: Conditional Logic, Macro Variables and Subqueries.

By
Christopher Zogby

Outline

■ Introduction to SQL
  ◆ Basic SQL syntax
■ Conditional Logic
  ◆ Syntax/Examples
■ Macro Variables
  ◆ Syntax/Examples
■ Subqueries
  ◆ Syntax/Examples
Basic SQL Syntax

- What is SQL?
- SELECT, FROM, WHERE statements
- CREATE/DROP/ALTER statements
- ORDER BY, GROUP BY clauses

What is SQL?

- SQL = structured query language
- An ANSI database language
  - ANSI = American National Standards Institute
- SQL functionality provided as a SAS procedure (i.e. PROC SQL;)
SELECT, FROM, WHERE

statements

- **SELECT**
  - comma delimited variable list, or
  - ‘*’, for all variables in table or view

- **FROM**
  - table name
  - view name

- **Where**
  - clause that determines which rows pass to results set.

SELECT-FROM

Example
```sql
proc sql;

title 'Select entire contents of table, SASHELP.CLASS.';

SELECT *
FROM sashelp.class;

title;
```

Select entire contents of table, SASHELP.CLASS.

<table>
<thead>
<tr>
<th>Name</th>
<th>Sex</th>
<th>Age</th>
<th>Height</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alfred</td>
<td>M</td>
<td>14</td>
<td>69</td>
<td>112.5</td>
</tr>
<tr>
<td>Alice</td>
<td>F</td>
<td>13</td>
<td>56.5</td>
<td>84</td>
</tr>
<tr>
<td>Barbara</td>
<td>F</td>
<td>13</td>
<td>65.3</td>
<td>98</td>
</tr>
<tr>
<td>Carol</td>
<td>F</td>
<td>14</td>
<td>62.8</td>
<td>102.5</td>
</tr>
<tr>
<td>Henry</td>
<td>M</td>
<td>14</td>
<td>63.5</td>
<td>102.5</td>
</tr>
<tr>
<td>James</td>
<td>M</td>
<td>12</td>
<td>57.3</td>
<td>83</td>
</tr>
<tr>
<td>Jane</td>
<td>F</td>
<td>12</td>
<td>59.8</td>
<td>84.5</td>
</tr>
<tr>
<td>Janet</td>
<td>F</td>
<td>15</td>
<td>62.5</td>
<td>112.5</td>
</tr>
<tr>
<td>Jeffrey</td>
<td>M</td>
<td>13</td>
<td>62.5</td>
<td>84</td>
</tr>
<tr>
<td>John</td>
<td>M</td>
<td>12</td>
<td>59</td>
<td>99.5</td>
</tr>
<tr>
<td>Joyce</td>
<td>F</td>
<td>11</td>
<td>51.3</td>
<td>50.5</td>
</tr>
<tr>
<td>Judy</td>
<td>F</td>
<td>14</td>
<td>64.3</td>
<td>90</td>
</tr>
<tr>
<td>Louise</td>
<td>F</td>
<td>12</td>
<td>56.3</td>
<td>77</td>
</tr>
<tr>
<td>Mary</td>
<td>F</td>
<td>15</td>
<td>66.5</td>
<td>112</td>
</tr>
<tr>
<td>Philip</td>
<td>M</td>
<td>16</td>
<td>72</td>
<td>150</td>
</tr>
<tr>
<td>Robert</td>
<td>M</td>
<td>12</td>
<td>64.8</td>
<td>128</td>
</tr>
<tr>
<td>Ronald</td>
<td>M</td>
<td>15</td>
<td>67</td>
<td>133</td>
</tr>
<tr>
<td>Thomas</td>
<td>M</td>
<td>11</td>
<td>57.5</td>
<td>85</td>
</tr>
<tr>
<td>William</td>
<td>M</td>
<td>15</td>
<td>66.5</td>
<td>112</td>
</tr>
</tbody>
</table>
SELECT-FROM-WHERE

EXAMPLE

title 'Select girls with first names that begin with ' 'J' from table, SASHELP.CLASS.';

SELECT name,
      sex
FROM sashelp.class
WHERE name like 'J%' and
      sex='F' ;
title;
quit;
Select girls with first names that begin with J from table, SASHELP.CLASS.

