

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

MWSUG
October 2024

Charu Shankar
SAS Education



Charu Shankar, SAS® Institute

With a background in computer systems management. SAS Instructor Charu Shankar engages with logic, visuals, and analogies to spark critical thinking since 2007.

Charu curates and delivers unique content on SAS, SQL, Viya, etc. to support users in the adoption of SAS software.

When not coding, Charu teaches yoga and loves to explore Canadian trails with her husky Miko.



Agenda



- Introduction



- Vertical Stacking
 - Data/Proc Steps
 - PROC SQL Set Operation



- Horizontal Stacking
 - Data Step Merge
 - PROC SQL Joins



- Handy Links

Terminology

	Vertical Stacking		Horizontal Stacking	
	Data Step	PROC SQL	Data Step	PROC SQL
<i>terminology</i>	<i>Concatenate</i>	<i>Set Operation</i>	<i>Merge</i>	<i>Join</i>
	Proc Append		Match Merge	✓ Inner Join
	Interleave		Non Matches	✓ Outer Join
	Proc Datasets			• Left Join
				• Right Join
				• Full Join

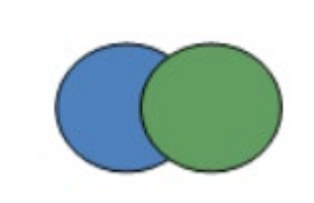

visually the difference can be explained as follows - joins tend to extend breadthways, but set operations in depth.

Before we look at the effect of this statement, let's look at the syntax and compare it to that of the join. Notice that "UNION" is inserted between two SELECTs (each of which has, as it must, a subordinate FROM clause). A set operator works on the results of two SELECTs. This is unlike a join, which is implemented within the FROM clause of a single SELECT.

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	<i>Concatenate</i>	<i>Set Operations</i>
	Data Step Concatenate	Union
	Proc Append	Outer Union
		Except
		Intersect
Horizontal Stacking	<i>Merge</i>	<i>Joins</i>
	Match Merge	Inner Join
	Non Matches	Outer Join
		Left Join
		Right Join
		Full Join

Vertical Stacking



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

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;
```

Vertical Stacking – Proc Datasets-Syntax

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

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

Vertical Stacking – Proc Append - Output

Like-Structured Data Sets

```
84  proc append base=emps  
85                data=emps2008;  
86  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.

emps

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

Vertical Stacking – Proc Append

unique situations

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.

Vertical Stacking – Proc Append - Advantage

1. Processing time

Vertical Stacking – proc append is the fastest concatenation technique as it doesn't read the base dataset

```
proc append  
    base=prepsales  
    data=sales;  
run;
```

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.

Vertical Stacking – Data Step - Concepts

Compilation

empsdk

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

empsfr

First	Gender	Country
Pierre	M	France
Sophie	F	France

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

PDV

First	Gender	Country

empsall1

First	Gender	Country
-------	--------	---------

Vertical Stacking – Data Step - Concepts

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

Vertical Stacking – Data Step - Concepts

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.
```

Vertical Stacking –Data Step - Scenarios

Scenario 1 – Standard Data Step Concatenate

Create new table by reading every row in every table listed on the set statement

```
data dataconc;  
set aep.prepsales  
    aep.nonsales;  
run;
```

```
SET SAS-data-set1 SAS-data-set2 ...;
```

- The SET statement reads observations from each data set in the order in which they are listed.
- Any number of data sets can be included in the SET statement

p110d01

Vertical Stacking – Data Step - Advantage

Scenario 2. Build new variables

```
data dataconc;  
    set aep.prepsales  
        aep.nonsales;  
    sales='YS' ;  
    nonsales='YN' ;  
run;
```

p110d01

Vertical Stacking – Data Step - Advantage

Scenario 3. Determine which table contributed to final results

The data step can keep all uncommon columns, plus determine which table contributed to final results.

