

Scaling up for the SQL payoff





Averaging

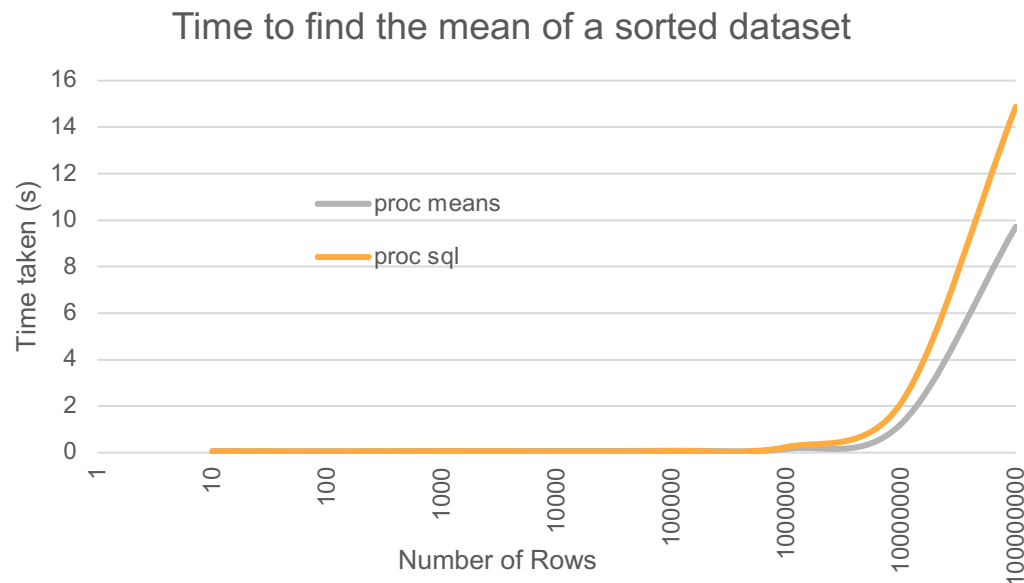
One may assume proc means would be more efficient than proc sql, and while that is true for simple cases, there are many instances when proc sql gives the upper hand.

Finding the class means from a sorted dataset

Proc sql is no match for proc means when finding the mean of a string of values in a sorted dataset...

```
proc sql;  
  create table dsout as  
  select studyid, age)  
  from dsin_sorted  
  group by studyid;  
quit;
```

