

# Advanced SQL Processing

Prepared by

## Destiny Corporation

### International SAS® Training and Consulting

Destiny Corporation - 100 Great Meadow Rd Suite 601 - Wethersfield, CT 06109-2355

Phone: (860) 721-1684 - 1-800-7TRAINING - Fax: (860) 721-9784

Email: [info@destinycorp.com](mailto:info@destinycorp.com) Web: [www.destinycorp.com](http://www.destinycorp.com)

Copyright 1999

#### 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 than 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;
```

## OUTPUT

| RETAIL   | VAT    | GROSS    |
|----------|--------|----------|
| 750.00   | 111.70 | 861.70   |
| 800.00   | 119.14 | 919.14   |
| 950.00   | 141.48 | 1,091.48 |
| 950.00   | 141.48 | 1,091.48 |
| 1,150.00 | 171.27 | 1,321.27 |
| 1,150.00 | 171.27 | 1,321.27 |
|          | ...etc |          |
| 2,350.00 | 350.00 | 2,700.00 |
| 2,450.00 | 364.89 | 2,814.89 |
| 3,350.00 | 498.93 | 3,848.93 |
| 3,750.00 | 558.51 | 4,308.51 |

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

```

## LOG

```

379 select cpu,
380       disk,
381       (retail -wholesale)
382         as profit label='Profit',
383       mean(retail-wholesale)
384         as avprofit label = 'Average Profit',
385       (retail-wholesale) - mean(retail -wholesale)
386         as diff label = 'Difference'
387 from saved.computer
388 where supplier contains 'FLOPPY';
NOTE: The query requires remerging summary statistics back
      with the original data.

```

### OUTPUT

| CPU    | DISK | Profit | Average Profit | Difference |
|--------|------|--------|----------------|------------|
| 286    | 20   | 200    | 231.25         | -31.25     |
| 286    | 40   | 200    | 231.25         | -31.25     |
| 286    | 100  | 200    | 231.25         | -31.25     |
| 386SX  | 40   | 200    | 231.25         | -31.25     |
| etc... |      |        |                |            |
| 386DX  | 100  | 300    | 231.25         | 68.75      |
| 386DX  | 200  | 500    | 231.25         | 268.75     |
| 286    | 60   | 200    | 231.25         | -31.25     |

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-wholesale;
      totprof+profit;
    end;
    avprofit = totprof / numobs;
  end;
  set saved.computer;
  profit = retail - wholesale;
  diff = (retail - wholesale) - 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
rom saved.computer;
```

## OUTPUT

|          |
|----------|
| NO ROWS  |
| 36       |
| AVERAGE  |
| 1929.167 |

Analyzing groups of data is performed using the GROUP BY clause on the SELECT statement. The HAVING clause also affects the result. This option results in 5 styles of query:

### SAS programming:

| STYLE | SELECT statement                                                                                      | RESULT                                                                                                                                                                 |
|-------|-------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1     | 2 items: GROUP BY variable and summary function on second variable.                                   | Equivalent of BY statement. Has one row each value of GROUP by variable. Data calculated for each GROUP BY value. Ordered by Group BY.                                 |
| 2     | Any number of items: GROUP BY variable and several variables, at least one with summary function      | Has one row in original file, subject to WHERE or HAVING clauses. Data calculated for each GROUP BY value. Ordered by GROUP BY.                                        |
| 3     | Any number of items: GROUP BY variable and several variables all have summary function                | Has one row for each value of GROUP BY variable. Data calculated for each GROUP BY value. Ordered by GROUP BY                                                          |
| 4     | Any number of items: GROUP BY variable and several variables, summary function on HAVING not SELECT   | Has one row for each row in the original file, subject to the HAVING clause. Data is calculated for each GROUP BY value. Data ordered by GROUP BY variable             |
| 5     | Any number of items: GROUP BY variable and several variables, no summary function on SELECT or HAVING | GROUP BY translated into an ORDER BY option. Has one row for each value in the original table, subject to WHERE and HAVING clauses. Data ordered by GROUP BY variable. |

### Style 1

#### PROGRAM EDITOR

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

#### PROGRAM EDITOR

```
*Q02E17 Group By will group by statistic on select statement;

proc sql;
  select disk,
    mean(retail) as avgret
  from saved.computer
  group by disk;
```

#### OUTPUT

| DISK | AVGRET   |
|------|----------|
| 20   | 1483.333 |
| 40   | 1485     |
| 60   | 2300     |
| 100  | 1835     |
| 120  | 2483.333 |
| 200  | 3250     |

#### Accumulating values for a column

With traditional SAS programming, the data step can be programmed to count the number in a group, or to sum a variable for the unique values of another, as well as any other statistical measure:

#### PROGRAM EDITOR