```
data dataconc;  
    set aep.prepsales(in=ins)  
        aep.nonsales(in=inn) ;  
    sales=ins;  
    nonsales=inn;  
run;
```

Vertical Stacking – Data Step Tips

Data step concatenate

DATA STEP concatenation with the SET statement will almost always be the best method for creating a table from several input datasets in the following cases

- read in all input datasets with a SET statement without any prior data manipulation and
- perform any subsequent necessary data manipulation within the same DATA STEP

Comparing PROC Append - Data Step Concatenate

Criterion	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 concatenates much faster since it doesn't process observations from BASE= data set	
Build new variables	Cannot build new variables as descriptor portion information in base SAS dataset cannot change	The data step can build new variables.
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.

Using Set Operators

Vertical Stacking – PROC SQL Set Operation

```
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.

Objectives

Vertical Stacking – PROC SQL Set operations

Which employees
have completed
training A or B?



Business Data

Vertical Stacking – PROC SQL Set Operation

The data required to answer the questions is stored in two tables.

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**

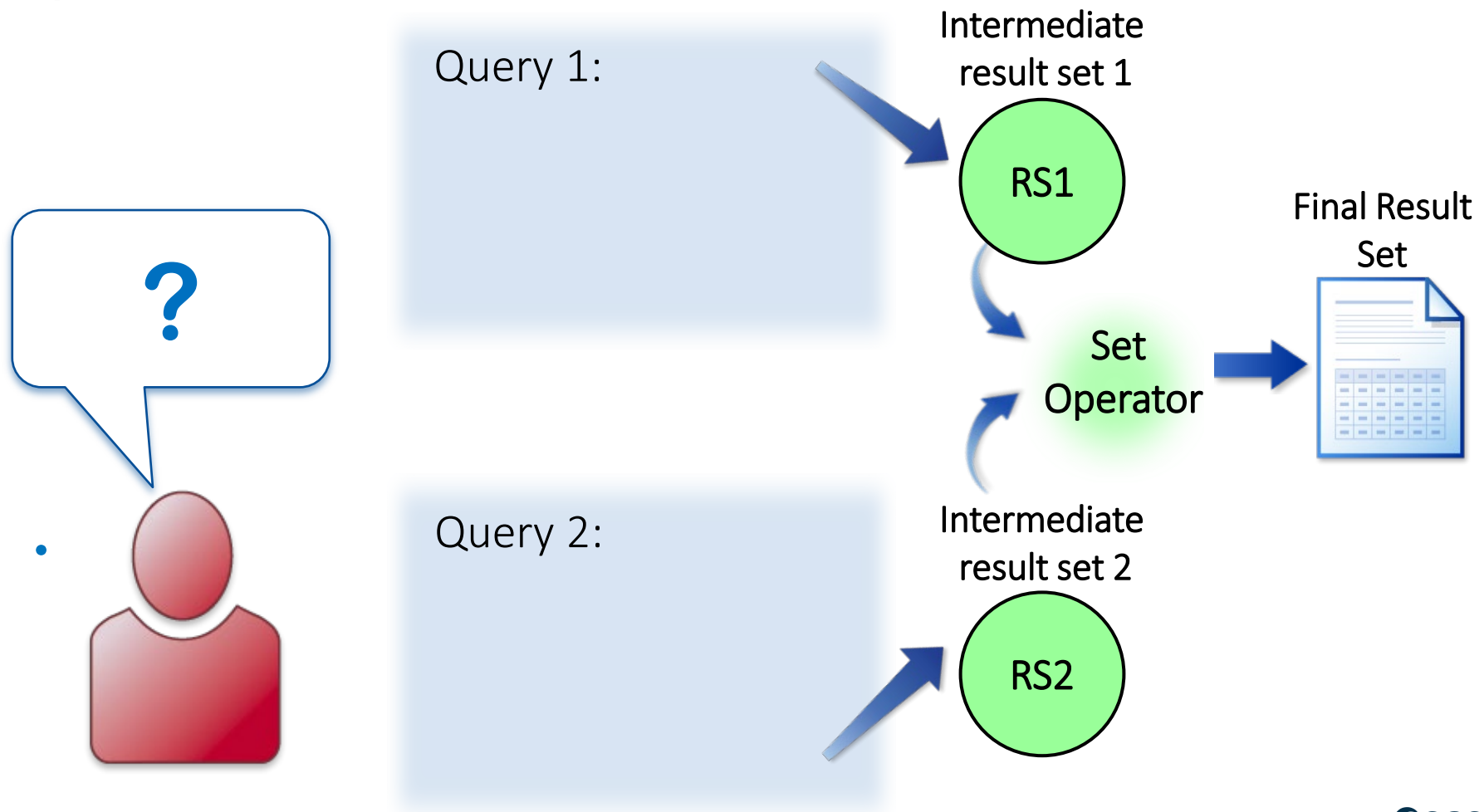
Name	ID	SDate	EDate
Bob	11	9JUL2012	13JUL2012
Pam	15	25JUL2012	27JUL2012
Kyle	19	12JUL2012	20JUL2012
Chris	21	29JUL2012	.

