

# Proc SQL, the Data Step Killer

Mike Atkinson  
Acko Systems Consulting Inc



# The Data Step

- It's a complete programming language
- Easy to combine (or merge) datasets
- It allows you to perform sequential algorithm steps
  - Some algorithms require data to be in sorted order first
  - Can use values of variables from different observations, using retain
- It has the “where” statement (stolen from SQL) that allows efficient use of filtering criteria prior to other processing
- There are some things the data step can do that can't be done in Proc SQL, e.g.
  - Can create multiple datasets in one step
  - Can easily join multiple datasets – each left, right, or full outer join in a Proc SQL query can join only two datasets at a time (although inner joins without the join keyword can bring together any number)

# Proc SQL

- SQL is the de facto standard query language, widely used (beyond SAS even!) for retrieving and summarizing data
- Proc SQL can summarize results in the same step as performing row level calculations
  - without Proc SQL, summarizing requires a separate proc summary step, and often a pre-sort
- Proc SQL can sort its results in the same step
- It can perform distinct counts
- It can use “like” expressions
- While SQL isn’t a complete programming language, it has “case” expressions, which can be very powerful

# Some stuff SAS Proc SQL can do

- Sending (pass-through) queries to Oracle (or another DBMS) for processing, and receiving the results into a SAS dataset
- Administration tasks, such as managing SAS datasets and indexes
- Using the SQL language against SAS datasets as an alternative to the Data Step
- Setting values of macro variables
- As an alternative to Proc Print

# Data step

Basic syntax:

```
data new_SAS_dataset;  
  set some_existing_dataset;  
  /* set some_existing_dataset (keep=column_1  
    column_2); to subset variables */  
  * do stuff;  
run;
```

# Proc SQL Create Table

Basic syntax:

```
proc sql;
  create table new_SAS_dataset as
  /* select * for all columns/variables */
  select column_1,
         column_2
  from some_existing_dataset;
quit;
```

- Although it says create *table*, it is actually creating a SAS dataset.
- PROC SQL terminates with a `quit;` statement (not `run;`).

# WHERE clause

Can be used in data step statement and within Proc SQL, including within a “case” expression

Comparison operators

`<, >, =, <=, >=, ^=` or `lt, gt, eq, le, ge, ne`

Logic operators, brackets

`and, or, not` e.g. `((a > b) and (not (c = d)))`

IN operator

`in (values, separated, by, commas)`

# WHERE clause “like” & “contains”

While the “in (list)” has made its way to the IF statement, “like” (and “contains”) have not; they are only in the WHERE

The syntax for CONTAINS is straightforward, e.g.

```
where name contains 'JONES'
```

But I prefer LIKE, which is more powerful

percent sign (%) – match zero or more characters

underscore (\_) – match any one character

e.g. 

```
where name like 'JONES%'
```

# ORDER BY clause

Not only does PROC SQL not require data to be sorted beforehand, but you can ask it to sort its resulting output simply by adding an ORDER BY clause

The ORDER BY clause appears last, after the GROUP BY clause and the HAVING clause, if those are present

The ORDER BY clause can be used on his own, without grouping

The syntax of the ORDER BY clause is slightly different than the Data Step (and other Procs') BY statements; the BY statement separates variables by spaces, while the ORDER BY separates them using commas.

# GROUP BY clause

The GROUP BY clause in Proc SQL lets you summarise data (similar to Proc Summary) but without requiring the data to be sorted beforehand.

The GROUP BY clause (if present) follows the WHERE clause

Variables in the GROUP BY are separated by commas

Using a GROUP BY statement does not guarantee the sort order of the results (although SAS is more likely to put the data into that order than Oracle is). Use an ORDER BY with a GROUP BY if you need to ensure the sort order of the data.

Note: the variables in the ORDER BY clause need not match the variables in the GROUP BY clause.

# Summary functions



If you have a GROUP BY in your query, then every variable you select should either be listed in the GROUP BY, or be summarised in some way. If you select a variable that is not summarised and not listed in the GROUP BY clause, you will almost certainly not get the summarized results you expect.