<table>
<thead>
<tr>
<th>Name</th>
<th>Sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>F</td>
</tr>
<tr>
<td>Janet</td>
<td>F</td>
</tr>
<tr>
<td>Joyce</td>
<td>F</td>
</tr>
<tr>
<td>Judy</td>
<td>F</td>
</tr>
</tbody>
</table>

CREATE/DROP/ALTER statements

- **CREATE**
  - table/view/index

- **DROP**
  - table/view/index

- **ALTER**
  - table
    - add (column-definition, constraint-clause)
    - modify (column-definition)
    - drop (column-definition, constraint clause)
proc sql;
libname devl 'c:\development';
create table devl.class as /* CREATE TABLE */
select * from sashelp.class
;
drop table devl.class; /* DROP TABLE */
libname devl;
create table o_negative_donors /* CREATE TABLE */
(donor_name varchar(40),
blood_type varchar(3),
rh_factor char(1))
;
/* ALTER TABLE */
alter table o_negative_donors /* ADD COLUMN, CONSTRAINT */
add donor_comments varchar(1000),
constraint only_o_neg check(blood_type = 'O' and
rh_factors='')
;
drop table o_negative_donors; /* DROP TABLE */
quit;
ORDER BY, GROUP BY clauses

- GROUP BY
  - Used when summarizing by one or more class variable
  - comma delimited
- Order BY
  - Used when sorting resulting data
  - comma delimited list
  - ascending or descending order

GROUP BY-ORDER BY

Example
proc sql;

title 'Student frequency by sex.';

select  sex,  
        count(*) as count  
from sashelp.class  
group by sex  
order by count desc  
;

title;

quit;
Outline

- Introduction to SQL
  - Basic SQL syntax
- Conditional Logic
  - Syntax/Examples

Conditional Logic

- Conditionally assign values to a variable.
- Valid in SELECT/UPDATE/INSERT statement.
- CASE expression
CASE expression syntax

```plaintext
CASE <case-operand>
  WHEN when-condition THEN result-expression
  <WHEN when-condition THEN result-expression>...
  <ELSE result-expression>
END
```

CASE expression’s rules of operation.

- Need at least one WHEN-THEN clause
- Need END to terminate the expression
- CASE operand present
  - WHEN condition & operand compared for equality
    - operand=WHEN condition, adjoining THEN’s result-expression executes
      - no further WHEN-THEN statements are evaluated.
    - operand ^= WHEN condition, following WHEN condition(s) evaluated.
Rules of operation (continued).

- CASE operand omitted
  - WHEN condition evaluated as Boolean value
    - WHEN condition returns non-missing, non-zero result, adjoining THEN’s result-expression executes
      - no further WHEN-THEN statements are evaluated.
    - If operand ^= WHEN condition, following WHEN condition(s) evaluated.

- No WHEN conditions true, ELSE’s results-expression or (without ELSE) null returned

CASE Expression: no operand.

Example
proc sql;
    title "CASE expression, without operand.";
    select weight,
         case
             when weight = 80 then ''
             when 80 < weight < 110 then 'Medium'
             when 110 <= weight <= 150 then 'Heavy'
             else                          'Heavy'
         end as weight_class
    from sashelp.class;
    title;
CASE Expression: operand.

Example

title "CASE expression, with operand.";

select sex,
    case sex
        when 'M' then 'BOY'
        when 'F' then 'GIRL'
    end as boy_or_girl
from sashelp.class;

title;
quit;
CASE expression, with operand.