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

Using Set Operators

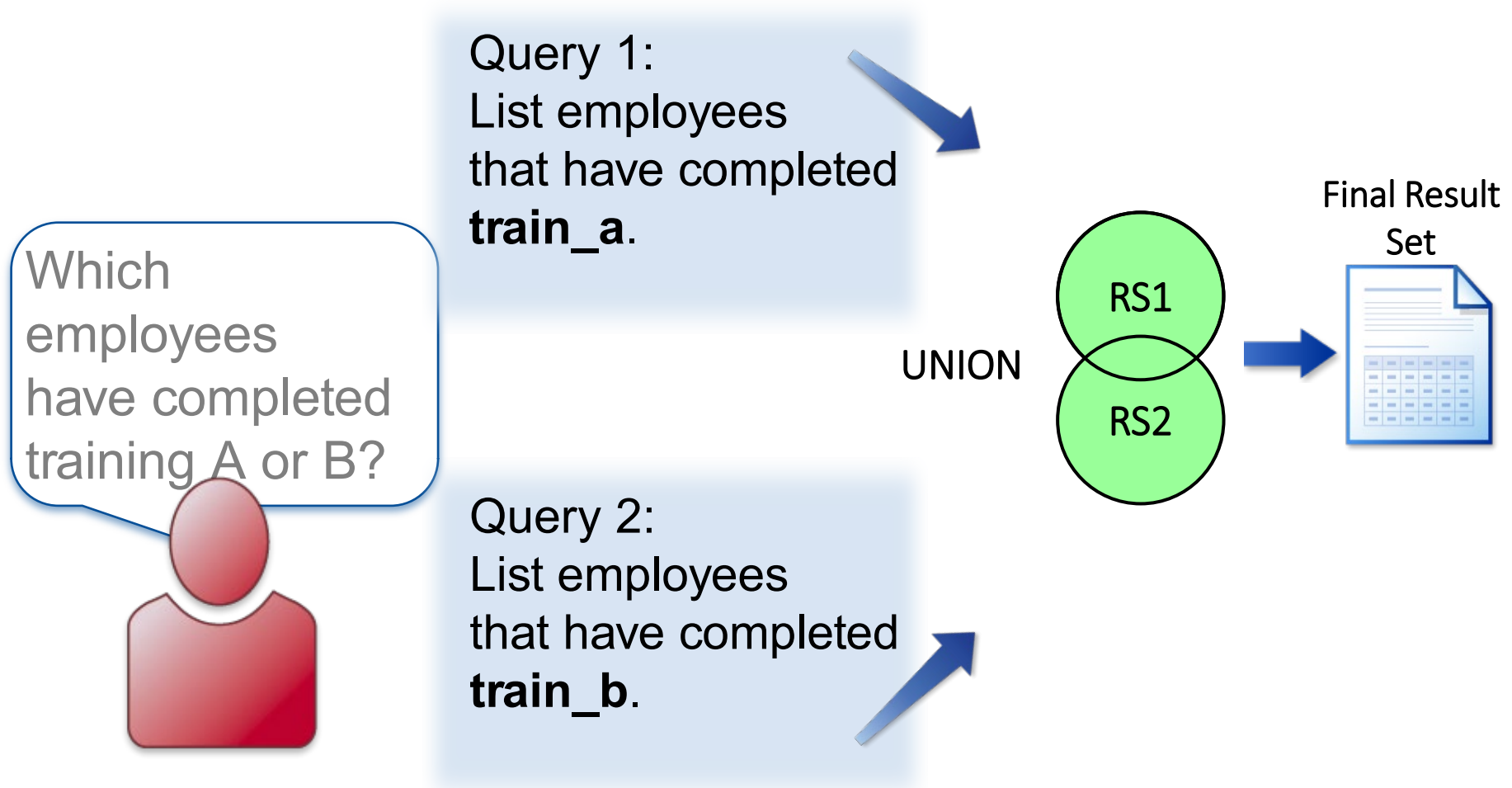
Vertical Stacking – PROC SQL Set Operation