Here are some sample summary functions:

```
sum (services)           as services,  
max (service_date)      as max_servdt,  
mean (paid_amount)      as avg_paidamt,  
count (distinct PHN)    as patient_count
```



# HAVING Claus



The HAVING clause applies after the GROUP BY, WHEREas the WHERE clause applies before grouping

The HAVING clause looks at summarised values, and cannot be used without a GROUP BY clause

e.g.

```
proc sql;
    create table three_or_more as
    select service_date,
           count(*)          as record_count
    group by service_date
    having count(*) >= 3;
quit;
```

# CASE expression

This is PROC SQL's closest equivalent to the IF statement. A CASE expression, however, can only return a single value. (an IF statement can use a do/end to to perform multiple actions)

The CASE expression consists of a series of WHEN conditions (that use the same syntax as WHERE conditions), followed by ELSE. So it's really more like an IF THEN/ELSE.

Each WHEN condition is accompanied by a THEN expression that evaluates to a value.

The CASE expression will use the THEN expression of the first WHEN condition that is found to be True. If none of the WHEN conditions are true, the ELSE expression will be used.

It's good practice to always have an ELSE.

# CASE expression example

```
proc sql;
  select case when age = 0                then ' 0  '
            when age between 1 and 5    then ' 1- 5'
            when age between 6 and 10  then ' 6-10'
            when age between 11 and 15 then '11-15'
            ...
            else '?????' end            as age_group,
         count(distinct recipient_id)  as person_cnt
  from health_services
  group by calculated age_group;
quit;
```

# Aliases

When joining two or more tables, it is useful to use an alias for each table.

The alias can be used as a prefix to variable names to indicate which table the variable comes from, which is handier than using the whole table name as a prefix.

When a variable of the same name appears in more than one table (being joined using a Proc SQL select statement), you must specify which table you want to refer to each time you refer to the variable name. Prefixing variables with the table alias is the usual way to do this.



# LEFT JOIN, RIGHT JOIN

The default SQL join is an Inner Join, meaning that only rows that match across both tables are included

LEFT JOIN and RIGHT JOIN in Proc SQL always operate on exactly two tables, and the order the tables are listed is very significant. Imagine writing them on the same line – the first dataset listed is the Left one, and the second is the Right dataset.

When you use LEFT JOIN or RIGHT JOIN, you use the ON keyword (instead of the WHERE keyword) to indicate the join criteria.

If you use the INNER JOIN syntax to perform an inner join, you will also need to use the ON keyword

# Comparing Inner, Left, and Right joins

Here's some sample data in two datasets.

## ***Students***

<b>Student_ID</b>	<b>Name</b>
34	Gray, Jane
56	Adams, Giselle
78	Keppel, Len

## ***Grades***

<b>Student_ID</b>	<b>Subject</b>	<b>Grade</b>
34	Math	A
34	English	B
56	Math	C+
99	French	F

# Inner Join (usual, without JOIN keyword)

```
proc sql;
  create table after_inner as
  select a.*,
         b.*
  from students  a,
       grades    b
  where a.student_id = b.student_id
  order by a.student_id;
quit;
```



**Note:** This will give a note in the log that `student_id` already exists in the dataset. Because `student_id` is the same in both datasets (guaranteed by the `WHERE` condition), this note can be safely ignored.

Okay, here's how you could rid of the note  
(without listing all the columns you want)

```
proc sql;
  create table after_inner (drop=student_id2) as
  select a.*,
         b.*
  from students a,
       grades
         (rename=(student_id=student_id2)) b
  where a.student_id = b.student_id2
  order by a.student_id;
quit;
```

It's probably easier just to ignore the note in the log.

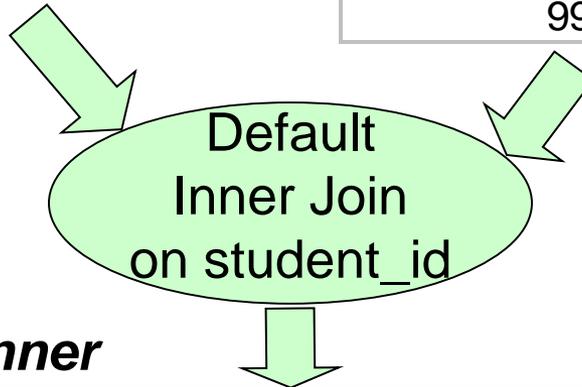
# Results of (default) Inner Join

## *Students*

Student_ID	Name
34	Gray, Jane
56	Adams, Giselle
78	Keppel, Len

## *Grades*

Student_ID	Subject	Grade
34	Math	A
34	English	B
56	Math	C+
99	French	F



## *After\_Inner*

Student_ID	Name	Subject	Grade
34	Gray, Jane	Math	A
34	Gray, Jane	English	B
56	Adams, Giselle	Math	C+

