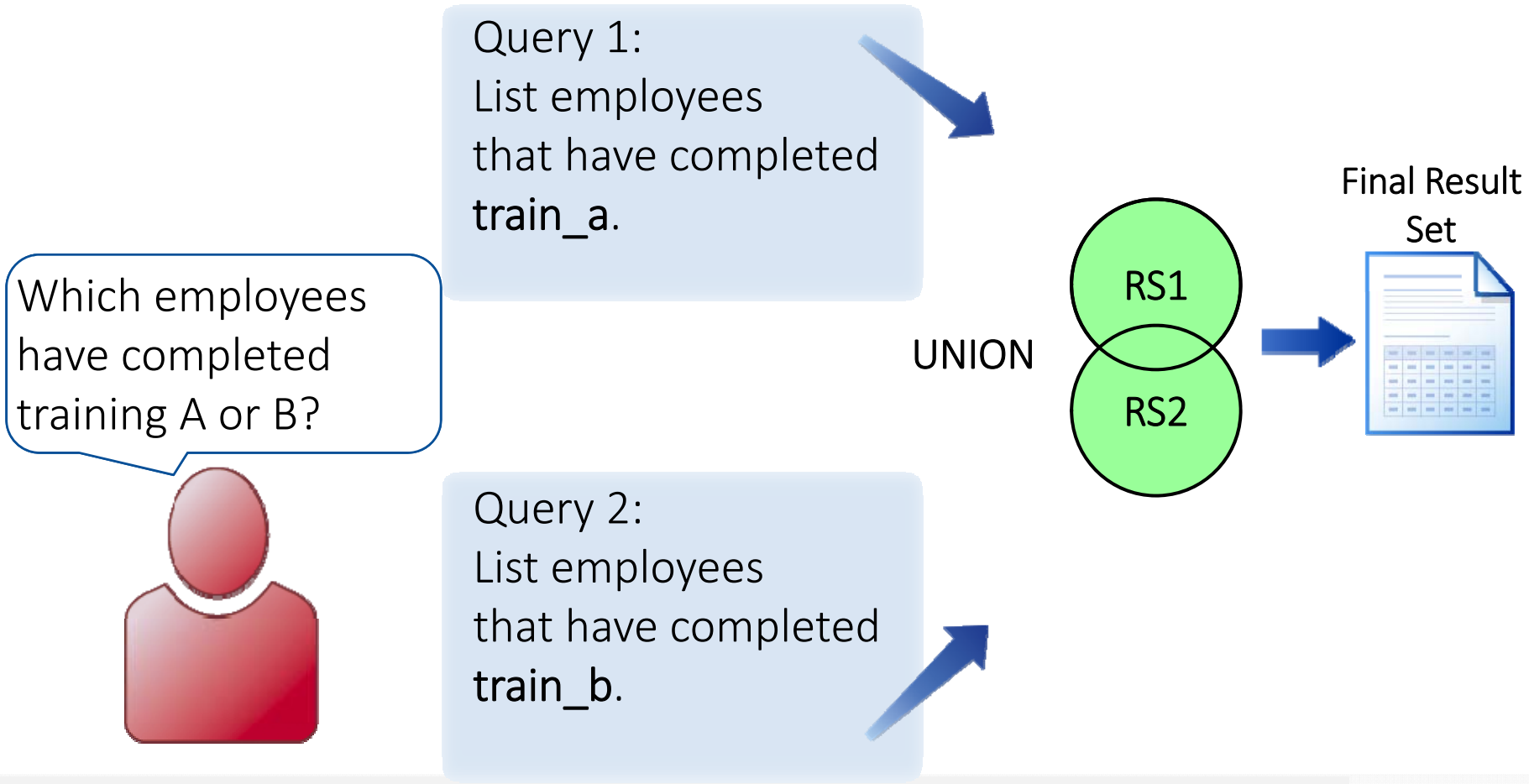
Outer Union Set Operator

Who completed Training A and /or B and on what dates?

ID	Name	Completion Date	Name	ID	Start Date	End Date
11	Bob	15JUN2012		.	.	.
16	Sam	05JUN2012		.	.	.
14	Pete	21JUN2012		.	.	.
21	Chris	07JUN2012		.	.	.
18	Kim	04JUN2012		.	.	.
17	Pat	22JUN2012		.	.	.
20	Mary	11JUN2012		.	.	.
12	Sue	06JUN2012		.	.	.
87	Ted	05JUN2012		.	.	.
91	Rand	07JUN2012		.	.	.
.		.	Bob	11	09JUL2012	13JUL2012
.		.	Pam	15	25JUL2012	27JUL2012
.		.	Kyle	19	12JUL2012	20JUL2012
.		.	Ted	87	09JUL2012	13JUL2012

1. Vertical Stacking - PROC SQL Concepts

Union Set Operator



1. Vertical Stacking - PROC SQL Syntax

Union Set Operator

The **UNION** operator in Proc SQL is used to append rows of two or more SELECT statements having the same number of columns with similar data types.

```
proc sql;
    select ID, Name from
    work.train_a
    union
    select ID, Name from
    work.train_b
    where EDate is not missing;
quit;
```