SQL: **x**

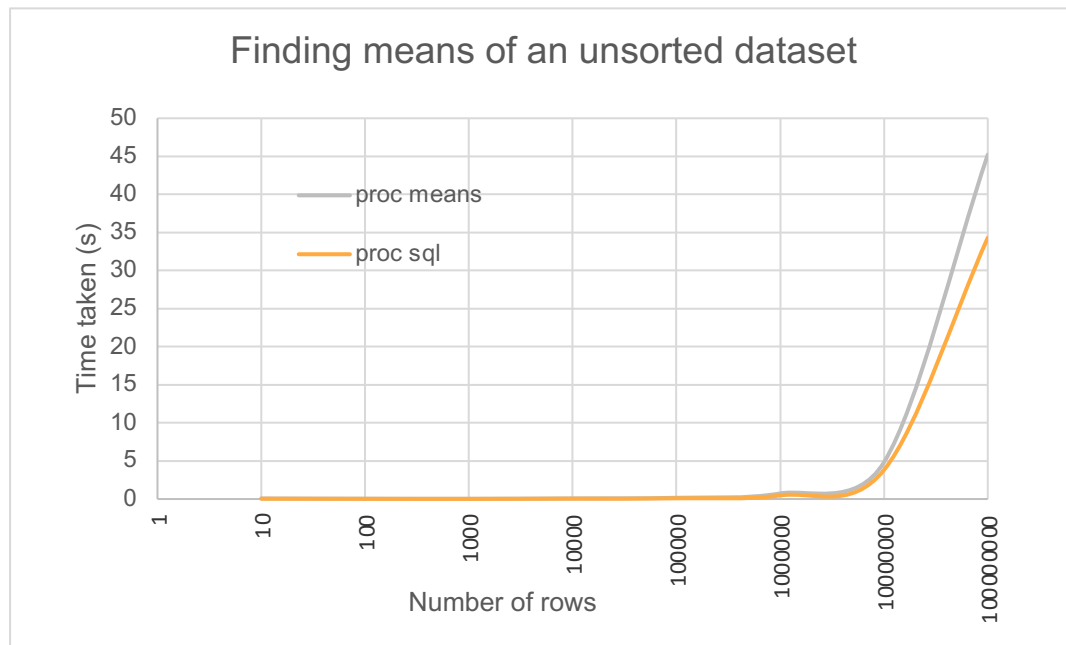


Finding the class means from an unsorted dataset

However, when working with unsorted datasets, proc means is no match for a single proc sql on large datasets, irrespective of whether using a class statement or a separate proc sort.

```
proc sql;  
  create table dsout as  
  select studyid, age)  
  from dsin_unsorted  
  group by studyid;  
quit;
```

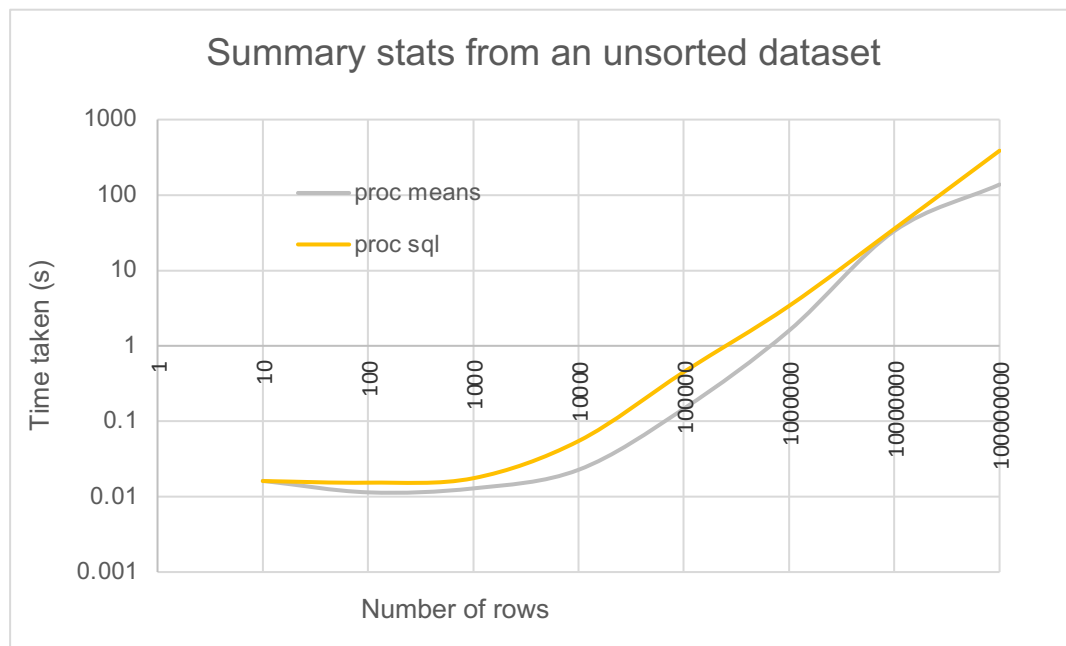
SQL: ✓



Finding the summary statistics of an unsorted dataset

When adding min, max, median, stddev, and n into the mix, proc means shows its advantage.

```
proc sql;  
  create table dsout as  
  select distinct studyid,  
    count(age) as n,  
    mean(age) as mean,  
    std(age) as stddev,  
    median(age) as median,  
    min(age) as min,  
    max(age) as max,  
  from dsin_unsorted  
  group by studyid;  
quit;
```