# LEFT Join

```
proc sql;
    create table after_left as
    select a.*,
           b.*
    from students    a left join
         grades      b
         on a.student_id = b.student_id
    order by a.student_id;
quit;
```

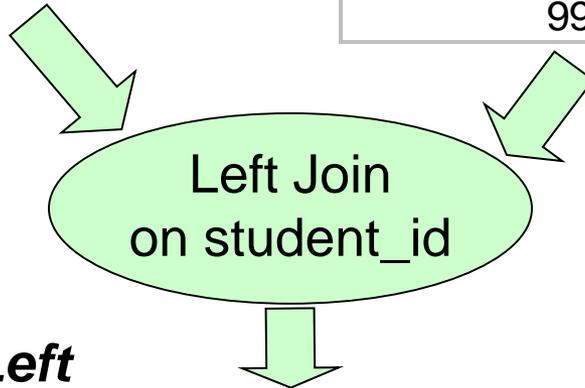
# Results of Left Join

## *Students*

Student_ID	Name
34	Gray, Jane
56	Adams, Giselle
78	Keppel, Len

## *Grades*

Student_ID	Subject	Grade
34	Math	A
34	English	B
56	Math	C+
99	French	F



## *After\_Left*

Student_ID	Name	Subject	Grade
34	Gray, Jane	Math	A
34	Gray, Jane	English	B
56	Adams, Giselle	Math	C+
78	Keppel, Len		

# RIGHT join

```
proc sql;
    create table after_right as
    select a.*,
           b.*
    from students    a    right join
         grades      b
         on a.student_id = b.student_id
    order by a.student_id;
quit;
```

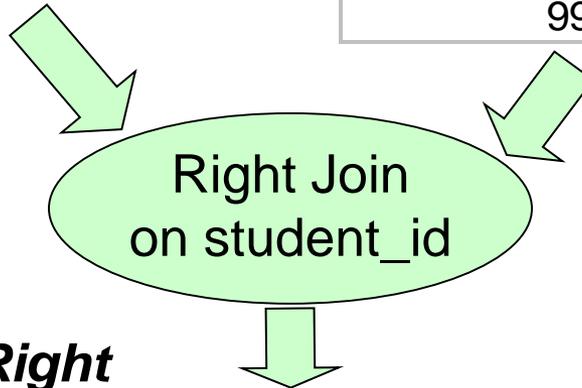
# Results of Right Join

## *Students*

Student_ID	Name
34	Gray, Jane
56	Adams, Giselle
78	Keppel, Len

## *Grades*

Student_ID	Subject	Grade
34	Math	A
34	English	B
56	Math	C+
99	French	F



## *After\_Right*

Student_ID	Name	Subject	Grade
34	Gray, Jane	Math	A
34	Gray, Jane	English	B
56	Adams, Giselle	Math	C+
		French	F

# FULL (Outer) join

```
proc sql;
  create table after_full as
  select coalesce(a.student_id, b.student_id) as
         student_id,
         a.name,
         b.subject,
         b.grade
  from students  a full join
         grades  b
  on a.student_id = b.student_id
  order by a.student_id;
quit;
```

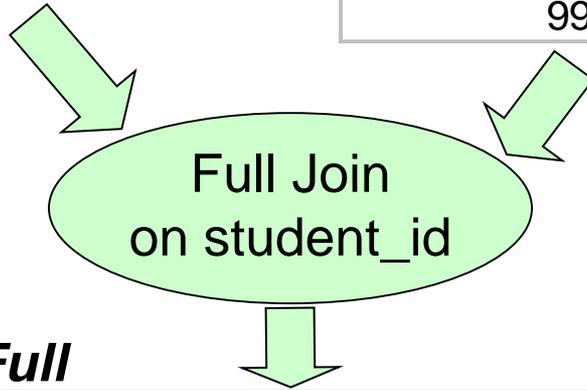
# Results of Full (Outer) Join

## *Students*

Student_ID	Name
34	Gray, Jane
56	Adams, Giselle
78	Keppel, Len

## *Grades*

Student_ID	Subject	Grade
34	Math	A
34	English	B
56	Math	C+
99	French	F



## *After\_Full*

Student_ID	Name	Subject	Grade
34	Gray, Jane	Math	A
34	Gray, Jane	English	B
56	Adams, Giselle	Math	C+
78	Keppel, Len		
99		French	F