SELECT …
UNION
<ALL><CORR>
SELECT …;

s106d01

1. Vertical Stacking - PROC SQL Output

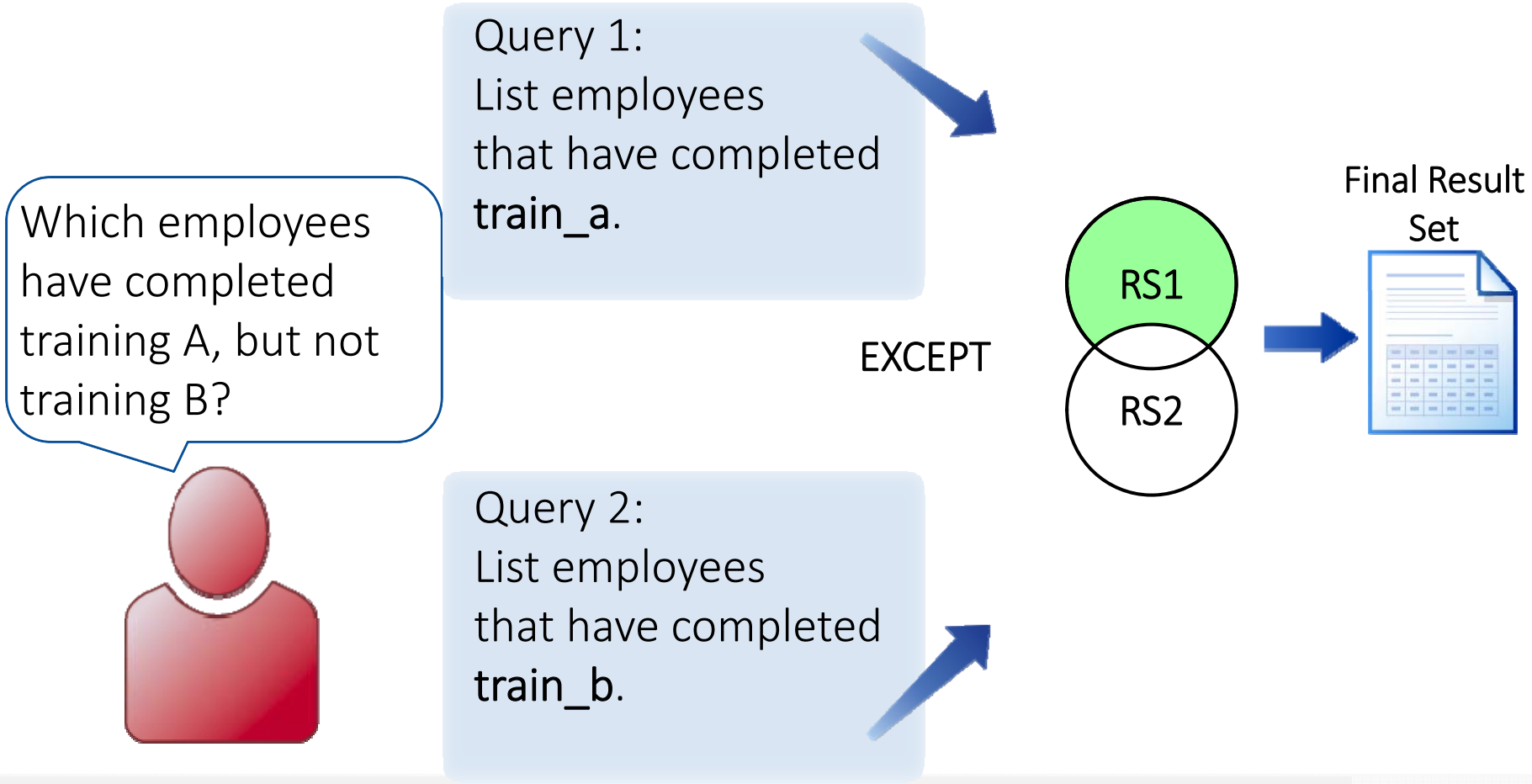
Union Set Operation

Which Employees Have Completed Training A or B?

ID	Name
11	Bob
12	Sue
14	Pete
15	Pam
16	Sam
17	Pat
18	Kim
19	Kyle
20	Mary
21	Chris
87	Ted
91	Rand

1. Vertical Stacking - PROC SQL Concepts

Except Set Operator



Vertical Stacking - PROC SQL Syntax

Except Set Operator

List employees who have completed training A, but not training B with the EXCEPT operator.

```
proc sql;
select ID, Name from train_a
except
select ID, Name from train_b
   where Edate is not missing;
quit;
```

