More with SQL Queries

Advanced SAS Programming
Some PROC SQL Options

- Options available in the PROC SQL statement...
  - FEEDBACK(NOFEEDBACK): provides more detailed information in the SAS log.
  - FLOW\(\leq n < m\rangle\) (NOFLOW): sets up wrapping of columns in the display.
  - NOPRINT(PRINT): suppresses printing of results to the output window.
Some PROC SQL Options

- More options...
  - NOEXEC(EXEC): stops statements from being executed—they’re still compiled and checked for proper syntax.
  - ERRORSTOP(NOERRORSTOP): terminates PROC SQL if a statement with an error is encountered.
  - INOBS=$n$, OUTOBS=$n$: restricts the number of rows input or output.
More on the Where Clause

- The WHERE clause looks very similar to the WHERE statement used elsewhere in SAS.

- Special operators allowed:
  - BETWEEN – AND: selecting values in a range
    - `where date between '01JAN2007'd and '25JAN2007'd`
  - CONTAINS (or ?): values that contain the string
    - `where name contains 'Mc'`
  - IN: matches one of the values in the list
    - `where region in ('Beaumont','Boston')`
More on the Where Clause

- **Special operators allowed:**
  - **IS MISSING** (or **IS NULL**): self explanatory
    - where date is missing
  - **=*:** sounds like the string
    - where name =* 'Bill'
  - **LIKE:** matches a pattern, possibly based on wildcards:
    - **%:** any number of characters (including 0)
    - **_**: a single character
    - where region like 'B&t%'  
    - still picks out Beaumont and Boston
Using Calculated Columns

Consider the following:

```sql
proc sql;
    select region, pol_type, jobtotal, 0.02*jobtotal as incidental
    from mysas.projects
    where region in ('Beaumont', 'Boston')
        and incidental ge 1000
    order by region, pol_type
    ;
    quit;
```
Using Calculated Columns

The following appears in the log:

```
    proc sql;
    select region, pol_type, jobtotal, 0.02*jobtotal as incidental
    from mysas.projects
    where region in ('Beaumont','Boston') and incidental ge 1000
    order by region, pol_type
    ;
ERROR: The following columns were not found in the contributing tables: incidental.
    quit;
NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE SQL used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds
```
Using Calculated Columns

- Why is an error generated?
  - Like in the data step and other procedures, where is a read condition—effectively, it’s applied before selection of columns.
  - Any variable referenced in the where statement is presumed to be in the data table(s) listed in the from statement.
  - Can use the CALCULATED keyword to override this behavior
Using Calculated Columns

- This gives the desired result:

```sql
proc sql;
    select region, pol_type, jobtotal, 0.02*jobtotal as incidental
    from mysas.projects
    where region in ('Beaumont','Boston')
        and calculated incidental ge 1000
    order by region, pol_type
;
quit;
```
Using Calculated Columns

- It would also have been possible to use
  
  where region in ('Beaumont', 'Boston') and
  
  0.02*jobtotal ge 1000

- But this will do the calculation twice for every record that is read (and still once on those skipped)—once in the where clause and then again in the select statement.
- The calculated keyword forces the calculation for each record, but it is preserved if the record is selected.
- Often it doesn’t matter which strategy is used, but it can be important for large data sets.
Subsetting by Summary Values

- The HAVING clause can be used in conjunction with the GROUP BY clause to subset on group summary values.

- It’s similar to the WHERE clause, but it’s applied after grouping, so the CALCULATED keyword is not used.
Subsetting by Summary Values

Example:

```sql
proc sql;
select region, pol_type,
    mean(jobtotal) as jobmean,
    mean(0.02*jobtotal) as incidental
from mysas.projects
where region in ('Beaumont','Boston')
group by region, pol_type
having incidental gt 1000
;
quit;
```
Subqueries

- It is possible to subset a query based on a condition requiring a query of its own, called a subquery.

- Subqueries can be made in both the HAVING and WHERE clauses.

