

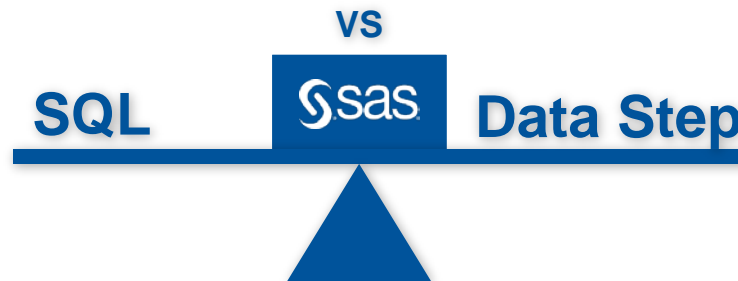
SQL VS. DATA STEP PROCESSING



Agenda

PROC SQL VS. DATA STEP PROCESSING

- Joining data
- Additional comparisons
 - Conditional processing
 - Indexing data
 - Subsetting
 - Sorting and summarizing
 - Creating macro variables



JOINING SAS DATA USING THE DATA STEP AND PROC SQL



TYPES OF JOINS

ANSI STANDARD SQL

- Natural
 - Uses no 'keys' – typically termed a Cartesian product
- Inner
- Outer Joins
 - Left
 - Right
 - Full

PROC SQL and the DATA Step can deliver the same results in many cases

WHAT DOES THE DATA LOOK LIKE?

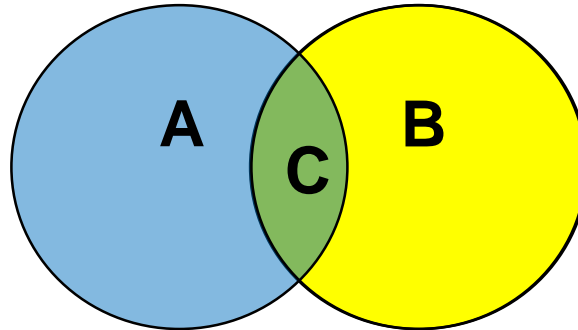
- One to One
- One to Many
- Many to One
- Many to Many

PROC SQL and the DATA Step can deliver the same results for one-to-one and one-to-many; they produce different results for many-to-many.

TYPES OF JOINS INNER JOIN

Inner Join

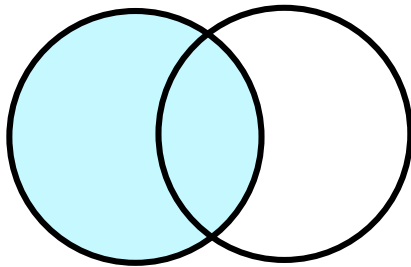
- The intersection of two or more sets
- Return only matching rows



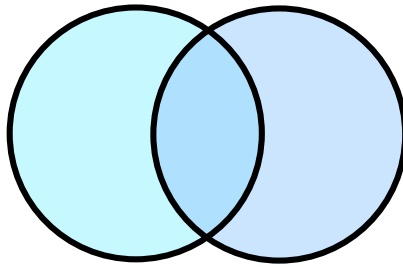
TYPES OF JOINS **OUTER JOINS**

Outer Joins

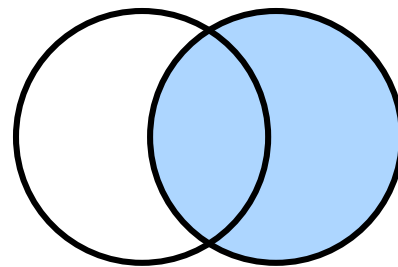
- return all matching rows, plus nonmatching rows from one or both tables
- can be performed on only two tables or views at a time.



Left



Full



Right

HOW DO THE TECHNIQUES COMPARE?

SAMPLE DATA: ONE-TO-ONE JOINS

FRUITS

DAY	FRUIT
1	apples
3	apples
4	apples
6	apples

VEGGIES

DAY	VEGGIE
1	broccoli
2	broccoli
4	broccoli
5	broccoli

JOINS ONE-TO-ONE JOINS: DEFAULT BEHAVIOR

```
proc sql;  
    create table sql_default_join1 as  
        select f.*, v.*  
            from fruit f, veggies v;  
run;
```

Don't do this –
The result is a
Cartesian join

Don't do this –
Overlaid values
result

```
data data_default_join1;  
    merge fruits veggies;  
run;
```

HOW DO THE
TECHNIQUES
COMPARE?