SQL: **x**

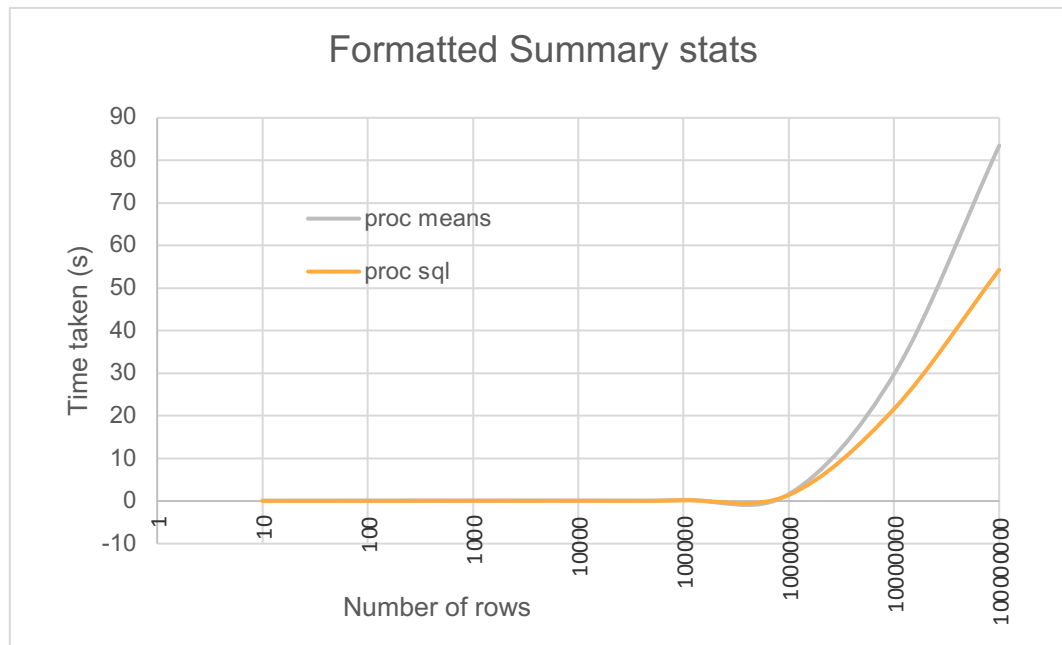
Finding and formatting summary stats of unsorted data

When formatting is needed, proc sql does not need an extra data step:

```
proc sql;
  create table dsout as
  select distinct studyid,
    put(count(age),8.) as n,
    put(round(mean(age),0.01),5.2)||" ("||put(round(std(age),0.001),5.3)||")" as mean_std,
    put(round(median(age),0.01),5.2) as median,
    put(round(min(age),0.1),4.1)||" to "||put(round(max(age),0.1),4.1) as min_max
  from dsin_unsorted
  group by studyid;
quit;
```

Finding and formatting summary stats of unsorted data

When formatting is needed, proc sql does not need an extra data step:



SQL: ✓

Averages and counts as new parameters

SQL offers many benefits when creating new parameters, especially when involving counts and means.

USUBJID	AVISIT	PARAMCD	ADY	AVAL
ABC001	Week 1	SEVERITY	1	20
ABC001	Week 1	SEVERITY	4	35
ABC001	Week 1	SEVERITY	5	25
ABC001	Week 1	MEAN		26.6667
ABC001	Week 1	COUNT		3
ABC001	Week 2	SEVERITY	8	30
ABC001	Week 2	SEVERITY	9	35
ABC001	Week 2	SEVERITY	11	40
ABC001	Week 2	SEVERITY	12	45
ABC001	Week 2	MEAN		37.5
ABC001	Week 2	COUNT		4