# Traditional SAS Code

## (Data Step needs helpers!)

```
proc sort data=prac_info;  
  by prac_lha;  
run;
```

```
proc summary data=prac_info;  
  by prac_lha;  
  output out=prac_lha_counts  
    (drop=_type_ rename=( _freq_ =prac_cnt) );  
run;
```

# Proc SQL doing a “summary”

```
proc sql;  
  create table prac_lha_counts as  
  select prac_lha,  
         count(*) as prac_cnt  
  from prac_info  
  group by prac_lha  
  order by prac_lha;  
quit;
```

# Calculated keyword in Proc SQL

The keyword “calculated” can be used to refer to a column being created within a Proc SQL query by name, in a reference later within the same query.

It can be used to reference a calculated column within the GROUP BY expression, or even in expressions to create other columns.

There is no abbreviation for “calculated”.

# Traditional SAS Code

## summarize and lookup a description

```
proc sort data=fitm_servcd;  
  by servcd;  
run;
```

```
proc summary data=fitm_servcd;  
  by servcd;  
  output out=servcd_fitm_cnts_0  
         (drop=_type_ rename=(_freq_=fitm_cnt));  
run;
```

```
data servcd_fitm_cnts;  
  set servcd_fitm_cnts_0;  
  servcd_descrip = put(servcd, svcd2ds.);  
run;
```

# Proc SQL Code

```
proc sql;  
  create table servcd_fitm_cnts as  
  select servcd,  
         put(servcd, svcd2ds.) as servcd_descrip,  
         count(*)             as fitm_cnt  
  from fitm_servcd  
  group by servcd, calculated servcd_descrip  
  order by servcd;  
quit;
```

# Partial results...

SERVCD\_FITM\_CNTS ▾

  Filter and Sort  Query Builder | Data ▾ Describe ▾ Graph ▾ *A*

	 SERVCD	 servcd_descrip	 fitm_cnt
1	1	REGIONAL EXAM...	124
2	2	CONSULTATION...	28
3	3	COMPLETE EXA...	20
4	4	COUNSELLING (...)	25
5	5	HOME VISITS	13
6	6	EMERGENCY VI...	27
7	7	INSTITUTIONAL...	30
8	8	MISCELLANEOU...	32
9	9	VISIT PREMIUMS...	33
10	11	PROLONGED OR...	6

# Proc SQL Code with join

```
proc sql;  
  create table servcd_fitm_cnts as  
  select a.servcd,  
         b.servcd_descrip,  
         count(*)           as fitm_cnt  
  from fitm_servcd      a  left join  
         service_codes  b  
     on a.servcd = b.servcd  
  group by a.servcd, b.servcd_descrip  
  order by 1, 2;  
quit;
```

# Select desired observations using a Data Step

```
%let startdt_sas = '01apr2012'd;
%let enddt_sas   = '31mar2013'd;

data data_centres_2;
  set data_centres;
  where efctvdt      <= &enddt_sas
        and cncltdt  >= &startdt_sas
        and dt_cntr_status in ('D', 'P')
        and dt_cntr_type   in ('C', 'P')
        and not ( ' ' || dt_cntr_nm || ' ' like '% HOLDINGS %' or
                  ' ' || dt_cntr_nm || ' ' like '% HOSP%' or
                  ' ' || dt_cntr_nm || ' ' like '%SYS%' );

run;
```