SELECT …
EXCEPT *<ALL><CORR>*
SELECT …

s106d07

1. Vertical Stacking - PROC SQL Output

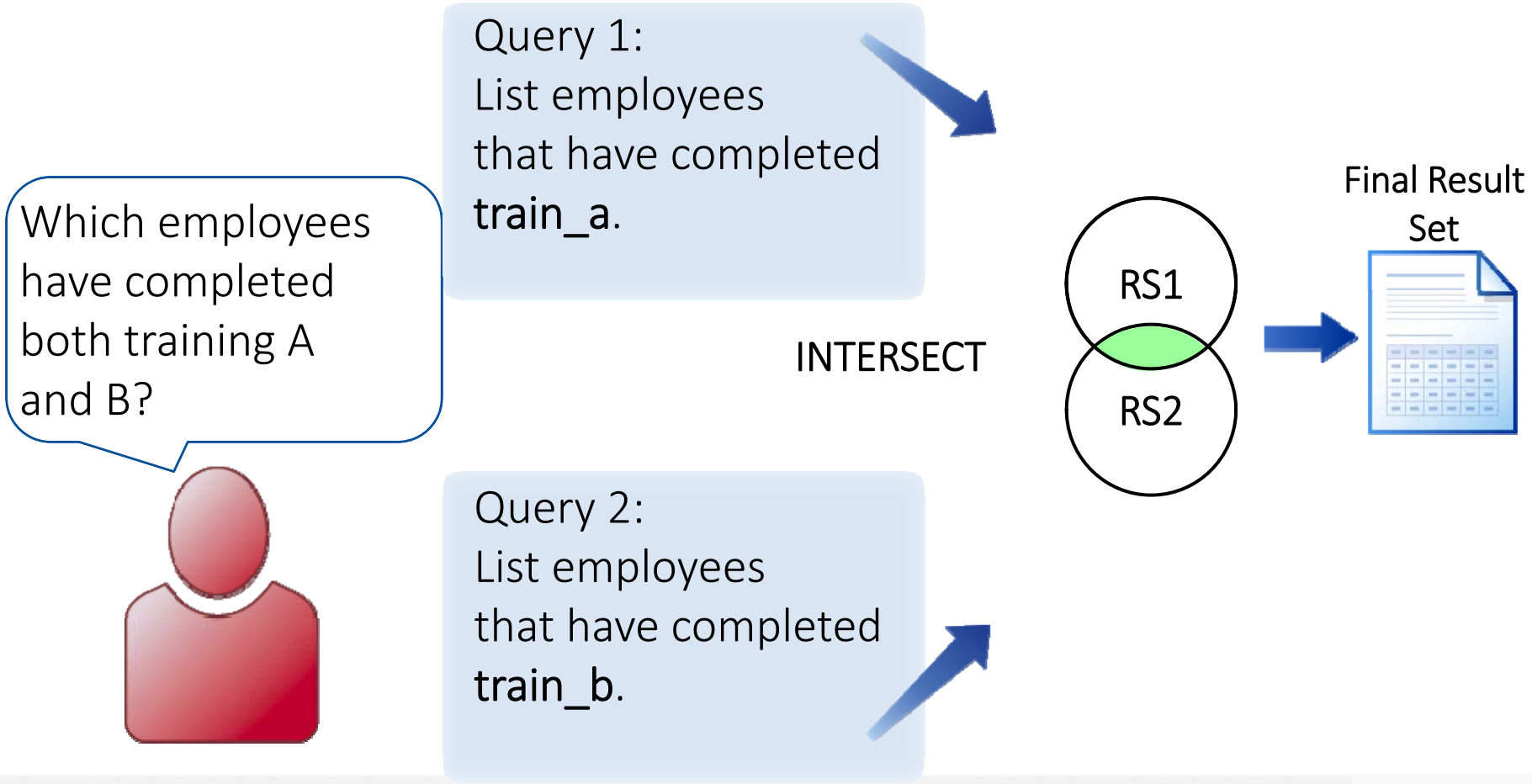
Except Set Operator

Which Employees Have Completed Training A, But Not Training B

ID	Name
12	Sue
14	Pete
16	Sam
17	Pat
18	Kim
20	Mary
21	Chris
91	Rand

1. Vertical Stacking - PROC SQL Concepts

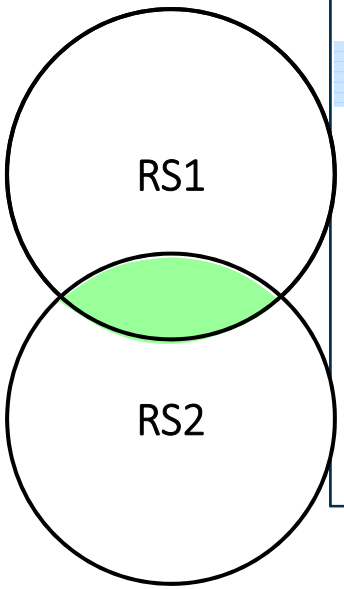
Intersect Set Operator



1. Vertical Stacking - PROC SQL Syntax

Intersect Set Operator

Rows that exist in both **train_a** and **train_b**. The INTERSECT operator will accomplish this.




```
proc sql;
select ID, Name from train_a
intersect
select ID, Name from train_b
   where EDate is not
missing;
quit;
```