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



UNION Set Operator - Concept

Vertical Stacking – PROC SQL Set Operation



UNION Set Operator - Syntax

The manager requested a report that shows employees from both result sets; the UNION operator is appropriate.

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

UNION Set Operator - Output

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

OUTER UNION Set Operator - concept

Vertical Stacking – PROC SQL Set Operation

Query 1:
List employees
that have completed
train_a and the
completion date.

Query 2:
List employees
that have completed
train_b and the
completion date.

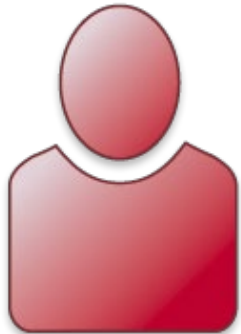
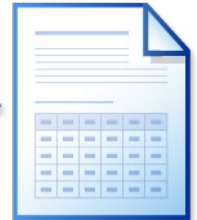
Which employees
have completed
training A and/or B
and on what dates?

OUTER
UNION

RS1

RS2

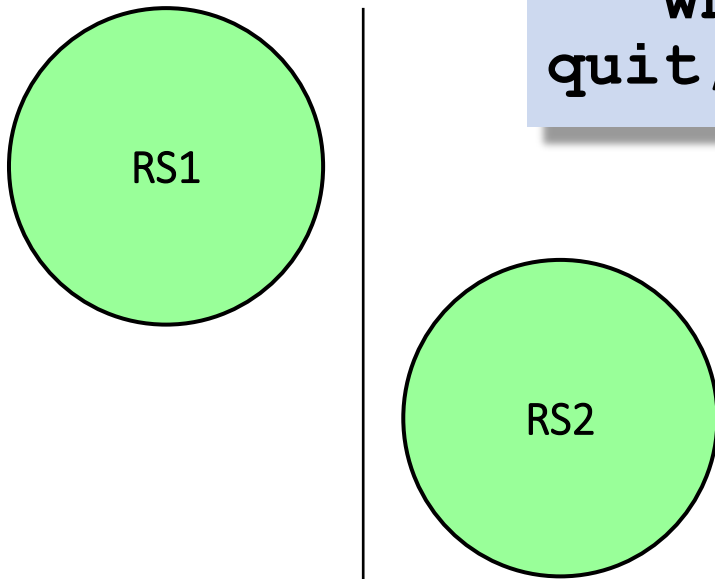
Final Result
Set



OUTER UNION - Syntax

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

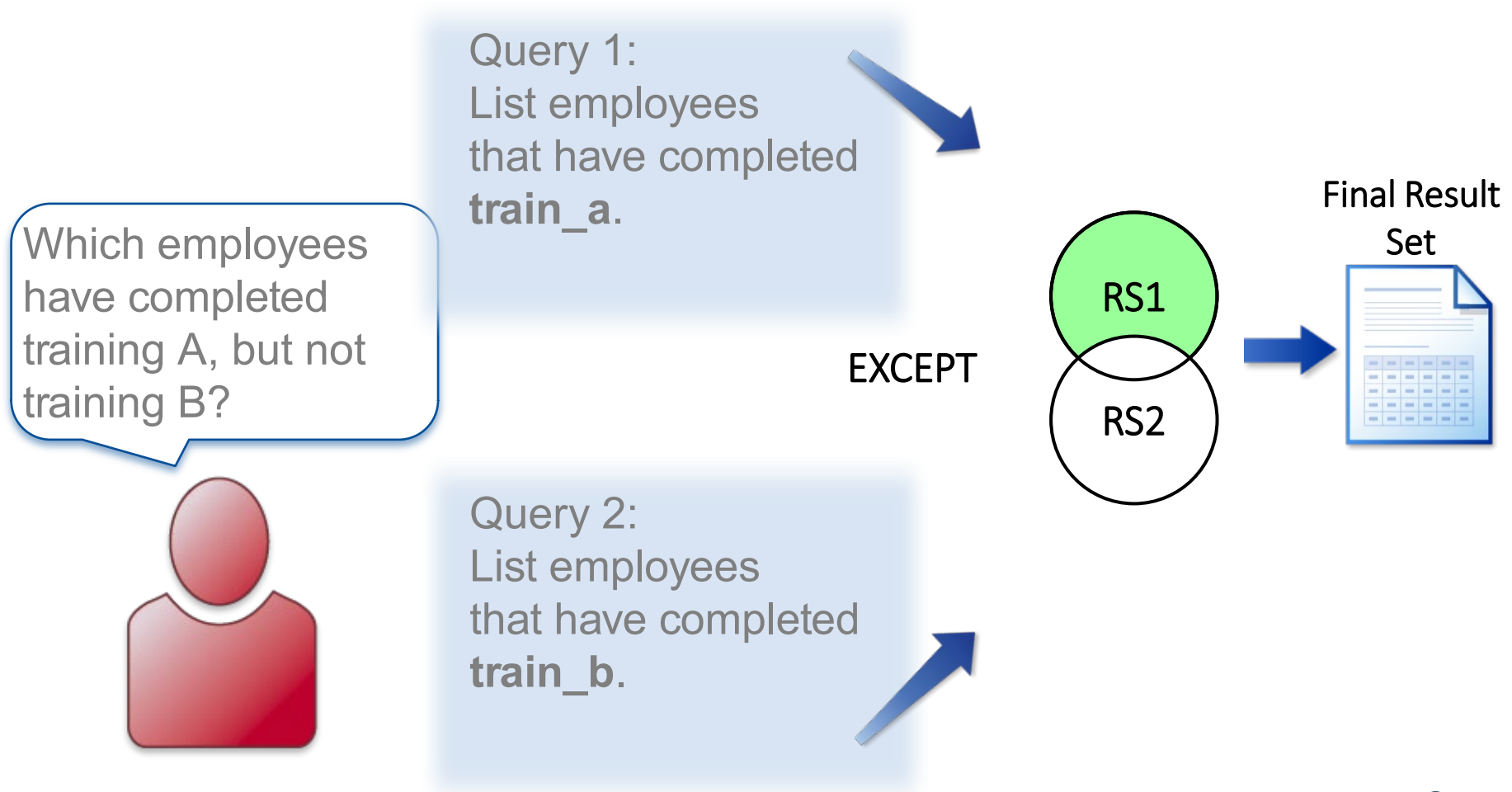
OUTER UNION Set Operator - Output

