** SQL Statements **

PROC SQL consists of TEN statements:

1. SELECT to perform queries
2. VALIDATE to validate the syntax of a query
3. DESCRIBE to show how a view has been defined
4. CREATE to create a table (SAS data set), index or view
5. DROP to delete a table, index or view
6. UPDATE change the values in a table
7. INSERT add rows to a table
8. DELETE to delete rows from a table
9. ALTER to add, delete or modify columns in a table
10. RESET add, change or deletion of options

Of these, by far the most important is the SELECT statement.

** The Select Statement **

*Q02E01 The Select statement;*

```sql
proc sql;
select * 
from saved.computer;
quit;
```

** Usage **

Invoke Proc SQL and then use a SELECT statement:

```sql
proc sql;
select * 
from saved.computer;
quit;
```

Note that while Proc SQL is running, the RUN; statement itself simply produces an interesting note on the Log:

** Basic Examples **

The SQL procedure uses the SELECT statement to perform a wide variety of queries. Within the SELECT statement are different Clauses:

*Q02E02 The Order By clause to determine order for rows; *

```sql
select * 
from saved.computer 
order by disk;
```

This code produces the same result as:

```sql
proc sort data = saved.computer; 
by disk; 
run;
```

```sql
proc print data = saved.computer; 
by disk; 
run;
```
Advantages of SQL:
** No physical sort has to be performed
Notice that no physical output file has been produced from this query, simply a listing as output.

Selecting Rows
Selecting Rows is done with the WHERE clause:

**Q02E03  The subsetting rows with where;
program
  select *
  from saved.computer
  where supplier = 'KETCHUP COMPUTERS'
  order by disk;
quit;

The WHERE clause syntax:
(These variants also apply to the Having clause. Notes:
** ¬ = is normally the symbol used for Not Equals on an IBM mainframe.
** ^= is normally the symbol used for Not Equals on ASCII based machines.
** NOT= and NE can also be used for Not Equals.
** The ‘Sounds Like’ operator uses the Soundex algorithm. This is normally used to search for variants in names, for example telephone directory applications, but is not perfect.
** The BETWEEN operator includes values defined in the range.
** IS NULL and IS MISSING are equivalent and match with all missing values including special missing values.
** Computed columns cannot be referred to by name in a WHERE clause. They have to be recalculated for the WHERE clause.
** SAS functions are supported except for LAG, DIF and SOUND. These ‘ordinary SAS’ functions should not be confused with SQL summary functions seen in Q2.3.

Controls and Enhancements
Selecting columns
Using the column names from the SAS file allows choice of the columns in the report:

**Q02E04  Selecting columns to display in select statement;
program
  select type, disk, retail
  from saved.computer
  where supplier = 'KETCHUP COMPUTERS'
  order by disk;
quit;

This is the equivalent of:

**PROGRAM EDITOR

proc sort data = saved.computer
  by disk;
run;
proc print data = saved.computer
  (where = (supplier = 'KETCHUP COMPUTERS'));
  var type disk retail;
run;

Again, the SQL advantage is the lack of the sort step.
Note that the list of column names require to be delimited by commas:

select type, disk, retail

All lists of column names and table names require commas in SQL syntax.

Calculating Columns
Columns can be calculated by assigning an expression to an item name:

**Q02E05 Assigning an expression to column name;
program
  select type,
    disk,
    retail,
    retail * 7/47 as vat
  from saved.computer
  where supplier = 'KETCHUP COMPUTERS'
  order by disk;
quit;

Notice the structure: expression AS variable
as opposed to the traditional SAS: variable = expression;

Notice that the traditional equivalent would now require an additional step:

**PROGRAM EDITOR

data new;
  set saved.computer;
  vat = retail * 7/47;
  proc sort data = new
    by disk;
  proc print data = new
    (where = (supplier = 'KETCHUP COMPUTERS'));
    var type disk retail vat;
run;
**Formatting Values - A Column Modifier**

The format option appears after the column name, not as a separate statement:

```sas
*Q02E06  Formatting values with a column modifier;

title 'Ketchup Computers, VAT element of prices';
select type,
    disk,
    retail,
    retail*7/47 as vat format=7.2
from saved.computer
where supplier='KETCHUP COMPUTERS'
order by disk ;
```

This is a SAS enhancement to a standard SQL.

**Labeling Columns - a Column Modifier**

Column headings can be changed with the LABEL option:

```sas
*Q02E07  Labeling columns with a column modifier;

proc format;
    picture pound low-high='000,000,009.99'(prefix=');
    proc sql;
    title 'Ketchup Computers, VAT element of prices';
    select type label='Computer Type',
        retail label='Retail Price' format=pound9.2,
        retail*7/47 as vat format=pound7.2
    from saved.computer
    where supplier='KETCHUP COMPUTERS'
    order by disk ;
```

This is a SAS enhancement to standard SQL.

**Using Functions**

Ordinary 'Data Step' functions can be used in these expressions as noted in Q2.1:

```sas
*Q02E08   Using functions;

options nodate nonumber;
    title 'Charges raised for car hire';
    proc format;
        picture pound low-high = '000,000,009.99'(prefix='');
        proc sql;
            title 'Which disk types are sold by Ketchup?';
            select substr(carkey,2,1) format=$model.,
                custkey label='Customer Code',
                daychg label='Daily Charge',
            (endate - stdate +1) as duration label= 'Days Hired',
            (endate - stdate +1)*daychg format = pound7.2,
            round(((endate - stdate +1)*daychg*7/47),0.01) as vat format = pound7.2
        from saved.carhire
        order by 5 desc;
```

Notes:

Functions can be used to derive calculated columns as shown by the SUBSTR function above and can be used to change calculate values as shown by the ROUND function. Derived columns need not be given aliases using the AS syntax.