SELECT …
INTERSECT <ALL><CORR>
SELECT …

s106d09

1. Vertical Stacking - PROC SQL Output

Except Set Operator

Employees Who Have Completed Both Training Classes

ID	Name
11	Bob
87	Ted

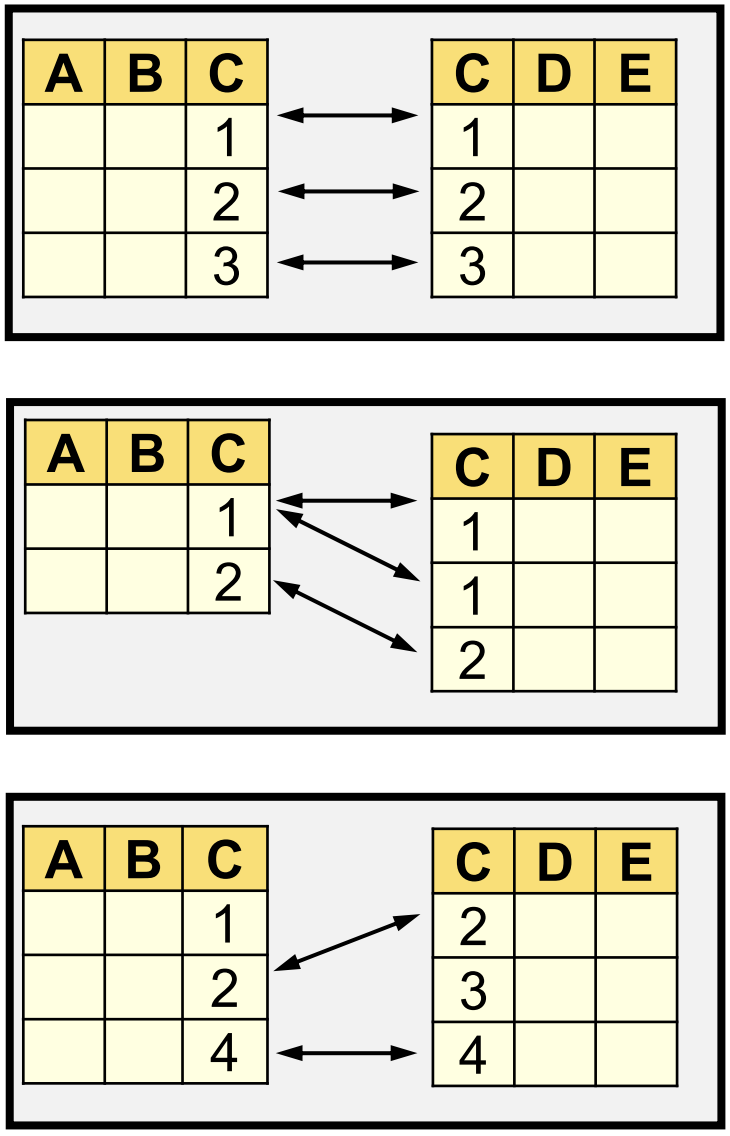
1. Vertical Stacking - Proc Sql or Data step?

Comparison	Proc SQL	Data Step
Length	Less lengthy code	
Speed	May execute faster for smaller tables	
Portability	More portable for Non-SAS programmers & Non-SAS Applications	
Transparency		Data step explicitly prints messages in log
Manipulation		Data step arrays, hash objects
Sorting	No explicit code required for sorting	
Same named variables	Not required, although same type and length are required.	
Caution	Builds Cartesian products if JOIN criteria not explicitly listed.	
Reading non DBMS data	Cannot read	Data step strength. Ability to read variety of input data sources including non-RDBMS