Averages and Counts as new parameters

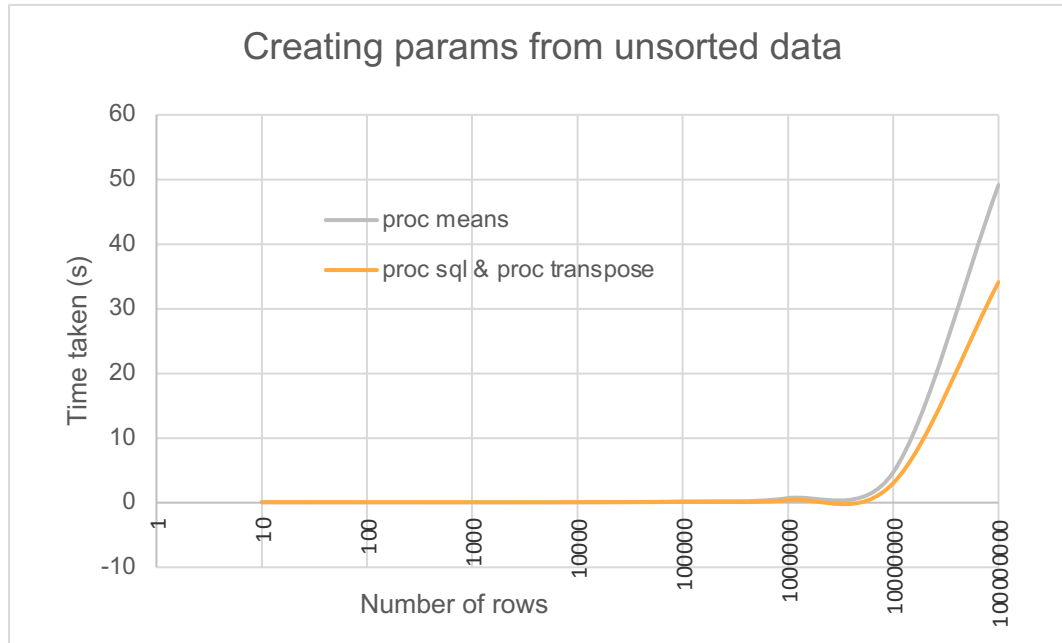
While SQL can create each param on a separate row using union, it runs more quickly if we create each param/aval pair as a new variable and then transpose:

```
proc sql;
  create table dsin_sql as
  select subjid, trtan, trta, avisitn, avisit, ady,
         mean(aval) as mean,
         count(aval) as count
  from dsin
  group by subjid, trtan, avisitn, ady;
quit;

proc transpose data=dsin_sql
               out=dsout (rename=(_name_=paramcd coll=aval));
  by subjid trtan trta avisitn avisit ady;
  var mean count;
run;
```

Averages and Counts as new parameters

Again, when using unsorted data SQL outperforms proc means.



SQL: ✓

Minimums & maximums

Even on sorted data, proc sql outperforms proc means is when deriving a maximum or minimum row.

USUBJID	AVISIT	ADY	AVAL
ABC001	Week 1	7	20
ABC001	Week 2	14	35
ABC001	Week 3	21	25
ABC001	Week 4	28	30
ABC001	Maximum	14	35
ABC002	Week 1	7	30
ABC002	Week 2	14	35
ABC002	Week 3	21	40
ABC002	Week 4	28	45
ABC002	Maximum	28	45