# Anything you can do...

(well, not *anything*, but this thing...)

```
proc sql;
  create table data_centres_with_flags as
  select efctvdt,
         cncldt,
         dt_cntr_status,
         dt_cntr_type,
         case when (efctvdt > &enddt_sas) or
                  (cncldt < &startdt_sas)
              then '1. Outside date range'
         when (dt_cntr_status not in ('D', 'P'))
              then '2. Status not D or P'
         when (dt_cntr_type not in ('C', 'P'))
              then '3. Type not C or P'
         when (' ' || dt_cntr_nm || ' ' like '% HOLDINGS %'
              or ' ' || dt_cntr_nm || ' ' like '% HOSP%'
              or ' ' || dt_cntr_nm || ' ' like '%SYS%' )
              then '4. Computing type'
         else ' ' end
         as error_type
  from data_centres;
quit;
```

# Informative report

```
proc freq data=data_centres_with_flags;  
  tables error_type / missing;  
run;
```

## The FREQ Procedure

error_type	Frequency	Percent	Cumulative Frequency	Cumulative Percent
	2888	57.76	2888	57.76
1. Outside date range	1905	38.10	4793	95.86
2. Status not D or P	48	0.96	4841	96.82
3. Type not C or P	151	3.02	4992	99.84
4. Computing type	8	0.16	5000	100.00

Page Break

# Getting the goods, either way

```
proc sql;  
  create table data_centres_2 as  
  select *  
  from data_centres_with_flags  
  where error_type is null;  
quit;
```

or

```
data data_centres_2;  
  set data_centres_with_flags;  
  where error_type is null;  
run;
```

# Distinct keyword

If “distinct” appears as in “select distinct”, it applies to all selected columns, and is basically the same as using PROC SORT with NODUP. e.g.

```
select distinct provider, patient,  
              service_date
```

Distinct can also appear within a count summary function. e.g.

```
count(distinct provider) as uniq_prac_cnt,  
count(*)                 as record_cnt,  
count(provider)         as cnt_recs_w_provider
```

# Demonstration of calculated

```
proc sql;
  create table attached_w_age_range as
  select *,
    floor(yrdif(datepart(birth_date),
      '31mar2013'd, 'AGE')) as age,
    5 * (floor(calculated age/5)) as age_temp,
    case when (calculated age) = 0 then '000'
      else put(calculated age_temp, z3.)
      || '-' ||
      put(calculated age_temp + 4, z3.)
    end as age_range
  from attached_2012_2013
  order by res_at_yr_end;
quit;
```

# Some sample data...

```
data specialty_claims;  
  infile cards4;  
  input specialty clntage paidamt;  
cards4;  
00  5    5000  
00 10   10000  
00 20   10000  
00 30   10000  
00 40   15000  
00 50   25000  
00 60   35000  
00 70   55000  
00 80   75000  
00 90   85000  
01 10   15000  
01 20   15000  
;;;  
run;
```

# Get percent costs for patients (clients) aged 65 or over

```
proc sql;
  create table pct_over_65 as
  select specialty,
         sum(paidamt)                as paidamt,
         sum(case when clntage >= 65 then paidamt
              else 0 end)           as paidamt_ge65,
         (calculated paidamt_ge65) / (calculated paidamt)
         as pct_paid_over_65
         format=percent7.1
  from specialty_claims
  group by specialty;
quit;
```

# Results – percent costs for patients aged 65 or over

The screenshot shows a software interface with a menu bar at the top containing 'Program', 'Log', and 'Output Data (2)'. Below the menu bar is a dropdown menu labeled 'PCT\_OVER\_65'. A toolbar below the dropdown includes icons for 'Filter and Sort', 'Query Builder', 'Data', 'Describe', 'Graph', 'Analyze', and 'Exp'. The main area displays a table with the following data:

	specialty	paidamt	paidamt_ge65	pct_paid_over_65
1	0	325000	215000	66.2%
2	1	30000	0	0.0%

# Practitioner, service code data

```
data prac_servcd;  
  infile cards4;  
  input pracnum servcd paidamt;  
cards4;  
00001 01 50000  
00001 12 10000  
00001 91 20000  
00002 01 45000  
00002 90 8000  
00003 01 60000  
00003 12 5000  
00003 92 30000  
;;;  
run;
```

# Flag practitioners with all three

```
proc sql;
  create table prac_3 as
  select pracnum,
         sum(case when servcd = 01 then paidamt else 0 end)
           as paidamt_01,
         sum(case when servcd = 12 then paidamt else 0 end)
           as paidamt_12,
         sum(case when servcd >= 89 then paidamt else 0 end)
           as paidamt_89_plus,
         case when (calculated paidamt_01) > 0 and
                   (calculated paidamt_12) > 0 and
                   (calculated paidamt_89_plus) > 0      then 1
                else 0 end          as all_3
  from prac_servcd
  group by pracnum;
quit;
```