JOINS: DEFAULT BEHAVIOR

Default SQL Join

DAY	FRUIT	VEGGIE
1	apples	broccoli
1	apples	broccoli
1	apples	broccoli
1	apples	broccoli
3	apples	broccoli
3	apples	broccoli
3	apples	broccoli
3	apples	broccoli
4	apples	broccoli
4	apples	broccoli
4	apples	broccoli
4	apples	broccoli
4	apples	broccoli
6	apples	broccoli
6	apples	broccoli
6	apples	broccoli
6	apples	broccoli

Default DATA Step Join

DAY	FRUIT	VEGGIE
1	apples	broccoli
2	apples	broccoli
4	apples	broccoli
5	apples	broccoli

JOINS ONE-TO-ONE JOINS: USING A KEY

```
proc sql;  
  create table sql_key_join2 as  
    select f.*, v.veggie  
      from fruits f, veggies v  
      where f.day=v.day;  
run;
```

SORT steps are
needed if the data
is not already
in BY order

```
data data_key_join2 ;  
  merge fruits veggies;  
  by day;  
run;
```

HOW DO THE TECHNIQUES COMPARE?

JOINS: USING A KEY

SQL Join with Key

DAY	FRUIT	VEGGIE
1	apples	broccoli
4	apples	broccoli

DATA Step Join with Key

DAY	FRUIT	VEGGIE
1	apples	broccoli
2		broccoli
3	apples	
4	apples	broccoli
5		broccoli
6	apples	

```
proc sql;  
  create table sql_key_join2 as  
    select f.*, v.veggie  
      from fruits f, veggies v  
      where f.day=v.day;  
run;
```

(same step as
before)

Add the
IN=
operator

```
data data_key_join2b ;  
  merge fruits (in=f) veggies (in=v);  
  by day;  
  if f and v;  
run;
```

HOW DO THE TECHNIQUES COMPARE?

JOINS: USING A KEY (modified DATA Step)

SQL Join with Key

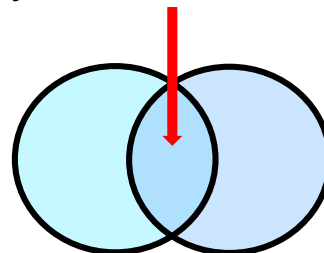
DAY	FRUIT	VEGGIE
1	apples	broccoli
4	apples	broccoli

DATA Step Join with Key and IN=

DAY	FRUIT	VEGGIE
1	apples	broccoli
4	apples	broccoli

A MATCH!

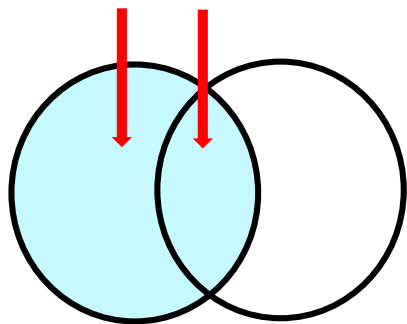
This type of join is called an INNER join – it returns only the rows where the key field matches.



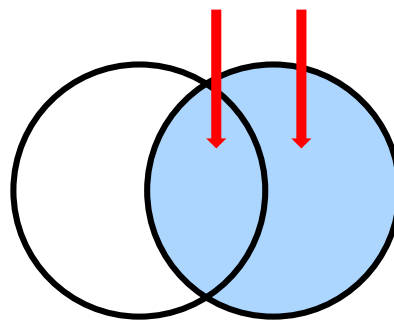
Inner Join

JOINS ONE-TO-ONE DATA: LEFT AND RIGHT (OUTER) JOINS

- Return all matching rows, plus nonmatching rows from the left or the right table
- Can be performed on only two tables or views at a time.