Maximum and Minimum rows

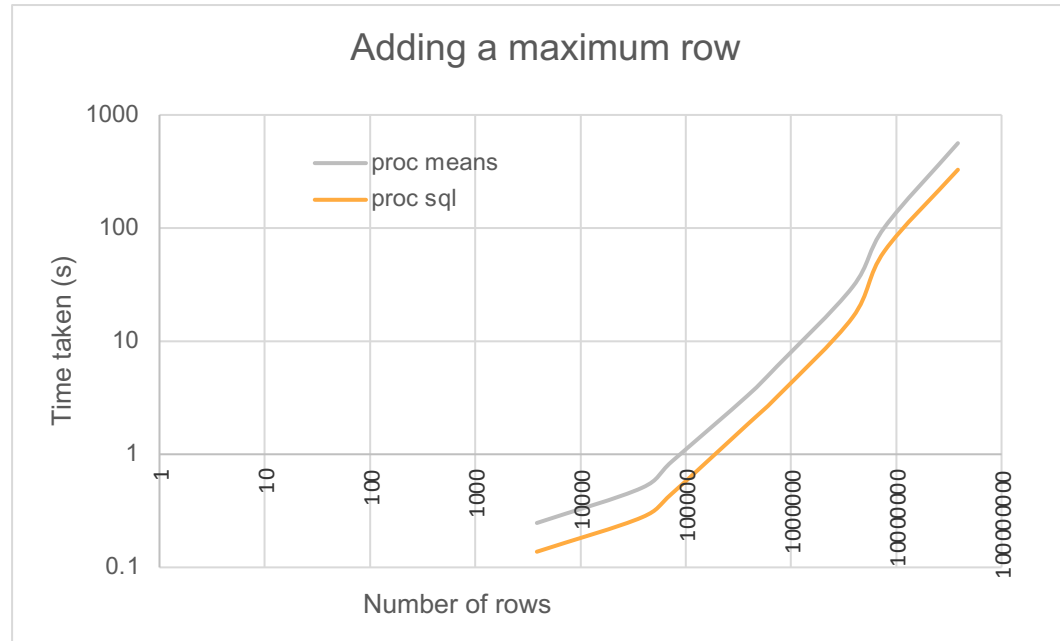
This is because SQL can keep variables not included in the group by statement, for example the date on which the maximum value occurred.

Proc means cannot, and will need to be merged on to the original data to populate these.

```
proc sql;
  create table dsin_2 as
  select distinct subjid, paramcd, adt, ady, aval, trtan, trta, 7777 as avisitn, "Maximum" as avisit length=50
  from dsin
  group by subjid, paramcd
  having aval=max(aval)
  order by subjid, paramcd, adt;
quit;

proc sort data=dsin_2 out=dsout nodupkey;
  by subjid paramcd;
run;
```

Maximum and Minimum rows



SQL: ✓

Adding an average row

Sometimes we'll want to find an average only if there are multiple observations on the same day or visit.

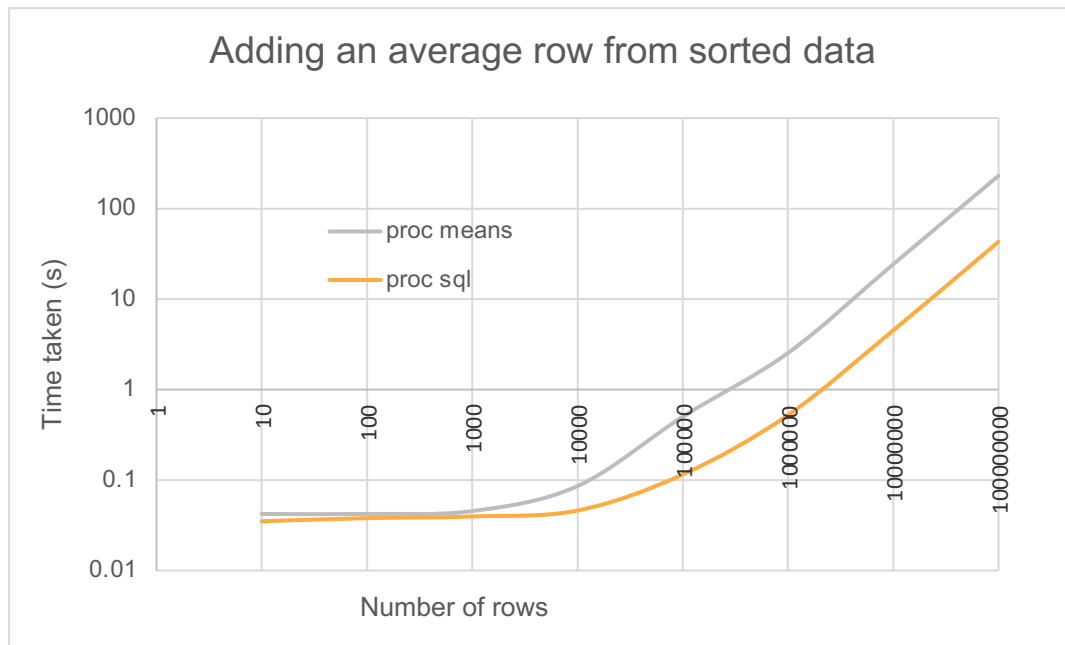
USUBJID	ADY	AVAL	DTYPE
ABC001	1	20	
ABC001	2	35	
ABC001	3	25	
ABC001	3	30	
ABC001	3	27.5	AVERAGE
ABC001	4	30	
ABC001	5	35	
ABC001	6	40	
ABC001	6	45	
ABC001	6	42.5	AVERAGE