2. Horizontal Stacking

2. Horizontal Stacking - Data step Merge

Concepts

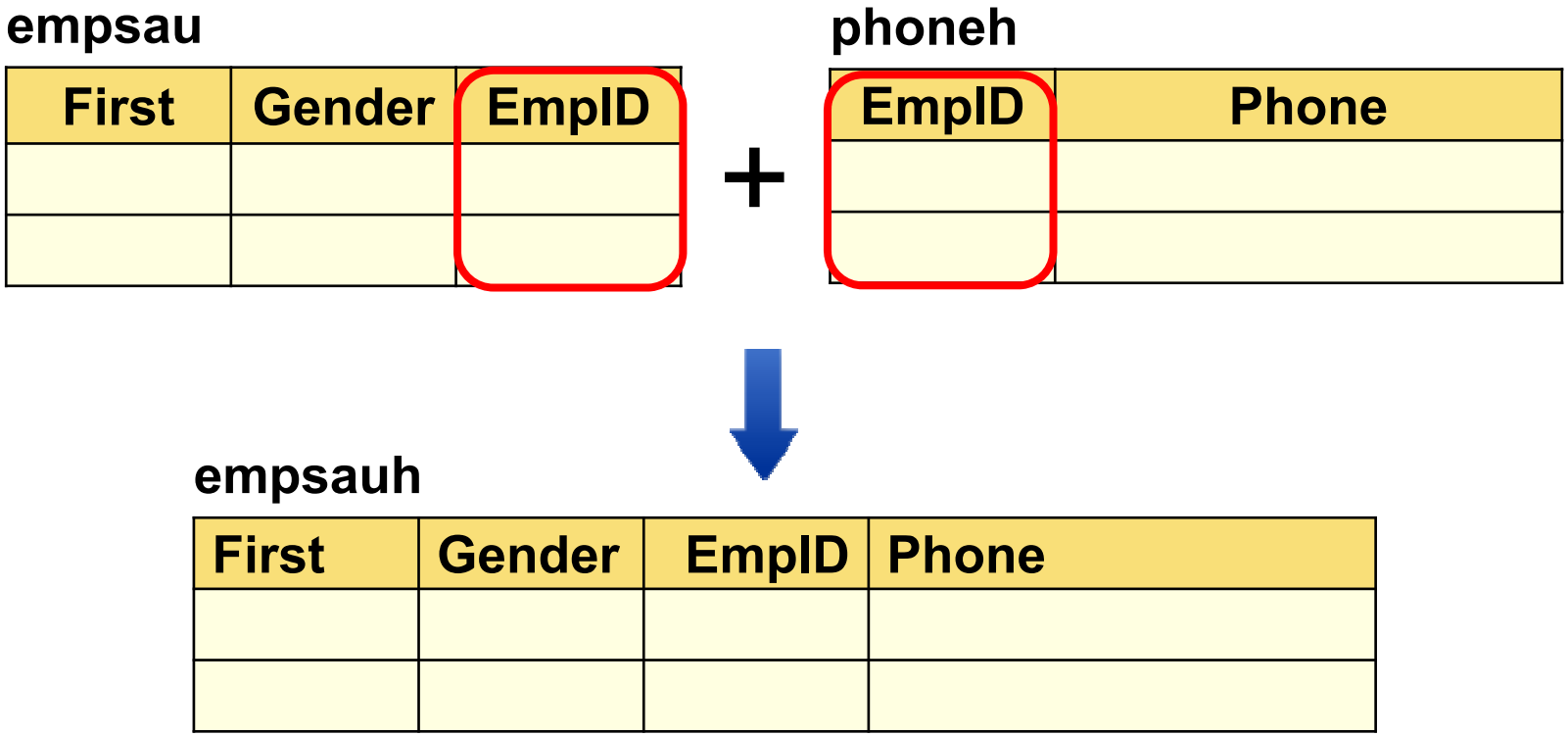


Match-Merging

2. Horizontal Stacking - Data step Concepts

One-to-one Merge

Merge the Australian employee data set with a phone data set to obtain each employee's home phone number, storing the results in a new data set.



2. Horizontal Stacking - Data step Syntax

One-to-one Merge

```
data empsauh;
    merge empsau phoneh;
    by EmpID;
run;
```

MERGE *SAS-data-set1 SAS-data-set2 . . .*;
BY <DESCENDING> *BY-variable(s)*;

The data sets are sorted by **EmpID**.

2. Horizontal Stacking - Data step Output

One-to-one Merge

empsau

First	Gender	EmpID
Togar	M	121150
Kylie	F	121151
Birin	M	121152

phoneh

EmpID	Phone
121150	+61(2)5555-1793
121151	+61(2)5555-1849
121152	+61(2)5555-1665

empsauh

First	Gender	EmpID	Phone
Togar	M	121150	+61(2)5555-1793
Kylie	F	121151	+61(2)5555-1849
Birin	M	121152	+61(2)5555-1665

2. Horizontal Stacking - Data Step

Matches & Non Matches

IN= Data Set Option

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

MERGE *SAS-data-set* **(IN=***variable***) …**

variable is a temporary numeric variable that has two possible values:

0	Indicates that the data set did ***not*** contribute to the current observation.
1	Indicates that the data set ***did*** contribute to the current observation.

2. Horizontal Stacking - Data Step Merge

Matches & Non Matches

empsau

First	Gender	EmpID
Togar	M	121150
Kylie	F	121151
Birin	M	121152

phonec

EmpID	Phone
121150	+61(2)5555-1795
121152	+61(2)5555-1667
121153	+61(2)5555-1348

Execution

```
data empsauc;
   merge empsau(in=Emps)
         phonec(in=Cell);
   by EmpID;
run;
```

match

p110d08

PDV

First	Gender	EmpID	D Emps	Phone	D Cell
Togar	M	121150	1	+61(2)5555-1795	1

2. Horizontal Stacking - Data Step Merge

Isolating Matches & Non Matches

empsau

First	Gender	EmpID
Togar	M	121150
Kylie	F	121151
Birin	M	121152

phonec

EmpID	Phone
121150	+61(2)5555-1795
121152	+61(2)5555-1667
121153	+61(2)5555-1348

Execution

```
data empsauc;
   merge empsau(in=Emps)
         phonec(in=Cell);
   by EmpID;
run;
```

non-match

PDV

First	Gender	EmpID	D Emps	Phone	D Cell
Kylie	F	121151	1		0

2. Horizontal Stacking - Data Step Merge

Isolating Matches & Non Matches

empsau

First	Gender	EmpID
Togar	M	121150
Kylie	F	121151
Birin	M	121152

phonec

EmpID	Phone
121150	+61(2)5555-1795
121152	+61(2)5555-1667
121153	+61(2)5555-1348

Execution

```
data empsauc;
   merge empsau(in=Emps)
         phonec(in=Cell);
   by EmpID;
run;
```

match

PDV

First	Gender	EmpID	D Emps	Phone	D Cell
Birin	M	121152	1	+61(2)5555-1667	1

2. Horizontal Stacking - Data Step Merge

Matches & Non Matches

PDV

First	Gender	EmpID	D Emps	Phone	D Cell
Togar	M	121150	1	+61(2)5555-1795	1
Kylie	F	121151	1		0
Birin	M	121152	1	+61(2)5555-1667	1
		121153	0	+61(2)5555-1348	1

The variables created with the IN= data set option are only available during DATA step execution.

- They are not written to the SAS data set.
- Their value can be tested using conditional logic.

2. Horizontal Stacking - Data Step Merge

Isolating Matches only

Add a subsetting IF statement to select the employees that have company phones.

```
data empsauc;
   merge empsau(in=Emps)
         phonec(in=Cell);
   by EmpID;
   if Emps=1 and Cell=1;
run;
```

empsauc

First	Gender	EmpID	Phone
Togar	M	121150	+61(2)5555-1795
Birin	M	121152	+61(2)5555-1667

p110d08

2. Horizontal Stacking - Data Step Merge

Isolating Non Matches from empsau only

Select employees that do not have company phones.

```
data empsauc;
   merge empsau(in=Emps)
         phonec(in=Cell);
   by EmpID;
   if Emps=1 and Cell=0;
run;
```

empsauc

First	Gender	EmpID	Phone
Kylie	F	121151	

p110d08

2. Horizontal Stacking - Data Step Merge

Isolating Non Matches from phoneC only

Select phones associated with an invalid employee ID.

```
data empsauc;
   merge empsau(in=Emps)
         phonec(in=Cell);
   by EmpID;
   if Emps=0 and Cell=1;
run;
```

empsauc

First	Gender	EmpID	Phone
		121153	+61(2)5555-1348

p110d08

2. Horizontal Stacking - Data Step Merge

Isolating all Non-matches

```
data empsauc;
   merge empsau(in=Emps)
         phonec(in=Cell);
   by EmpID;
   if Emps=0 or Cell=0;
run;
```

empsauc

First	Gender	EmpID	Phone
Kylie	F	121151	
		121153	+61(2)5555-1348

Use the OR operator, not the AND operator.