# Practitioners with flag for all 3

PRAC\_3 ▾

 |  Filter and Sort |  Query Builder | Data ▾ | Describe ▾ | Graph ▾ | Analyze ▾ | Export ▾ | Send To

	 pracnum	 paidamt_01	 paidamt_12	 paidamt_89_plus	 all_3
1	1	50000	10000	20000	1
2	2	45000	0	8000	0
3	3	60000	5000	30000	1

# Subset to those with all 3

```
proc sql;
  create table prac_all_3 as
  select *
  from
  (
    select pracnum,
      sum(case when servcd = 01 then paidamt else 0 end)
        as paidamt_01,
      sum(case when servcd = 12 then paidamt else 0 end)
        as paidamt_12,
      sum(case when servcd >= 89 then paidamt else 0 end)
        as paidamt_89_plus,
      case when (calculated paidamt_01) > 0 and
        (calculated paidamt_12) > 0 and
        (calculated paidamt_89_plus) > 0 then 1
        else 0 end as all_3
    from prac_servcd
    group by pracnum
  )
  where all_3
;
quit;
```

# Subset to Practitioners with flag for all 3

PRAC\_ALL\_3 ▾

 |  Filter and Sort |  Query Builder | Data ▾ | Describe ▾ | Graph ▾ | Analyze ▾ | Export ▾ | Send To

	 pracnum	 paidamt_01	 paidamt_12	 paidamt_89_plus	 all_3
1	1	50000	10000	20000	1
2	3	60000	5000	30000	1

# Select Values into Macro Variable

```
proc sql noprint;  
  select count(distinct pracnum) into :prac_cnt  
  from pracds;  
quit;
```

```
%put prac_cnt = &prac_cnt;
```

Results (In Log):

```
prac_cnt =      24793
```

# Select Values into Macro Variables

```
proc sql noprint;  
  select sum(popn), count(*)  
    into :pop_lha61, :rec_cnt_lha61  
  from people_data  
  where lfsclyr = 20042005  
    and clntlha = 61;  
quit;  
  
%put pop_lha61=&pop_lha61;  
%put rec_cnt_lha61=&rec_cnt_lha61;
```

## Results (In Log):

```
pop_lha61= 208372  
rec_cnt_lha61= 40
```

# Aside: Formatting Macro Variables for Footnotes

```
%let pop_fmt = %sysfunc(putn(&pop_lha61, comma9.));
```

```
footnote1 j=1 "Note: LHA 61 had population of  
&pop_fmt in 2004/2005";
```

Resulting footnote:

Note: LHA 61 had population of 208,372 in 2004/2005

# Select list into macro variable

```
proc sql noprint;  
    select distinct servcd into :servcd_list  
        separated by ", "  
    from prac_servcd;  
quit;  
  
%put &servcd_list;
```

Results (In Log):

1, 12, 90, 91, 92

# Many to Many Joins Are Possible with Proc SQL

```
proc sql  stimer;
  create table uniq_pracs as
  select distinct pracnum from prac_servcd;

  create table uniq_servcd as
  select distinct servcd from prac_servcd;

  create table prac_servcd_combos as
  select a.pracnum,
         b.servcd
  from   uniq_pracs      a,
         uniq_servcd    b;
quit;
```

# All possible combinations of practitioner and service code

PRAC\_SERVCD\_COMBOS ▾

 |  Filter and Sort |  Query Builder |  Data ▾

	 pracnum	 servcd
1	1	1
2	1	12
3	1	90
4	1	91
5	1	92
6	2	1
7	2	12
8	2	90
9	2	91
10	2	92
11	3	1
12	3	12
13	3	90
14	3	91
15	3	92

# Proc SQL instead of Proc Print

Proc SQL is a handy alternative to Proc Print when you want to get a quick report.

Simply leave out the bit “`create table result_table as`”, and start with `select ...`

You might find it handy to use a TITLE statement *before* the SELECT statement.

```
proc sql;
title "List of all subjects";
  select distinct subject
  from grades
  order by subject;
quit;
```