Left Join



Right Join

JOINS ONE-TO-ONE: LEFT JOIN

```
proc sql;  
  create table sql_left_join3 as  
  select f.*, v.*  
    from fruits f  
  left join  
    veggies v  
  on f.day = v.day;  
run;
```

```
data data_left_join3a;  
  merge fruits (in=f) veggies (in=v);  
  by day;  
  if f;  
run;
```


HOW DO THE TECHNIQUES COMPARE?

JOINS: Left Join

SQL Left Join

DAY	FRUIT	VEGGIE
1	apples	broccoli
3	apples	
4	apples	broccoli
6	apples	

DATA Step Left Join

DAY	FRUIT	VEGGIE
1	apples	broccoli
3	apples	
4	apples	broccoli
6	apples	

A MATCH!

JOINS ONE-TO-ONE: RIGHT JOIN

```
proc sql;  
  create table sql_right_join4 as  
    select f.fruit, v.*  
    from fruits f  
    right join  
    veggies v  
    on f.day = v.day;  
run;
```

```
data data_right_join4a;  
  merge fruits (in=f) veggies (in=v);  
  by day;  
  if v;  
run;
```

HOW DO THE TECHNIQUES COMPARE?

JOINS: Right Join

SQL Right Join

FRUIT	DAY	VEGGIE
apples	1	broccoli
	2	broccoli
apples	4	broccoli
	5	broccoli

DATA Step Right Join

DAY	FRUIT	VEGGIE
1	apples	broccoli
2		broccoli
4	apples	broccoli
5		broccoli

**A MATCH!
(ALMOST)**

JOINS ONE-TO-ONE: RIGHT JOIN

```
proc sql;  
  create table sql_right_join4a as  
    select v.day, f.fruit, v.veggie  
    from fruits f  
    right join  
    veggies v  
    on f.day=v.day;  
quit;
```

(same step as
before)

```
data data_right_join4a;  
  merge fruits (in=f) veggies (in=v);  
  by day;  
  if v;  
run;
```

HOW DO THE TECHNIQUES COMPARE?

JOINS: Right Join

SQL Right Join

DAY	FRUIT	VEGGIE
1	apples	broccoli
2		broccoli
4	apples	broccoli
5		broccoli

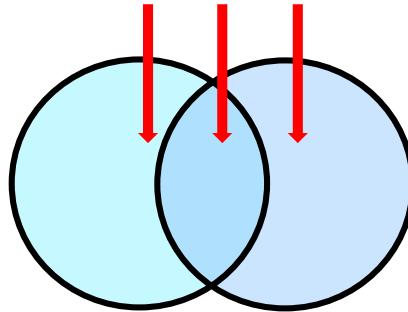
DATA Step Right Join

DAY	FRUIT	VEGGIE
1	apples	broccoli
2		broccoli
4	apples	broccoli
5		broccoli

A MATCH!

JOINS ONE-TO-ONE DATA: FULL (OUTER) JOIN

Retrieve the matching rows as well as the non-matches from the left table and the non-matches from the right table.



Full Join

JOINS ONE-TO-ONE: FULL JOIN

```
proc sql;  
  create table sql_full_join5 as  
    select f.*, v.*  
    from fruits f full join veggies v  
    on f.day=v.day;  
run;
```

```
data full_join5;  
  merge fruits (in=f) veggies (in=v);  
  by day;  
  if f or v;  
run;
```

HOW DO THE TECHNIQUES COMPARE?

JOINS: Full (Outer) Join

SQL Full Join

DAY	FRUIT	VEGGIE
1	apples	broccoli
.		broccoli
3	apples	
4	apples	broccoli
.		broccoli
6	apples	

DATA Step Full Join

DAY	FRUIT	VEGGIE
1	apples	broccoli
2		broccoli
3	apples	
4	apples	broccoli
5		broccoli
6	apples	

**A MATCH!
(ALMOST)**

The COALESCE function returns the value of the first non-missing argument.

General form of the COALESCE function:

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

JOINS ONE-TO-ONE: FULL JOIN

```
proc sql;  
  create table sql_full_join5a as  
    select coalesce(f.day,v.day) as Key, f.fruit, v.veggie  
    from fruits f full join veggies v  
    on f.day=v.day;  
run;
```

Add the COALESCE function

```
data full_join5a;  
  merge fruits veggies;  
  by day;  
run;
```

Remove the IN= option and
IF statement

HOW DO THE TECHNIQUES COMPARE?