Adding an average row

In order to keep only results with multiple observations, proc means must find all results and then filter the final dataset, unless we use another procedure to find when we need an average. SQL can filter earlier, so only produces results when there are multiple observations.

```
proc sql;  
  create table dsout as  
  select distinct usubjid, paramcd, adt, ady,  
    mean(aval) as aval,  
    count(aval) as count,  
    "AVERAGE" as dtype length=8  
  from dsin  
  group by usubjid, paramcd, adt  
  having count>1;  
quit;
```

Adding an average row



SQL: ✓

Consolidation of flags



USUBJID	VAR1FL	VAR2FL
ABC001	Y	Y
ABC002		Y
ABC003	Y	

If you want to find out if a subject ever has a variable flagged as Y, there's a really simple way with SQL.

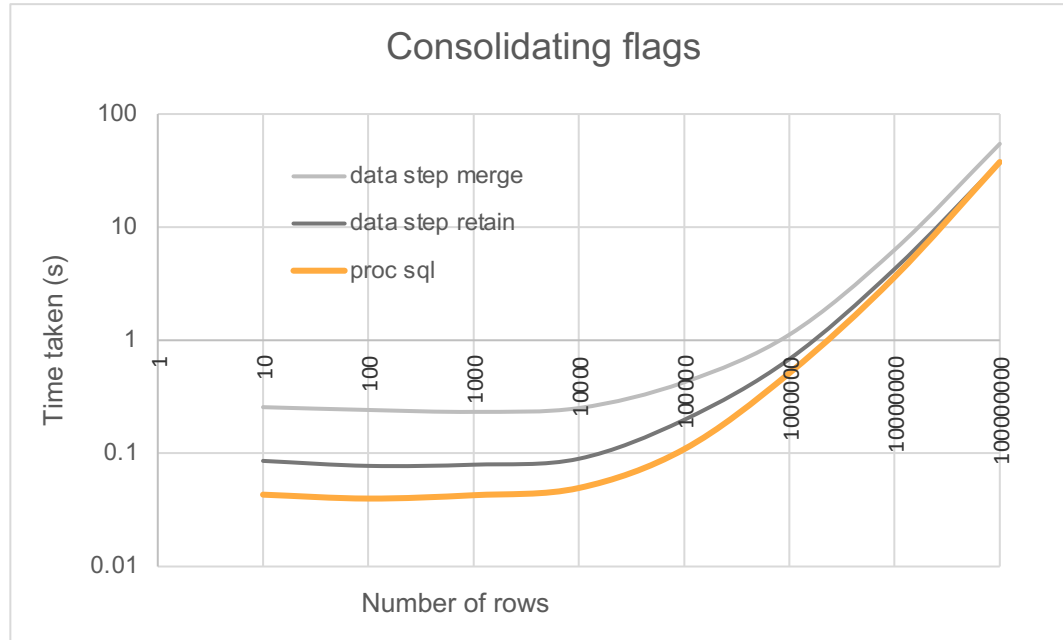
USUBJID	ADY	VAR1FL	VAR2FL
ABC001	1		Y
ABC001	2	Y	
ABC001	3	Y	
ABC001	4		
ABC002	1		
ABC002	2		Y
ABC002	3		
ABC002	4		
ABC003	1	Y	
ABC003	2		
ABC003	3		
ABC003	4		