Sex boy_or_girl
---------------
M    BOY
F    GIRL
F    GIRL
F    GIRL
M    BOY
M    BOY
F    GIRL
F    GIRL
F    GIRL
M    BOY
M    BOY
F    GIRL
F    GIRL
F    GIRL
F    GIRL
M    BOY
M    BOY
M    BOY
M    BOY
M    BOY
M    BOY

Outline

- Introduction to SQL
  - Basic SQL syntax
- Conditional Logic
  - Syntax/Examples
- Macro Variables
  - Syntax/Examples
Macro Variables

- Outer query’s SELECT statement
- Not valid when a table or view is created
- Can create macro variable for each queried column
- Macro variable can store single or multiple rows from queried column
- Macro variable created using INTO clause

INTO clause syntax

```sql
INTO macro-variable-specification
<, macro-variable-specification>...
  • macro-variable-specification is one of the following:
    macro-variable <SEPARATED BY 'character' <NOTRIM>>;

macro-variable-1 - macro-variable-n <NOTRIM>;
```
Macro Variables: multiple columns, single row.

Example

```sql
proc sql;
reset noprint;
select round(mean(height),0.01),
    round(mean(weight),0.01)
into :avg_height_boys,
    :avg_weight_boys
from sashelp.class
where sex='M';
reset print;
title 'Boys below average weight ('%trim(&avg_weight_boys)
    ') and average height ('%trim(&avg_height_boys)').';
select *
from sashelp.class
where sex='M' and
    weight < &avg_weight_boys and
    height < &avg_height_boys;
quit;
```
Boys below average weight (108.95) and average height (63.91).

<table>
<thead>
<tr>
<th>Name</th>
<th>Sex</th>
<th>Age</th>
<th>Height</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Henry</td>
<td>M</td>
<td>14</td>
<td>63.5</td>
<td>102.5</td>
</tr>
<tr>
<td>James</td>
<td>M</td>
<td>12</td>
<td>57.3</td>
<td>83</td>
</tr>
<tr>
<td>Jeffrey</td>
<td>M</td>
<td>13</td>
<td>62.5</td>
<td>84</td>
</tr>
<tr>
<td>John</td>
<td>M</td>
<td>12</td>
<td>59</td>
<td>99.5</td>
</tr>
<tr>
<td>Thomas</td>
<td>M</td>
<td>11</td>
<td>57.5</td>
<td>85</td>
</tr>
</tbody>
</table>

Macro Variables: variable for each resulting row.

Example
```sql
proc sql;
reset noprint;
select count(*) into :girls
from sashelp.class
where sex='F';
select weight into :gweight1 through :gweight%left(&girls)
from sashelp.class
where sex='F';
reset number print;
title "Girls' weight. Third girl's weight=%trim(&gweight3).";
select sex, weight
from sashelp.class
where sex='F';
title;
quit;
```

Girls' weight. Third girl's weight=102.5.

<table>
<thead>
<tr>
<th>Row</th>
<th>Sex</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td>84</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>98</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>102.5</td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td>84.5</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>112.5</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>50.5</td>
</tr>
<tr>
<td>7</td>
<td>F</td>
<td>90</td>
</tr>
<tr>
<td>8</td>
<td>F</td>
<td>77</td>
</tr>
<tr>
<td>9</td>
<td>F</td>
<td>112</td>
</tr>
</tbody>
</table>
Macro Variables:
Multiple, delimited rows stored in single variable.

Example

/* Create Temp Tables to be dropped */
data temp1 temp2 temp3;
run;

proc sql;
reset noprint;
select trim(libname)||'.'||trim(memname)
into :temp_files separated by ', ';
from dictionary.tables
where libname='WORK' and
      memtype='DATA' and
      memname like 'TEMP%'
;
%put &temp_files=;
drop table &temp_files;
quit;
proc sql;
/* Macro Variables: Multiple, Delimited Rows Stored in Single Variable */
reset noprint;
select trim(libname)||'.'||trim(memname)
into :temp_files separated by ', '
from dictionary.tables
where libname='WORK' and
memtype='DATA' and
memname like 'TEMP%';
%put &temp_files=;
WORK.TEMP1, WORK.TEMP2, WORK.TEMP3=
drop table &temp_files;
NOTE: Table WORK.TEMP1 has been dropped.
NOTE: Table WORK.TEMP2 has been dropped.
NOTE: Table WORK.TEMP3 has been dropped.
quit;