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

EXCEPT Set Operator - Concept

Vertical Stacking – PROC SQL Set Operation



EXCEPT Set Operator - Syntax

Lists 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 ...
```

Except Set Operator - Output

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

INTERSECT Set Operator

Vertical Stacking – PROC SQL Set Operation

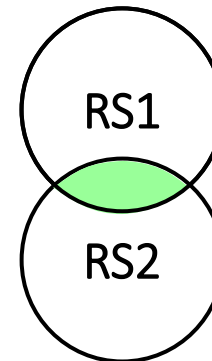
Which employees
have completed both
training A
and B?



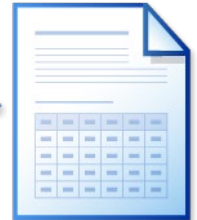
Query 1:
List employees
that have completed
train_a.

Query 2:
List employees
that have completed
train_b.

INTERSECT

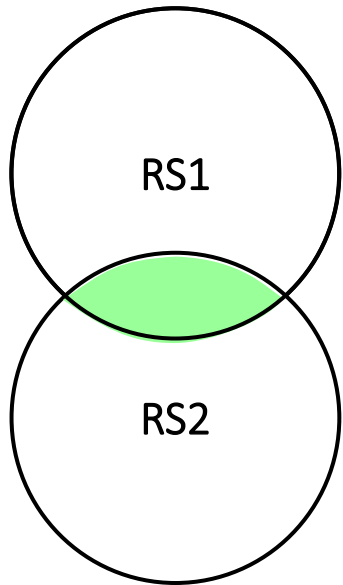


Final Result
Set



INTERSECT Set Operator - Syntax

This report requires 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

INTERSECT Set Operator - Output

**Employees Who Have Completed
Both Training Classes**

ID	Name
11	Bob
87	Ted

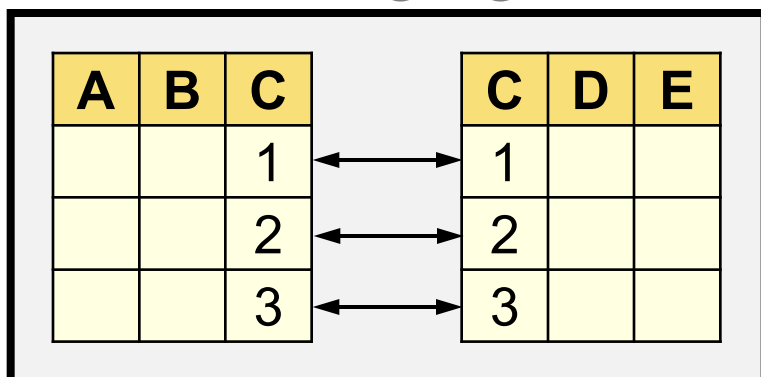
Default Behavior of Set Operators