JOINS: Full (Outer) Join

SQL Full Join

Key	FRUIT	VEGGIE
1	apples	broccoli
2		broccoli
3	apples	
4	apples	broccoli
5		broccoli
6	apples	

DATA Step Full Join

DAY	FRUIT	VEGGIE
1	apples	broccoli
2		broccoli
3	apples	
4	apples	broccoli
5		broccoli
6	apples	

A MATCH!

HOW DO THE TECHNIQUES COMPARE?

SAMPLE DATA: ONE-TO-MANY JOINS

FRUITS2

DAY	FRUIT
1	apples
3	apples
4	apples
6	apples

VEGGIES2

DAY	VEGGIE
1	broccoli
1	tomatoes
1	lettuce
2	broccoli
2	tomatoes
2	lettuce
4	broccoli
4	tomatoes
4	lettuce
5	broccoli
5	tomatoes
5	lettuce

JOINS ONE-TO-MANY INNER JOIN

```
proc sql;  
  create table sql_1_to_m_join1 as  
    select f2.*, v2.veggie  
      from fruits2 f2 inner join  
      veggies2 v2  
     on f2.day=v2.day;  
run;
```

Alternate coding here –
use of “inner join” in
syntax

```
data data_1_to_m_join1;  
  merge fruits2 (in=f2) veggies2 (in=v2);  
  by day;  
  if f2 and v2;  
run;
```

HOW DO THE TECHNIQUES COMPARE?

ONE-to-MANY INNER JOIN

SQL One-to-Many Inner Join

DAY	FRUIT	VEGGIE
1	apples	broccoli
1	apples	tomatoes
1	apples	lettuce
4	apples	broccoli
4	apples	tomatoes
4	apples	lettuce

DATA Step One-to-Many Inner Join

DAY	FRUIT	VEGGIE
1	apples	broccoli
1	apples	tomatoes
1	apples	lettuce
4	apples	broccoli
4	apples	tomatoes
4	apples	lettuce

A MATCH!

JOINS ONE-TO-MANY LEFT JOIN

```
proc sql;  
  create table sql_1_to_m_left_join2 as  
  select f2.*, v2.*  
    from fruits2 f2  
  left join  
    veggies2 v2  
  on f2.day = v2.day;  
run;
```

```
data data_1_to_m_left_join2a;  
  merge fruits2 (in=f2) veggies2 (in=v2);  
  by day;  
  if f2;  
run;
```

HOW DO THE
TECHNIQUES
COMPARE?

ONE-to-MANY LEFT JOIN

SQL One-to-Many Left Join

DAY	FRUIT	VEGGIE
1	apples	tomatoes
1	apples	lettuce
1	apples	broccoli
3	apples	
4	apples	broccoli
4	apples	lettuce
4	apples	tomatoes
6	apples	

DATA Step One-to-Many Left Join

DAY	FRUIT	VEGGIE
1	apples	broccoli
1	apples	tomatoes
1	apples	lettuce
3	apples	
4	apples	broccoli
4	apples	tomatoes
4	apples	lettuce
6	apples	

A MATCH!

JOINS ONE-TO-MANY RIGHT JOIN

```
proc sql;  
  create table sql_1_to_m_right_join3 as  
    select v2.day as key, f2.fruit, v2.veggie  
    from fruits2 f2  
    right join  
    veggies2 v2  
    on f2.day = v2.day;  
run;
```

```
data data_1_to_m_right_join3;  
  merge fruits2 (in=f2) veggies2 (in=v2);  
  by day;  
  if v2;  
run;
```

HOW DO THE TECHNIQUES COMPARE?