- Their results may have several rows, but they are restricted to one column when used for comparison.
Subqueries

Example:

```sql
proc sql;
    select region, pol_type, mean(jobtotal) as jobmean
    from mysas.projects
    where region in ('Beaumont', 'Boston')
    group by region, pol_type
    having jobmean gt (select mean(jobtotal)
                                from mysas.projects
                             )
quit;
```

How does this subset the results?
Subqueries

- Compare the previous result to these two:

```sql
proc sql;
    select region, pol_type, mean(jobtotal) as jobmean
    from mysas.projects
    where region in ('Beaumont','Boston')
    group by region, pol_type
;
    select mean(jobtotal)
    from mysas.projects
;
quit;
```
Subqueries

Example:

```sql
proc sql;
    select region, pol_type, mean(jobtotal) as jobmean
    from mysas.projects
    where region in ('Beaumont','Boston')
    group by region, pol_type
    having jobmean gt (select mean(jobtotal)
                        from mysas.projects
                        where region in ('Beaumont','Boston'))
; quit;
```
Returning Multiple Values in a Subquery

- Subqueries for comparison are limited to a single column, but they can have multiple values.

- Requires the use of one of the following operators:
  - IN
  - EXISTS
  - ANY
  - ALL
Example

```
proc sql;
  select state, mean(ave_tsp) as tsp
  from mysas.airqual
  group by state
  having state in (select stname
                   from mysas.projects
                   where pol_type='TSP'
                   and jobtotal ge 70000)

; quit;
```

Returns, potentially, more than one value
Create a query like the previous that displays the average of monthly total suspended particulates (TSP) for each state that has an average job cost of more than $60,000 for TSP jobs.
How about this...

- Summarize (average) monthly TSP job costs for each state and month where the TSP level exceeds 100.
More with functions

There is some potential ambiguity between data step functions and SQL functions, consider:

```sql
proc sql;
    select region, pol_type, mean(equipmnt),
        mean(personel)
    from mysas.projects
    group by region, pol_type
    ;
    select region, pol_type, mean(equipmnt, personel)
    from mysas.projects
    group by region, pol_type
    ;
quit;
```
More with functions

There is some potential ambiguity between data step functions and SQL functions.

Operates vertically, across rows

```sql
proc sql;
  select region, pol_type, mean(equipmnt), mean(personel)
  from mysas.projects
  group by region, pol_type
;
  select region, pol_type, mean(equipmnt, personel)
  from mysas.projects
  group by region, pol_type
;
quit;
```
More with functions

- There is some potential ambiguity between data step functions and SQL functions.

```sql
proc sql;
select region, pol_type, mean(equipmnt),
mean(personel)
from mysas.projects
group by region, pol_type;
;
select region, pol_type, mean(equipmnt, personel)
from mysas.projects
group by region, pol_type;
quit;
```

Operates vertically, across rows

Operates across columns
More with functions

Consider

```sql
proc sql;
   select region, pol_type, median(equipmnt), median(personel)
   from mysas.projects
   group by region, pol_type
;
quit;
```

Why do we not get a vertical summary in this case?
More with functions

- It is possible to put conditional logic into summary functions:

```sql
proc sql;
  select region, pol_type, freq(jobtotal) as freq
  from mysas.projects
  where jobtotal gt 60000
  group by region, pol_type
;
  select region, pol_type, sum(jobtotal gt 60000) as freq
  from mysas.projects
  group by region, pol_type
;
quit;
```
More with functions

- It is possible to put conditional logic into summary functions:

```sql
proc sql;
   select region, pol_type, freq(jobtotal) as freq
   from mysas.projects
   where jobtotal gt 60000
   group by region, pol_type;

   select region, pol_type, sum(jobtotal gt 60000) as freq
   from mysas.projects
   group by region, pol_type;
quit;
```

Counts observations greater than 60,000
More with functions

- It is possible to put conditional logic into summary functions:

```sql
proc sql;
select region, pol_type, freq(jobtotal) as freq
from mysas.projects
where jobtotal gt 60000
group by region, pol_type
;
select region, pol_type, sum(jobtotal gt 60000) as freq
from mysas.projects
group by region, pol_type
;
quit;
```

Counts observations greater than 60,000
Sums a 0/1 variable based on condition
Try...