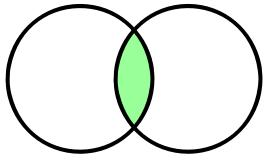
p110d08

2. Horizontal Stacking - Proc SQL Joins

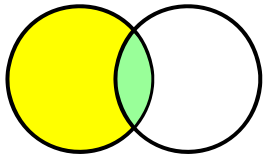
Types of Joins

PROC SQL supports two types of joins:

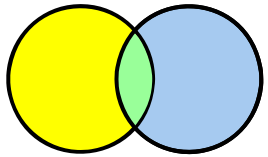
- *Inner joins* return only matching rows.



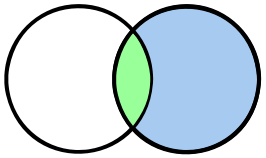
- *Outer joins* return all matching rows, plus nonmatching rows from one or both tables.



Left



Full



Right

2. Horizontal Stacking - Proc SQL Joins

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

Horizontal Stacking - Proc SQL Joins

Building 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
		3	Return	$52
		5	Return	$212
		2	Purchase	$100
		3	Return	$52
		5	Return	$212

9 rows

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

Horizontal Stacking - Proc SQL Joins

Cartesian product features

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

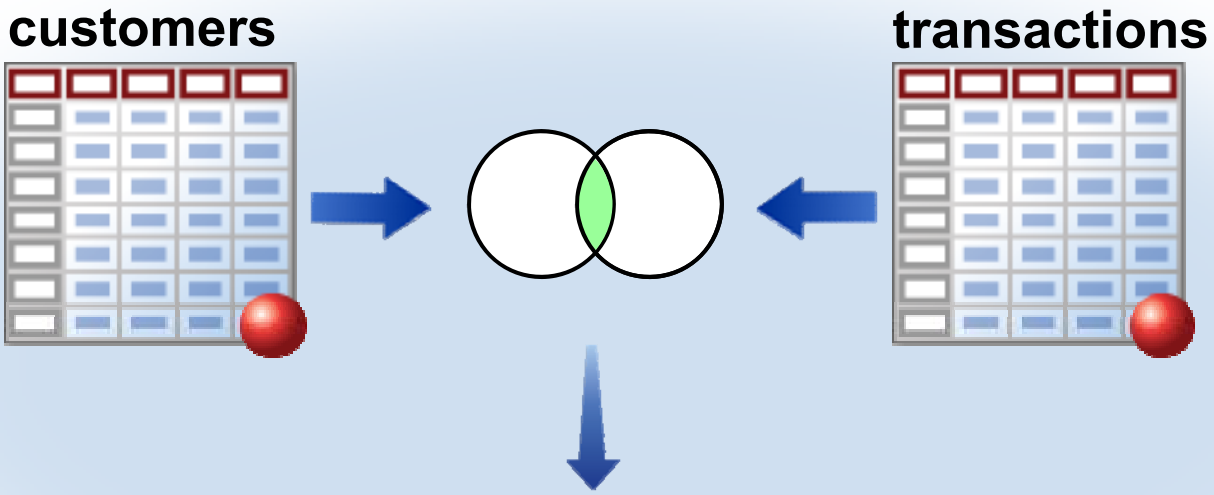
Partial SAS Log

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

2. Horizontal Stacking - Proc SQL Joins

Inner Join

report showing all valid order information.



ID	Name	Action	Amount
102	Blank	Purchase	$100

Horizontal Stacking – Proc SQL Inner Join

Syntax

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

s104d02

2. Horizontal Stacking - Proc SQL Inner Join vs DataStep 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;
```

s104d04

2. Horizontal Stacking – Proc SQL Inner Join vs DataStep 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
```

s104d05

2. Horizontal Stacking – Proc SQL Inner Join vs DataStep Merge

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

```
data work.new;
   merge customer2(in=InCust)transaction2(in=InTrans);
   by ID;
   if InCust=1 and InTrans=1;
run;

proc print data=work.new;
run;
```

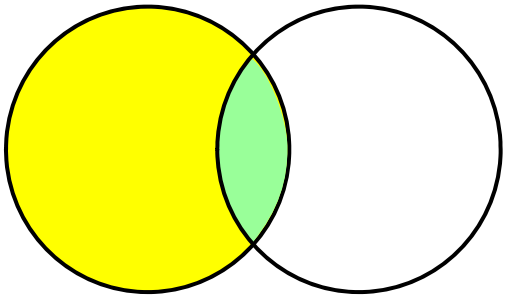
Obs	ID	Name	Action	Amount
1	102	Kent	Purchase	$376
2	102	Kent	Return	$119

2. Horizontal Stacking - Proc SQL Outer Join

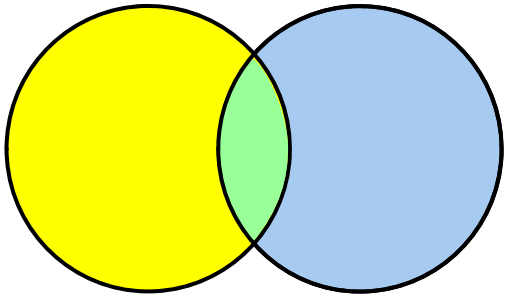
Concepts

You can retrieve both non-matching and matching rows using an outer join.