Consolidation of flags

This uses the principle of finding the maximum value of VARxFL per subject, but proc means can only find the maximum of a numeric variable.

```
proc sql;  
  create table dsout as  
  select distinct usubjid,  
                 max(var1fl) as var1fl,  
                 max(var2fl) as var2fl  
  from dsin  
  group by usubjid;  
quit;
```

Consolidation of flags



SQL: ✓



Merging

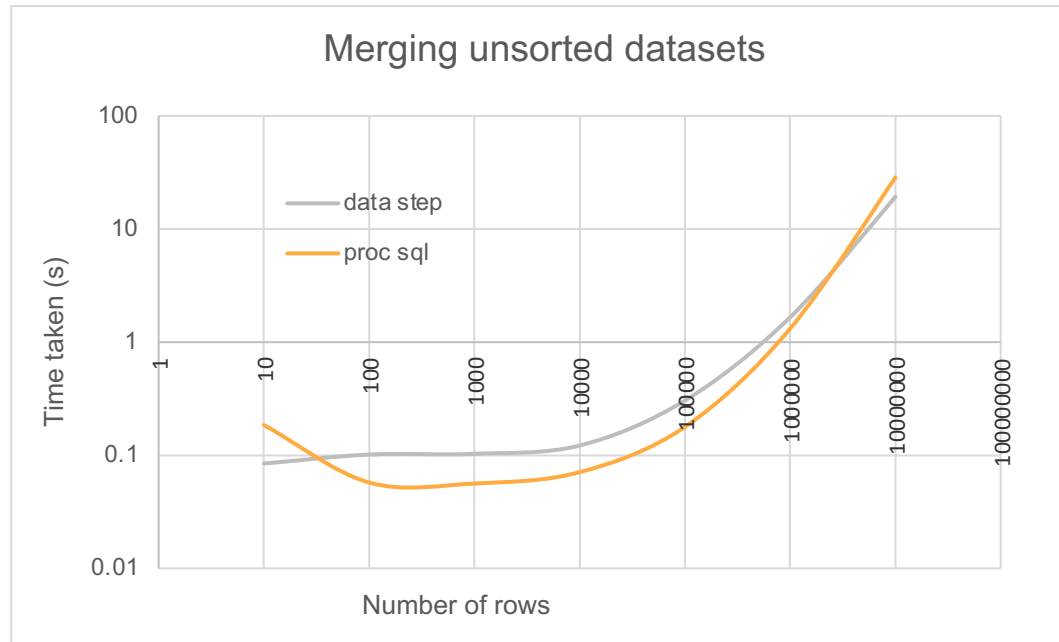
SQL merges are sometimes reserved only for many-to-many merges which cannot be done in a data step, but we shall see that it can also be quicker for inner joins.

SQL will use a different method for merging depending on the datasets involved, and the memory constraints of the system involved.

Inner merge of unsorted datasets

Proc SQL inner joins are quicker than a data step merge up to a point.

If both datasets are the same size, a data step becomes more efficient once the datasets are too big to be stored in memory.



SQL: **x**

Merging and memory

When the smaller of the two datasets fits in memory, it uses the hash method, which is faster than a standard dataset merge.