- For the flightdelays data set, create a table with: destination, average delay, maximum delay and percentage of flights delayed.
Other conditional logic

- For conditional computations, it is possible to construct logic similar to the IF-THEN-ELSE or SELECT blocks in the data step using the CASE expression.
Consider:

```sql
proc sql;
select case
  when lt eq 1 then 'Oligotrophic'
  when lt eq 2 then 'Eutrophic'
  when lt eq 3 then 'Mesotrophic'
else 'Unknown'
end
as lake_type,
  hg, elv, sa
from mysas.fish
;
quit;
```
Other conditional logic

Consider:

```
proc sql;
   select case
   when lt eq 1 then 'Oligotrophic'
   when lt eq 2 then 'Eutrophic'
   when lt eq 3 then 'Mesotrophic'
   else 'Unknown'
   end as lake_type,
   hg, elv, sa
   from mysas.fish
;
quit;
```

The CASE expression is a block...
Other conditional logic

Consider:

```sql
proc sql;
select case
  when lt eq 1 then 'Oligotrophic'
  when lt eq 2 then 'Eutrophic'
  when lt eq 3 then 'Mesotrophic'
else 'Unknown'
end
as lake_type,
hg, elv, sa
from mysas.fish
;
quit;
```

Multiple conditions and a catch all are permitted
Other conditional logic

Consider:

```sql
proc sql;
  select case
    when lt eq 1 then 'Oligotrophic'
    when lt eq 2 then 'Eutrophic'
    when lt eq 3 then 'Mesotrophic'
    else 'Unknown'
  end
  as lake_type,
  hg, elv, sa
  from mysas.fish
;
quit;
```

Based on the condition a value is assigned to a new variable
So these need to be the same type
Consider:

```sql
proc sql;
    select case
        when lt eq 1 then 'Oligotrophic'
        when lt eq 2 then 'Eutrophic'
        when lt eq 3 then 'Mesotrophic'
    else 'Unknown'
    end as lake_type,
    hg, elv, sa
    from mysas.fish
; quit;
```

A variable name is automatically assigned unless you set one...
Consider:

```sql
proc sql;
select case
    when lt eq 1 then 'Oligotrophic'
    when lt eq 2 then 'Eutrophic'
    when lt eq 3 then 'Mesotrophic'
else 'Unknown'
end
as lake_type,
hg, elv, sa
from mysas.fish
;
quit;
```

This entire expression sets up the first variable in the select statement (ends with the first comma)
Other conditional logic

Consider:

```sql
proc sql;
select case when lt 1 then 'Oligotrophic'
when 2 then 'Eutrophic'
when 3 then 'Mesotrophic'
else 'Unknown'
end
as lake_type,
mean(hg), mean(elv), mean(sa)
from mysas.fish
group by lake_type
; quit;
```

If each condition is equality, this can be short-cut much like a select block.
Other conditional logic

Consider:

```sql
proc sql;
    select case lt
        when 1 then 'Oligotrophic'
        when 2 then 'Eutrophic'
        when 3 then 'Mesotrophic'
        else 'Unknown'
    end
    as lake_type,
    mean(hg), mean(elv), mean(sa)
from mysas.fish
group by lake_type
;
quit;
```

And it can be treated just like any other calculated variable.
Consider...

- Previously, we may have generated the preceding summary using a format (in conjunction with means):

```plaintext
proc format;
  value lake
    1='Oligotrophic'
    2='Eutrophic'
    3='Mesotrophic'
  other='Unknown'
  ;
run;
```

- Can we use this in SQL?
Using formats

Formats (and labels) can be set in the select statement:

```sql
proc sql;
select  lt format=lake., mean(hg), mean(elv), mean(sa)
from mysas.fish
group by lt
;
quit;
```
Using formats

- How about this?

```plaintext
proc format;
  value safety
  low-<0.5='Safe'
  0.5-<1='Marginal'
  1-high='Unsafe';
run;
proc sql;
  select hg format=safety., mean(elv), mean(sa)
  from mysas.fish
  group by hg;
quit;
```

- SQL grouping does not respect formats in the same way as class statements typically do...can you fix this?