ONE-to-MANY RIGHT JOIN

SQL One-to-Many Right Join

DAY	FRUIT	VEGGIE
1	apples	tomatoes
1	apples	lettuce
1	apples	broccoli
2		lettuce
2		tomatoes
2		broccoli
4	apples	broccoli
4	apples	lettuce
4	apples	tomatoes
5		lettuce
5		tomatoes
5		broccoli

DATA Step One-to-Many Right Join

DAY	FRUIT	VEGGIE
1	apples	broccoli
1	apples	tomatoes
1	apples	lettuce
2		broccoli
2		tomatoes
2		lettuce
4	apples	broccoli
4	apples	tomatoes
4	apples	lettuce
5		broccoli
5		tomatoes
5		lettuce

A MATCH!

JOINS ONE-TO-MANY FULL JOIN

```
proc sql;  
  create table sql_1_to_m_full_join4 as  
    select coalesce(f2.day,v2.day) as Key, f2.fruit, v2.veggie  
    from fruits2 f2 full join veggies2 v2  
    on f2.day=v2.day;  
run;
```

```
data data_1_to_m_full_join4;  
  merge fruits2 veggies2;  
  by day;  
run;
```

HOW DO THE
TECHNIQUES
COMPARE?

ONE-to-MANY FULL JOIN

A MATCH!

SQL One-to-Many Full Join

Key	FRUIT	VEGGIE
1	apples	tomatoes
1	apples	lettuce
1	apples	broccoli
2		lettuce
2		tomatoes
2		broccoli
3	apples	
4	apples	broccoli
4	apples	lettuce
4	apples	tomatoes
5		lettuce
5		tomatoes
5		broccoli
6	apples	

DATA Step One-to-Many Full Join

DAY	FRUIT	VEGGIE
1	apples	broccoli
1	apples	tomatoes
1	apples	lettuce
2		broccoli
2		tomatoes
2		lettuce
3	apples	
4	apples	broccoli
4	apples	tomatoes
4	apples	lettuce
5		broccoli
5		tomatoes
5		lettuce
6	apples	

HOW DO THE TECHNIQUES COMPARE?

SAMPLE DATA: MANY-TO-MANY JOINS

FRUITS3

DAY	FRUIT
1	apples
1	bananas
1	oranges
3	apples
3	bananas
3	oranges
4	apples
4	bananas
4	oranges
6	apples
6	bananas
6	oranges

VEGGIES3

DAY	VEGGIE
1	broccoli
1	tomatoes
1	lettuce
2	broccoli
2	tomatoes
2	lettuce
4	broccoli
4	tomatoes
4	lettuce
5	broccoli
5	tomatoes
5	lettuce

JOINS MANY-TO-MANY INNER JOIN

```
proc sql;  
  create table sql_m_to_m_join1 as  
    select f3.*, v3.veggie  
      from fruits3 f3 inner join  
        veggies3 v3  
      on f3.day=v3.day;  
run;
```

```
data data_m_to_m_join1 ;  
  merge fruits3 (in=f3) veggies3 (in=v3);  
  by day;  
  if f3 and v3;  
run;
```

HOW DO THE TECHNIQUES COMPARE?

MANY-to-MANY INNER JOIN

NO MATCH

SQL Many-to-Many Inner Join

DAY	FRUIT	VEGGIE
1	apples	broccoli
1	apples	lettuce
1	apples	tomatoes
1	bananas	broccoli
1	bananas	lettuce
1	bananas	tomatoes
1	oranges	broccoli
1	oranges	lettuce
1	oranges	tomatoes
4	apples	broccoli
4	apples	lettuce
4	apples	tomatoes
4	bananas	broccoli
4	bananas	lettuce
4	bananas	tomatoes
4	oranges	broccoli
4	oranges	lettuce
4	oranges	tomatoes

DATA Step Many-to-Many Inner Join

DAY	FRUIT	VEGGIE
1	apples	broccoli
1	bananas	tomatoes
1	oranges	lettuce
4	apples	broccoli
4	bananas	tomatoes
4	oranges	lettuce

JOINS MANY-TO-MANY LEFT JOIN

```
proc sql;  
  create table sql_m_to_m_left_join2 as  
  select f3.*, v3.*  
    from fruits3 f3  
  left join  
    veggies3 v3  
    on f3.day = v3.day;  
run;
```

```
data data_m_to_m_left_join2;  
  merge fruits3 (in=f3) veggies3 (in=v3);  
  by day;  
  if f3;  
run;
```


HOW DO THE TECHNIQUES COMPARE?