Vertical Stacking – PROC SQL Set Operation

Set Operator	Rows	Columns
UNION	Unique rows from both result sets	Aligned by column position in both result sets
OUTER UNION	All rows from both result sets	All columns from both result sets
EXCEPT	Unique rows from the first result set, that are not in the second result set	Aligned by column position in both result sets
INTERSECT	Unique rows from the first result set that are in the second result set	Aligned by column position in both result sets

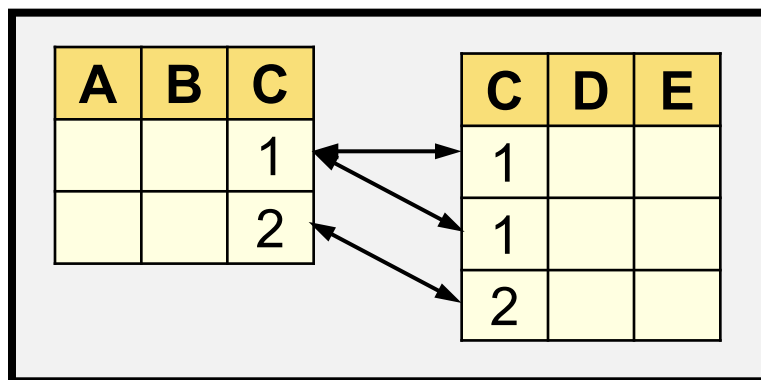
Horizontal stacking – data step Merge

Match-Merging



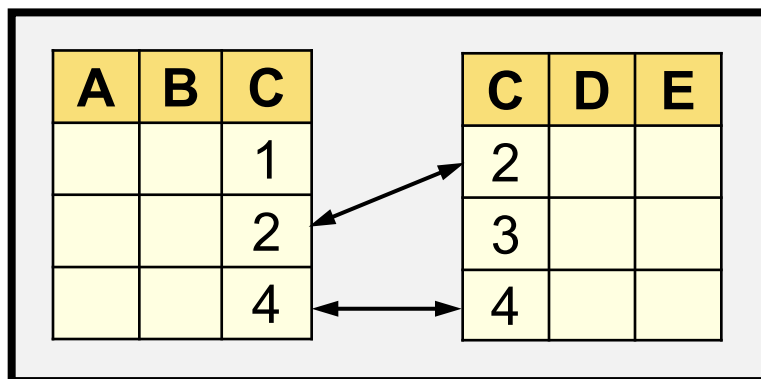
One-to-One

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



One-to-Many

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



Nonmatches

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

Horizontal stacking – data step Merge

Match-Merging: Sorting the Data Sets

The data sets in a match-merge must be sorted by the common variable or variables that are being matched.