Aliases cannot be used in further calculations. For example, if the expression in line 17 above:

```
(enddate - stdate +1)*daychg
```

were replaced by

```
duration*daychg
```

an error would result, duration not being found.

The order by clause can use the ordinal position of the column. In the example above the 5th column in used. Ordering can be done a calculated column with no alias.

**Distinct Values**

A useful keyword is DISTINCT, which allows selection of unique values of a column:

```sas
*Q02E09  Selection of unique key values with Distinct;

title 'Which disk types are sold by Ketchup?';
select distinct disk
from saved.computer
where supplier='KETCHUP COMPUTERS'
order by disk ;
```

```
Which disk types are sold by Ketchup?

<table>
<thead>
<tr>
<th>DISK</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>60</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>120</td>
</tr>
<tr>
<td>200</td>
</tr>
</tbody>
</table>
```

Traditional SAS programming would involve the following:
title 'Which disk types are sold by Ketchup?';
proc sort data=saved.computer out=sorted;
  by disk;
data unique(keep=disk);
  set sorted(keep=disk supplier);
  by disk;
  if last.disk;
  where supplier='KETCHUP COMPUTERS';
proc print data=unique;
run;

Selecting data with DISTINCT

The DISTINCT keyword can be used to select data for all combinations of columns:

```
*Q02E10  Distinct can apply to unique combinations;

   title 'All combinations of disk and type';
   select distinct disk label='Hard Disk Size',
              type label='Computer Type'
   from saved.computer
   where supplier='KETCHUP COMPUTERS'
   order by disk;
```

The DISTINCT keyword applies to all the column names and each unique combination of values is returned.

Data/Proc Step methods would use two BY variables. Note the effect on the inner variable (type) when the outer variable (disk) changes value:

```
title 'Which disk types are sold by Ketchup?';
proc sort data=saved.computer out=sorted;
  by disk type;
data unique(keep=disk type);
  set sorted(keep=disk type supplier);
  by disk type;
  if last.type;
  where supplier='KETCHUP COMPUTERS';
proc print data=unique;
run;
```

Syntax Checking

Use the VALIDATE statement, before the SELECT statement, to check the SQL statements without executing them:

```
*Q02E11  Checking select statement syntax without executing;

   validate
   select distinct disk label = 'Hard Disk Size',
              type label = 'Computer Type'
   from saved.computer
   order by disk
   where supplier = 'KETCHUP COMPUTERS' ;
```

LOG

```
Proc SQL has valid syntax

This facility can only be used with a Query-expression i.e. to qualify
the syntax of a SELECT.

Syntax Errors

This syntax error is caused by the ORDER BY option:

```
*Q02E12   Where is the syntax error here?;

   validate
   select distinct disk label = 'Hard Disk Size',
              type label = 'Computer Type'
   from saved.Computer
   order by disk
   where supplier = 'KETCHUP COMPUTERS' ;
```

LOG

```
ERROR 22-322: Expecting one of the following: (, **, *, /, +, -, !!,¦¦,<,<=,<>,=,>,>=,?,CONTAINS,
                      eq, ge, gt, le, lt, ne, ^=, ~=,
                      & AND, !, or, |, OR, !, '.
The statement is being ignored.
ERROR 202-322:  The option or parameter is not recognized. Make sure the ORDER by statement is the last option on the
SELECT statement.
```
Analysis on Groups

Summary Functions

A series of functions are provided to work 'down the columns'. A complete list of these functions is given in Q2.5.

**PROGRAM EDITOR**

*Q02E13 Analysis down a column for groups;

```
select mean(retail) as avprice
from saved.computer;
```

**OUTPUT**

```
AVPRICE
1929.167
```

This is the equivalent of:

**PROGRAM EDITOR**

```
proc means data = saved.computer mean;
var retail;
run;
```

With more than one argument, the function performs for each row:

**PROGRAM EDITOR**

*Q02E14 More then one argument to analyze each row;

```
select retail format= pound10.2,
    retail * 7/47 as VAT format = pound8.2,
    sum(retail,retail*7/47) as gross
    format =pound10.2
from saved.computer;
```

With a single argument, but with other selected columns, the function gives a result for all the rows, then merges the summary back with each row:

**PROGRAM EDITOR**

*Q02E15 Merges summary value onto each row of output;

```
select cpu,
    disk,
    (retail -wholesal) as profit label='Profit',
    mean(retail-wholesale) as avprofit label = 'Average Profit',
    (retail-wholesal) - mean(retail -wholesal) as diff label = 'Difference'
from saved.computer
where supplier contains 'FLOPPY';
```

To accomplish the same thing in Data/Proc step either requires use of Proc Means/Summary to create a one-observation, one-variable data set which is then read into the data step alongside saved.computer or two passes of the data in the same data step:

**PROGRAM EDITOR**

```
data new;
    retain avprofit;
    if _n_ = 1 then do;
        do until(finish);
            set saved.computer end = finish
                nobs = numobs;
            profit=retail-wholesal;
            totprof+profit;
        end;
        avprofit = totprof / numobs;
    end;
    set saved.computer;
    profit = retail - wholesal;
    diff = (retail - wholesal) - avprofit;
run;
proc print data=new;
    var cpu disk profit avprofit diff;
    label profit='Profit'
        avprofit='Average Profit'
        diff = 'Difference';
run;
```

An important function is COUNT(*) which gives the number of rows:

**PROGRAM EDITOR**

*Q02E16 The count function supplies the number of rows;

```
select count(*) as no_rows
from saved.computer;
select sum(retail)/count(*) as average
from saved.computer;
quit;
```