MANY-to-MANY LEFT JOIN

NO MATCH

SQL Many-to-Many Left Join

DAY	FRUIT	VEGGIE
1	bananas	tomatoes
1	oranges	tomatoes
1	apples	tomatoes
1	bananas	lettuce
1	oranges	lettuce
1	apples	lettuce
1	bananas	broccoli
1	oranges	broccoli
1	apples	broccoli
3	oranges	
3	bananas	
3	apples	
4	apples	broccoli
4	oranges	broccoli
4	bananas	broccoli
4	apples	lettuce
4	oranges	lettuce
4	bananas	lettuce
4	apples	tomatoes
4	oranges	tomatoes
4	bananas	tomatoes
6	oranges	
6	bananas	
6	apples	

DATA Step Many-to-Many Left Join

DAY	FRUIT	VEGGIE
1	apples	broccoli
1	bananas	tomatoes
1	oranges	lettuce
3	apples	
3	bananas	
3	oranges	
4	apples	broccoli
4	bananas	tomatoes
4	oranges	lettuce
6	apples	
6	bananas	
6	oranges	

JOINS MANY-TO-MANY RIGHT JOIN

```
proc sql;  
  create table sql_m_to_m_right_join3 as  
    select v3.day as key, f3.fruit, v3.veggie  
    from fruits3 f3  
    right join  
    veggies3 v3  
    on f3.day = v3.day;  
run;
```

```
data data_m_to_m_right_join3;  
  merge fruits3 (in=f3) veggies3 (in=v3);  
  by day;  
  if v3;  
run;
```

HOW DO THE TECHNIQUES COMPARE?

MANY-to-MANY RIGHT JOIN

NO MATCH

SQL Many-to-Many Right Join

DAY	FRUIT	VEGGIE
1	bananas	tomatoes
1	bananas	lettuce
1	bananas	broccoli
1	oranges	tomatoes
1	oranges	lettuce
1	oranges	broccoli
1	apples	tomatoes
1	apples	lettuce
1	apples	broccoli
2		lettuce
2		tomatoes
2		broccoli
4	apples	broccoli
4	apples	lettuce
4	apples	tomatoes
4	oranges	broccoli
4	oranges	lettuce
4	oranges	tomatoes
4	bananas	broccoli
4	bananas	lettuce
4	bananas	tomatoes
5		lettuce
5		tomatoes
5		broccoli

DATA Step Many-to-Many Right Join

DAY	FRUIT	VEGGIE
1	apples	broccoli
1	bananas	tomatoes
1	oranges	lettuce
2		broccoli
2		tomatoes
2		lettuce
4	apples	broccoli
4	bananas	tomatoes
4	oranges	lettuce
5		broccoli
5		tomatoes
5		lettuce

JOINS MANY-TO-MANY FULL JOIN

```
proc sql;  
  create table sql_m_to_m_full_join4 as  
    select coalesce(f3.day,v3.day) as Key, f3.fruit, v3.veggie  
    from fruits3 f3 full join veggies3 v3  
    on f3.day=v3.day;  
run;
```

```
data data_m_to_m_full_join4;  
  merge fruits3 veggies3;  
  by day;  
run;
```

HOW DO THE TECHNIQUES COMPARE?

MANY-to-MANY FULL JOIN

NO MATCH

SQL Many-to-Many Full Join

Key	FRUIT	VEGGIE
1	bananas	tomatoes
1	bananas	lettuce
1	bananas	broccoli
1	oranges	tomatoes
1	oranges	lettuce
1	oranges	broccoli
1	apples	tomatoes
1	apples	lettuce
1	apples	broccoli
2		lettuce
2		tomatoes
2		broccoli
3	oranges	
3	bananas	
3	apples	
4	apples	broccoli
4	apples	lettuce
4	apples	tomatoes
4	oranges	broccoli
4	oranges	lettuce
4	oranges	tomatoes
4	bananas	broccoli
4	bananas	lettuce
4	bananas	tomatoes
5		lettuce
5		tomatoes
5		broccoli
6	oranges	
6	bananas	
6	apples	