```
PROC SORT DATA=input-SAS-data-set  
           <OUT=output-SAS-data-set>;  
           BY <DESCENDING> by-variable(s);  
RUN;
```

Horizontal Stacking – Data Step Merge

The *MERGE statement* in a DATA step joins observations from two or more SAS data sets into single observations.

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

A *BY statement* indicates a match-merge and lists the variable or variables to match.

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

Horizontal Stacking – Data Step Merge

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) ...
```

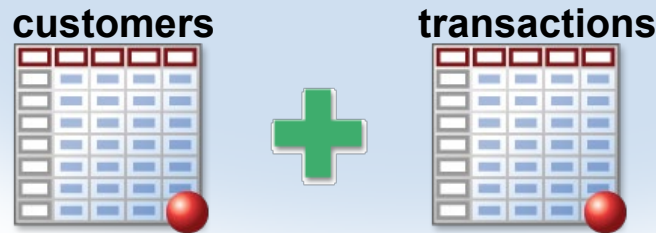
variable is a temporary numeric variable that has two possible values:

0	Indicates that the data set did <i>not</i> contribute to the current observation.
1	Indicates that the data set <i>did</i> contribute to the current observation.

Horizontal Stacking – PROC SQL Join

Combining Data from Multiple Tables

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

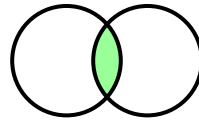


Horizontal Stacking – PROC SQL Join

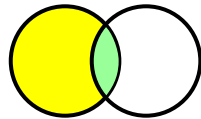
Types of Joins

PROC SQL supports two types of joins:

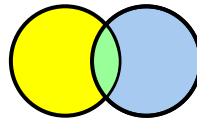
- *Inner joins* return only matching rows.



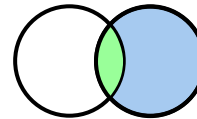
- *Outer joins* return all matching rows, plus nonmatching rows from one or both tables.



Left



Full



Right

Horizontal Stacking – PROC SQL Join

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.

Horizontal Stacking – PROC SQL Join

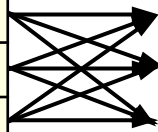
Non-Matching Data in the Cartesian Product

customers

ID	Name
101	Smith
104	Jones
102	Blank

transactions

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



Result Set

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

Non-matching IDs

9 rows

Horizontal Stacking – PROC SQL Join

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

Horizontal Stacking – PROC SQL Join

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
101	A

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

Horizontal Stacking – PROC SQL Join

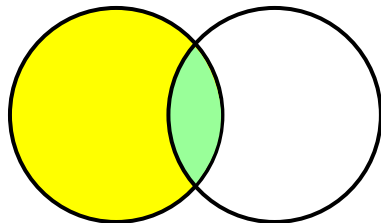
Compare SQL Join and DATA Step Merge

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

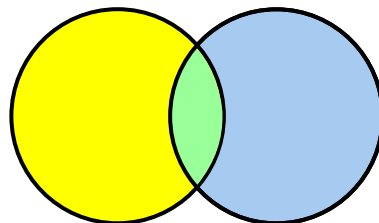
Horizontal Stacking – PROC SQL Join

Outer Joins

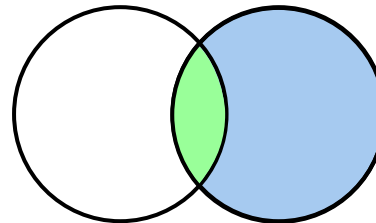
- You can retrieve both non-matching and matching rows using an outer join.
- Outer joins include left, full, and right outer joins. Many tables can be referenced in outer joins. The tables are processed two tables at a time.