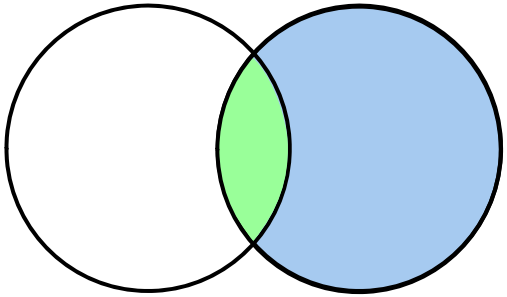
Outer joins include left, full, and right outer joins. Many tables can be referenced in outer joins. The tables are processed two tables at a time.



Left



Full

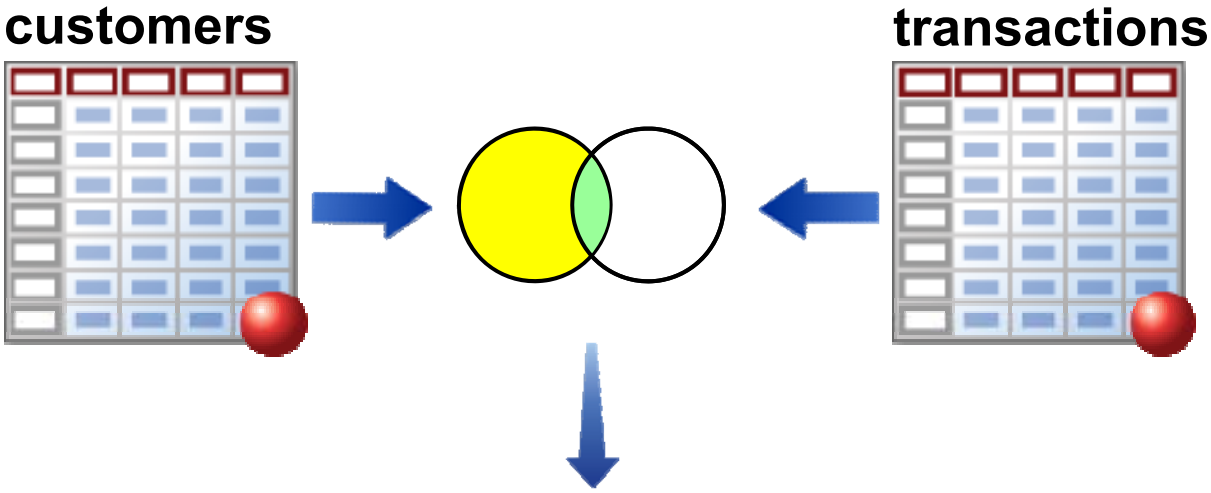


Right

2. Horizontal Stacking - Proc SQL Outer Join

Concepts

all customers and any recent transactions that they have completed.



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

2. Horizontal Stacking - Proc SQL Outer Join

Syntax

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;
```

s104d07

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 join criteria in outer joins.

2. Horizontal Stacking - Proc SQL Outer Join

Output

PROC SQL Output

All Customers

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

2. Horizontal Stacking - Proc SQL Left Join

Concepts

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

2. Horizontal Stacking - Proc SQL Right Join

Concepts

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.

s104d09

2. Horizontal Stacking - Proc SQL Full Join

Concepts

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

s104d10

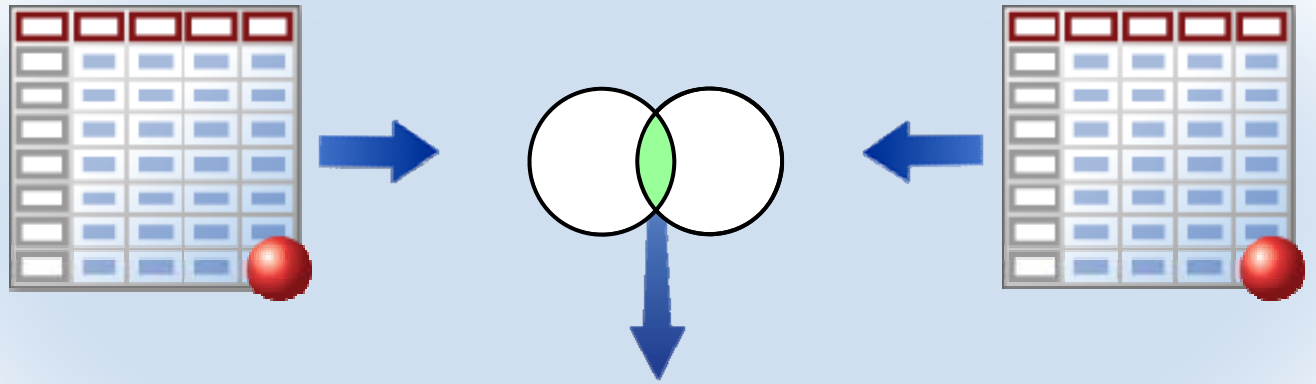
2. Horizontal Stacking - Proc SQL Self Join

Concepts

name of all sales employees and the name of each employee's direct manager.

orion.employee_addresses

orion.employee_organization



Employee ID	Employee Name	Manager ID	Manager Name	Country
120140	Minas, Michael	120103	Dawes, Wilson	AU
120141	Liebman, Amanda	120103	Dawes, Wilson	AU
120142	Eastley, Vincent	120103	Dawes, Wilson	AU
120143	Sloey, Phu	120103	Dawes, Wilson	AU
120144	Barbis, Viney	120103	Dawes, Wilson	AU

2. Horizontal Stacking - Proc SQL Self Join

Concepts

To return the employee name and the manager name, you need to read the **addresses** table twice.

1. Return the employee's ID and name.

addresses

EMP_ID	EMP_NAME
100	John
101	Sue

organization

EMP_ID	MGR_ID
100	101
101	57

EMP_ID	EMP_Name	MGR_ID	MGR_Name
100	John		

2. Horizontal Stacking - Proc SQL Self Join

Concepts

To return the employee name and the manager name, you need to read the **addresses** table twice.

1. Return the employee's ID and name.
2. Determine the ID of the employee's manager.

addresses

EMP_ID	EMP_NAME
100	John
101	Sue

organization

EMP_ID	MGR_ID
100	101
101	57

EMP_ID	EMP_Name	MGR_ID	MGR_Name
100	John	101	

2. Horizontal Stacking - Proc SQL Self Join

Concepts

To return the employee name and the manager name, you need to read the **addresses** table twice.

1. Return the employee's ID and name.
2. Determine the ID of the employee's manager.
3. Return the manager's name.

addresses

EMP_ID	EMP_NAME
100	John
101	Sue

organization

EMP_ID	MGR_ID
100	101
101	57

EMP_ID	EMP_Name	MGR_ID	MGR_Name
100	John	101	Sue

2. Horizontal Stacking - Proc SQL Self Join

Syntax