DATA Step Many-to-Many Full Join

DAY	FRUIT	VEGGIE
1	apples	broccoli
1	bananas	tomatoes
1	oranges	lettuce
2		broccoli
2		tomatoes
2		lettuce
3	apples	
3	bananas	
3	oranges	
4	apples	broccoli
4	bananas	tomatoes
4	oranges	lettuce
5		broccoli
5		tomatoes
5		lettuce
6	apples	
6	bananas	
6	oranges	

SUMMARY

JOINING DATA

- SQL joins and DATA step merges may produce the same output for the following data:
 - One-to-One
 - One-to-Many
- SQL joins and DATA step merges produce dissimilar results when data represents a many to many structure.

ADDITIONAL COMPARISONS



CONDITIONAL PROCESSING

- IF THEN statement in the DATA step
 - Very flexible
- CASE expression in SQL


```
proc sql;  
  select name, case  
    when continent = 'North America' then 'US'  
    when continent = 'Oceania' then 'Pacific Islands'  
    else 'None'  
    end as region  
  from states;
```

```
data new;  
  set states;  
  if continent = 'North America'  
    then region = 'US';  
  else if continent = 'Oceania'  
    then region = 'Pacific Islands';  
  else region = 'None';  
run;
```

INDEXES

Indexes can be created by

- SQL
- DATA step (at data set creation)

Indexes may also be administered through SQL.

```
data health.test(index=(memberID)) ;  
    set health.claims_sample;  
run;
```

Indexes are created at the time of data set creation.
PROC DATASETS can be used to change or maintain
the indexes.

```
proc sql;  
    drop index providerId from health.test;  
    create unique index ProviderID on  
    health.provider(providerID);
```

PROC SQL can be used to create and administer indexes.

SUBSETTING

- Use the WHERE clause in PROC SQL to select only the rows of data that meet a certain condition.
- Use the WHERE statement or WHERE= option in the DATA step to select only the rows of data that meet a certain condition

SUBSETTING SQL AND DATA STEP

```
proc sql;  
  create table sql_subset as  
  select * from sashelp.cars  
  where make='Acura' and type='Sedan';  
quit;
```

```
data data_subset;  
  set sashelp.cars;  
  where make='Acura' and type='Sedan';  
run;
```

SORTING, SUMMARIZING AND CREATING NEW VARIABLES

- PROC SQL can sort, summarize, and create new variables in the same step
- The DATA step requires separate steps to accomplish sorting, summarizing, and creating new variables

SORTING,
SUMMARIZING,
CREATING NEW
VARIABLES

PROC SQL

```
proc sql;  
  title 'SQL - Total Reimbursement';  
  title2 'for Each Trial Phase in Test 1';  
  select phase, sum(reimbursement) as tot_reimbursement  
    label='Total Reimbursement' format=dollar15.  
    from o.datafile  
   where test_level='Test 1'  
   group by Phase  
   order by phase;  
quit;
```

SORTING,
SUMMARIZING,
CREATING NEW
VARIABLES

MULTIPLE STEPS

```
proc summary data=o.datafile;  
  where test_level='Test 1';  
  class phase;  
  var reimbursement;  
  output out=tot_reimburse sum=tot_reimbursement;  
run;  
  
proc sort data=tot_reimburse;  
  by phase;  
run;  
  
proc print data=tot_reimburse noobs label;  
  var phase tot_reimbursement;  
  format tot_reimbursement dollar15.;  
  label tot_reimbursement='Total Reimbursement';  
  title 'PROCS - Total Reimbursement';  
  title2 'for Each Trial Phase in Test 1';  
  where _type_=1;  
run;
```

CREATING MACRO VARIABLES

Macro variables can be created at execution time using:

- PROC SQL with the INTO clause
- CALL SYMPUTX data step routine

```
proc sql noprint;  
  select country, barrels  
    into :country1, :barrels1  
  from sql.oilrsrvs;
```

```
data _null_;  
    call symputx(' items ', ' text to assign');  
    call symputx(' x ', 123.456);  
run;
```

Both the DATA step and SQL can create macro variables at execution time.

The DATA step might be considered more flexible.