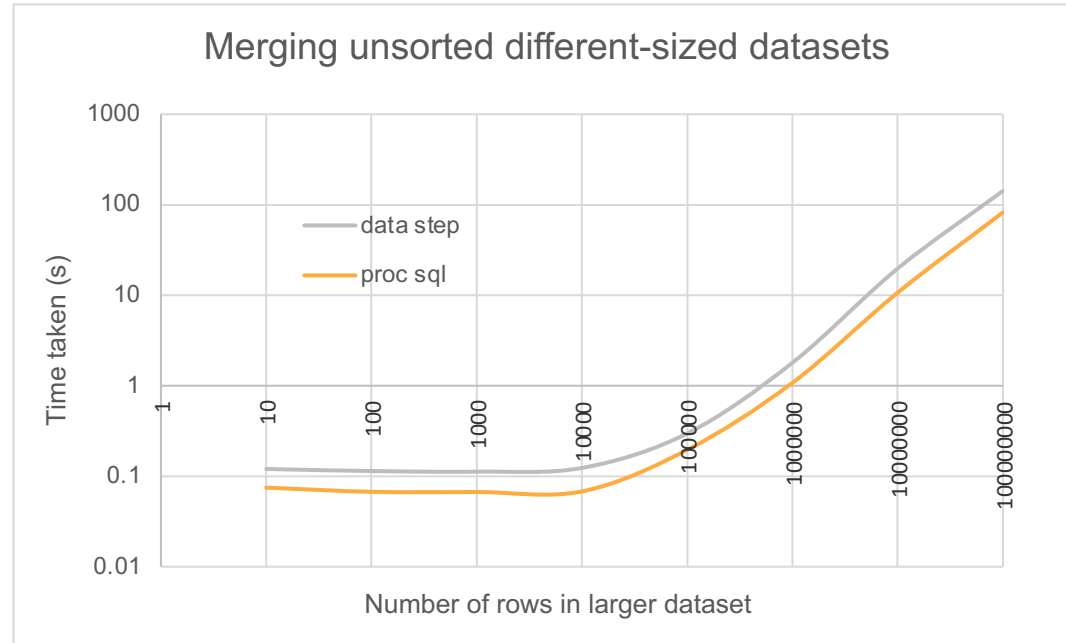
```
proc sql _method;  
  create table dsout as  
  select a.*, b.var2  
  from ds1 as a, ds2 as b  
  where a.ind=b.ind;  
quit;
```

NOTE: SQL execution methods chosen are:

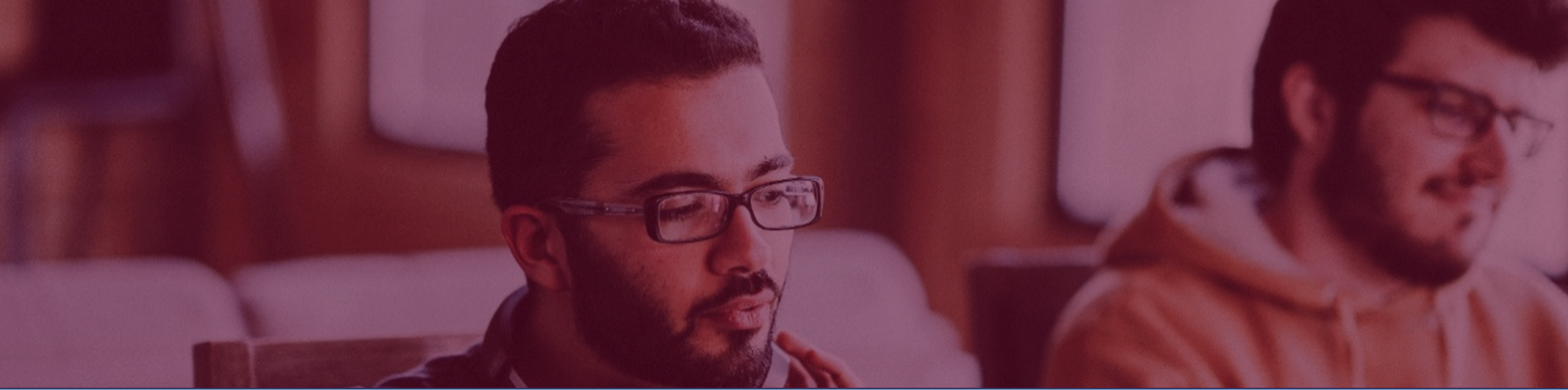
```
sqxcrt  
  sqxjsh  
    sqxsrc( WORK.DS1(alias = A) )  
    sqxsrc( WORK.DS2(alias = B) )
```

Inner Merge of unsorted datasets of different sizes

However, if the datasets are different sizes, only the smaller of the two needs to fit in memory for SQL to be more efficient.



SQL: ✓



Thank you