Left



Full



Right

Horizontal Stacking – PROC SQL Join

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
```

The ON clause specifies the join criteria in outer joins.

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

Horizontal Stacking – PROC SQL Join

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	<ul style="list-style-type: none">■ Multiple tables, separated by commas, in the FROM clause■ WHERE clause that specifies join criteria	ON clause that specifies join criteria

Comparing Data Step Merge with PROC SQL Join

	Data Step Merge	PROC SQL Join
Pros	<ul style="list-style-type: none">• Allows use of data step syntax, e.g. arrays, first. / last. processing• Easier understanding and comprehension for novice users• Data step hash objects and arrays are available• Multiple output tables can be created with a single read of the input dataset.	<ul style="list-style-type: none">• Sorts need not be explicitly sorted as SQL will sort behind the scenes• Multiple tables can be joined on multiple keys (even if variable types differ) in one query - Easier to deal with ranges on join, e.g. date between X and Y• Join query can also summarize• Pass-thru queries to RDBMS• Use of efficient hash joins
Cons	<ul style="list-style-type: none">• Data steps must be sorted or indexed before merging• Multi-table joins on different keys requires multiple sorts/merges• BY variables must be same type (char or num)	<ul style="list-style-type: none">• Limitations of SQL apply, e.g. flexibility of data step syntax is lost• SQL cannot read raw data files and an extra data step is required first.• Multiple output tables need multiple SQL steps in order to create them

- Wash
- Use of data set options allows variables to be dropped, renamed, kept etc....
 - the vast majority of SAS functions are available in both data step and SQL

Dataset A & Dataset B

```
Data tableA;  
  Input id $ Name $;  
  Datalines;  
  E01 John  
  E02 David  
  E03 Alice  
  E04 Betty  
  ;  
Run;
```

```
Data tableB;  
  Input id $ age;  
  Datalines;  
  E01 25  
  E03 30  
  E05 23  
  E06 34  
  ;  
Run;
```

Dataset A & Dataset B

```
Data tableA;  
  Input id $ Name $;  
  Datalines;  
  E01 John  
  E02 David  
  E03 Alice  
  E04 Betty  
  ;  
Run;
```

```
Data tableB;  
  Input id $ age;  
  Datalines;  
  E01 25  
  E03 30  
  E05 23  
  E06 34  
  ;  
Run;
```


Choosing a Technique

There are many ways to perform a task using PROC SQL and non-SQL base SAS for data management techniques. While elegance vs. functionality is always a consideration, the choice of one technique over another should be made based on

Familiarity	In today's fast pace, demanding faster and more accurate results, the programmer would employ techniques that are familiar and comfortable.
Goal	What is the ultimate goal of the effort? Where will the code reside? Pertinent design questions to be considered. Data management and extraction with PROC SQL can become as powerful (and convoluted) as non-SQL base SAS programming. Complex grouping, and complicated summarizations can be accomplished with either facility
Correctness	Care in the choice of tools must be taken. While all techniques will provide results, not all results are correct relative to the process design.
Maintenance	While succinct code is often desirable, emphasis should be placed on clear, concise (but not necessarily terse), and maintainable code which is An often-overlooked feature of program development
Human Efficiency	From a programmer perspective efficiency relates to the comfort level and skill sets of the individual programmer.
Machine processing efficiency	Users need to consider these gains or losses within their own environment based on platform, file size, use of indices (or not), and use of RDBMSs.

Handy Links

- <http://blogs.sas.com/content/sastraining/2012/03/23/whats-in-a-name-sql-join-or-set/>
- <http://support.sas.com/documentation/cdl/en/lrcon/65287/HTML/default/viewer.htm#n1tgk0uanvisvon1r26lc036k0w7.htm>
- <http://www2.sas.com/proceedings/forum2008/085-2008.pdf>
- <http://www2.sas.com/proceedings/sugi29/269-29.pdf>
- Ask the Expert – How many ways can you join SAS Tables, Charu Shankar, <https://lnkd.in/gq-8k57i>

Thank You

Charu Shankar
SAS Institute Toronto

EMAIL Charu.shankar@sas.com
BLOG <https://blogs.sas.com/content/author/charushankar/>
TWITTER [CharuYogaCan](#)
LINKEDIN <https://www.linkedin.com/in/charushankar/>
LinkedIn Group <https://www.linkedin.com/groups/5095978>