PRESENTING YOUR DATA

PROC SQL has the capability to produce basic line-oriented reports.

The DATA step provides maximum flexibility for creating highly customized reports.

The DATA step code below produces the block-oriented report

```
data _null_;  
  set heartsorted;  
  file print notitles header=head;  
  put @10 'Patient ID: ' pat_id  
      @30 'Gender:      ' sex /  
      @30 'Height:      ' height/  
      @30 'Weight:       ' weight/  
      @30 'Cholesterol:  ' cholesterol //;  
  
  return;  
head:  
  put @22 'Patient Information' //;  
  return;  
run;
```

Patient Information

Patient ID: 124	Gender:	Female
	Height:	62.25
	Weight:	132
	Cholesterol:	250
 Patient ID: 125	Gender:	Female
	Height:	65.75
	Weight:	158
	Cholesterol:	242
 Patient ID: 126	Gender:	Male
	Height:	66
	Weight:	156
	Cholesterol:	281

SQL:

- Provides the combined functionality of the DATA step and several base SAS procedures
- PROC SQL code may execute faster for smaller tables
- PROC SQL code is more portable for non-SAS programmers and non-SAS applications
- Processing does not require explicit code to presort tables
- Processing does not require common variable names to join on, although same type and length are required
- By default, a PROC SQL SELECT statement prints the resultant query; use the NOPRINT option to suppress this feature
- Efficiencies within specific RDBMS are available with Pass-thru code (connect to) for the performance of joins
- Use of aliases for shorthand code may make some coding tasks easier

*From **DATA Step vs. PROC SQL: What's a neophyte to do?** Proceedings of 29th SAS User Group International Conference, by Craig Dickstein, Tamarack Professional Services, Jackman, ME and Ray Pass, Ray Pass Consulting, Hartsdale, NY

NON-SQL BASE SAS:

- DATA step set operators can handle more data sets at a time than PROC SQL outer joins
- Non-SQL techniques can open files for read and write at the same time
- Customized DATA step report writing techniques (DATA _NULL_) are more versatile than using PROC SQL SELECT clauses
- The straightforward access to RDBMS tables as if they were SAS data sets negates the need to learn SQL constructs
- Input of non-RDBMS external sources is easier

*From **DATA Step vs. PROC SQL: What's a neophyte to do?** Proceedings of 29th SAS User Group International Conference, by Craig Dickstein, Tamarack Professional Services, Jackman, ME and Ray Pass, Ray Pass Consulting, Hartsdale, NY

- Which technique more efficiently handles the task at hand?
- Which technique allows you to be more collaborative with your peers and coworkers?
- Which technique is easier to maintain?
- Which technique are you more familiar with?

RESOURCES



SUPPORT.SAS.COM RESOURCES

Base SAS documentation

<http://support.sas.com/documentation/onlinedoc/base/index.html>

SAS Training

<http://support.sas.com/training/>

RSS & Blogs

<http://support.sas.com/community/rss/index.html>

<http://blogs.sas.com>

Discussion Forums

<http://communities.sas.com/index.jspa>

SAS® 9.4 SQL Procedure User's Guide

<http://support.sas.com/documentation/cdl/en/sqlproc/65065/HTML/default/viewer.htm#titlepage.htm>

Papers & SAS Notes

<http://support.sas.com/resources/papers/sgf09/336-2009.pdf>
<http://support.sas.com/kb/20/783.html>

RESOURCES

PAPERS & BLOG

- [Yes, Proc SQL is Great, But I Love Data Step Programming, What's a SAS User To Do?](#) Mira J. Shapiro, MJS Consultants, Bethesda, MD
- [Proc SQL versus The Data Step](#) JoAnn Matthews, Highmark Blue Shield, Pittsburgh, PA
- [DATA Step versus PROC SQL Programming Techniques](#) Kirk Paul Lafler, Software Intelligence Corporation
- [DATA Step vs. PROC SQL: What's a neophyte to do?](#) Craig Dickstein, Tamarack Professional Services, Jackman, ME Ray Pass, Ray Pass Consulting, Hartsdale, NY

THANK YOU!



THE
POWER
TO KNOW[®]