Outline

- Introduction to SQL
  - Basic SQL syntax
- Conditional Logic
  - Syntax/Examples
- Macro Variables
  - Syntax/Examples
- Subqueries
  - Syntax/Examples
Subqueries (SQ)
- Contained within parenthesis.
- SQ is query that execute within context of another query.
- Innermost SQ executes first, followed by next innermost, and so on until outer query is executed.
- Can return single or many rows.
  - Single row SQ can be evaluated like constant or variable in next outermost query.
  - Multi-row SQ can be evaluated with IN operator, or treated as a separate table in FROM statement.

Subquery: single row returned in INSERT statement.

Example
proc sql;

insert into avg_growth_monthly
    set month = month(today()),
    year = year(today()),
    weight_boys = (select avg(weight) from sashelp.class
                    where sex='M'),
    height_boys = (select avg(height) from sashelp.class
                    where sex='M'),
    weight_girls = (select avg(weight) from sashelp.class
                    where sex='F'),
    height_girls = (select avg(height) from sashelp.class
                    where sex='F');

TITLE "Monthly average height & weight."
SELECT *
FROM avg_growth_monthly;
TITLE;
QUIT;
Subquery: multi-row, evaluated by outer select's IN operator

Example

```sql
proc sql;
create table postmortem_ge10K_payments as
select *
from daily_payments
where amount >= 10000 and
  vet_id in(select veteran_id
              from vet_demographics
              where death_date is not null
            );
quit;
```
**Subquery: referenced as a table.**

**Example**

```sql
proc sql;
  title "Students 10% below or above sex-specific average height & weight.";
  select a.*,
    round(b.sex_specific_avg_weight, 0.01) as avg_weight,
    round(b.sex_specific_avg_height, 0.01) as avg_height,
    case when weight < b.sex_specific_avg_weight then '-'
      else '//' end as indicator
  from sashelp.class a,
  (select sex,
    avg(weight) as sex_specific_avg_weight,
    avg(height) as sex_specific_avg_height
  from sashelp.class
  group by sex
  ) b
  where a.sex=b.sex and
  (a.weight/b.sex_specific_avg_weight le .90 and
   a.height/b.sex_specific_avg_height le .90
  ) or
  (a.weight/b.sex_specific_avg_weight ge 1.10 and
   a.height/b.sex_specific_avg_height ge 1.10
  );
quit;
```
Students 10% below or above sex-specific average height & weight.

<table>
<thead>
<tr>
<th>Name</th>
<th>Sex</th>
<th>Age</th>
<th>Height</th>
<th>Weight</th>
<th>avg_weight</th>
<th>avg_height</th>
<th>indicator</th>
</tr>
</thead>
<tbody>
<tr>
<td>James</td>
<td>M</td>
<td>12</td>
<td>57.3</td>
<td>83</td>
<td>108.95</td>
<td>63.91</td>
<td>-</td>
</tr>
<tr>
<td>Joyce</td>
<td>F</td>
<td>11</td>
<td>51.3</td>
<td>50.5</td>
<td>90.11</td>
<td>60.59</td>
<td>-</td>
</tr>
<tr>
<td>Philip</td>
<td>M</td>
<td>16</td>
<td>72</td>
<td>150</td>
<td>108.95</td>
<td>63.91</td>
<td>+</td>
</tr>
<tr>
<td>Thomas</td>
<td>M</td>
<td>11</td>
<td>57.5</td>
<td>85</td>
<td>108.95</td>
<td>63.91</td>
<td>-</td>
</tr>
</tbody>
</table>

Conclusions

◆ SAS’s SQL extends beyond SELECT, FROM and WHERE statements.
  ◆ Conditional logic provides flexibility in variable creation.
  ◆ Macro variable creation allows some automation of SQL code.
  ◆ Subqueries make SQL a compact, powerful & sometimes novel coding solution.
Questions or Comments

E-mail: czogby@erols.com