```
select e.Employee_ID "Employee ID",
       e.Employee_Name "Employee Name",
       m.Employee_ID "Manager ID",
       m.Employee_Name "Manager Name",
         e.Country
   from orion.employee_addresses as e,
        orion.employee_addresses as m,
        orion.employee_organization as o
   where e.Employee_ID=o.Employee_ID and
         o.Manager_ID=m.Employee_ID and
           Department contains 'Sales'
   order by Country,4,1;
```

s104d14

2. Horizontal Stacking - Proc SQL Self Join

Output

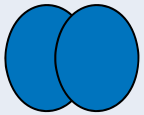
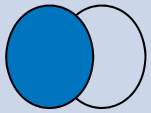
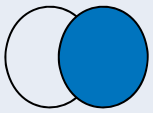
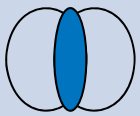
Partial PROC SQL Output

Employee ID	Employee Name	Manager ID	Manager Name	Country
120140	Minas, Michael	120103	Dawes, Wilson	AU
120141	Liebman, Amanda	120103	Dawes, Wilson	AU
120142	Eastley, Vincent	120103	Dawes, Wilson	AU
120143	Sloey, Phu	120103	Dawes, Wilson	AU
120144	Barbis, Viney	120103	Dawes, Wilson	AU

2. Comparing Datastep Merge with PROC SQL Join

Action	Data Step Merge	PROC SQL Join
Data manipulation	Arrays, first. / last. processing	Pass through queries to RDBMS
Sorting before Join/Merge	Yes explicit sorting prerequisite	No Explicit Sorting Implicit SQL sort behind the scenes
Join/Merge conditions	Equality only	Flexibility in join-operators, e.g. date between X and Y
Join/Merge columns	Same named column required	Same named columns not required
Multi table joins	Multi-table joins on different keys requires multiple sorts/merges	Multiple tables can be joined on multiple keys in one query
Common features	• data set options allows variables to be dropped, renamed, kept etc • vast majority of SAS functions are available in both data step and SQL	

Merging -SQL vs. Data Step

VISUAL	DATA STEP Merge	SQL Join
All rows from both tables 	Data tableC; Merge tableA tableB; By id; Run;	**Full Outer Join** Select * from tableA Full outer Join tableB On tableA.id=tableB.id;
All rows from left table & matching rows from right table 	Data tableD; Merge tableA(in=INA) tableB; By id; If INA; Run;	**Left Join** Select * from tableA Left Join tableB On tableA.id=tableB.id;
All rows from right table & matching rows from left table 	Data tableE; Merge tableA tableB(in=INB); By id; If INB; Run;	**Right Join** Select * from tableA Right Join tableB On tableA.id=tableB.id;
Matching rows only from both tables 	Data tableF; Merge tableA(in=INA) tableB(in=INB); By id; If INA and INB; Run;	**Inner Join** Select * from tableA, tableB INNER JOIN On tableA.id=tableB.id;

Questions?

Useful Links

- Life saver tip for comparing PROC SQL join with SAS data step merge
- When Proc Append may make more sense than the Data Step
- Data step vs. PROC SQL: what's a neophyte to do?
- What's in a name – SQL Join or Set Operation
- Combining SAS datasets: Methods
- PROC Append alternatives

Thank you!

Charu Shankar
Senior Technical Training Specialist
SAS Institute Inc. Toronto

BLOG http://blogs.sas.com/content/sastraining/author/charushankar/
LINKEDIN http://ca.linkedin.com/in/charushankar
TWITTER https://twitter.com/CharuSAS
EMAIL Charu.Shankar@sas.com