```
title 'what is the total retail for each disk sold?';
proc sort data=saved.computer out=sorted;
  by disk;
run;
data unique(keep=disk totret);
  set sorted(keep=disk supplier retail);
  by disk;
  if first.disk then totret=0;
  totret+retail;
  if last.disk;
  where supplier='KETCHUP COMPUTERS';
run;
proc print data=unique;
run;
```

With SQL, we use the summary function - SUM(), and GROUP BY:

### PROGRAM EDITOR

```
*Q02E18  Summary for each unique value in a column after subsetting;

title 'Total retail for each disk type sold';
select disk,
       sum(retail) as totret
from saved.computer
where supplier='KETCHUP COMPUTERS'
group by disk ;
```

### OUTPUT

#### Total retail for each disk type sold

| DISK | TOTRET |
|------|--------|
| 20   | 8150   |
| 40   | 7800   |
| 60   | 5600   |
| 100  | 7300   |
| 120  | 7450   |

### Style 2

#### Comparing the averages

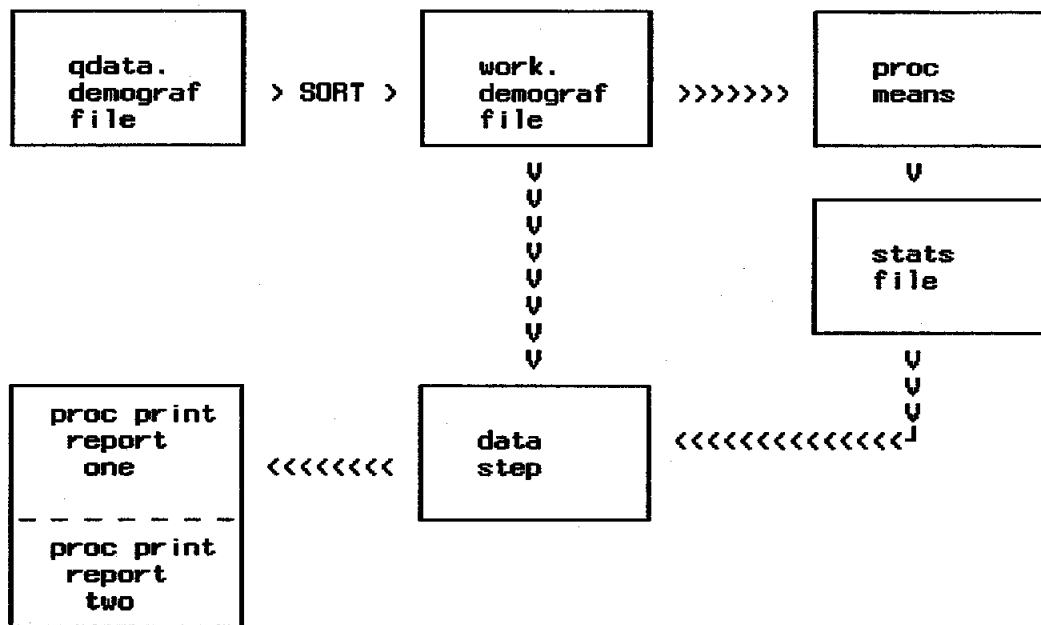
Quite often, we need to compare individual values with the average value for the group, instead of the whole file.

Traditional SAS programming would comprise:

### PROGRAM EDITOR

```
proc sort data=saved.demograf out=demograf;
  by gender;
run;
proc means data=demograf mean noprint;
  var salary;
  by gender;
  output out=stats mean=avgsal;
run;
data lowsals highsals;
  set demograf;
  by gender;
  if first.gender then set stats;
  if salary < avgsal then output lowsals;
  else output highsals;
run;
proc print data=lowsals;
  title 'Employees with lower than average salaries';
run;
proc print data=highsals;
  title 'Employees with higher than average salaries';
run;
```

### Program schematic



### Using SQL:

## Having

**Use HAVING when you want to perform a WHERE for groups in the data:**

## PROGRAM EDITOR

\*Q02E19 Having allows us to compare against group average;

```
select gender, status, salary, avg(salary) as avgsal
from saved.demograf
group by gender
having salary>avg(salary) ;
```

**LOG**

The query requires remerging summary statistics back with the original data

### OUTPUT

| GENDER | STATUS | SALARY | AVGSAL   |
|--------|--------|--------|----------|
| F      | M      | 30000  | 10980.95 |
| F      | M      | 15000  | 10980.95 |
| F      | SEP    | 18000  | 10980.95 |
| F      | S      | 30000  | 10980.95 |
| F      | M      | 15000  | 10980.95 |
| F      | M      | 13000  | 10980.95 |
| F      | M      | 15000  | 10980.95 |
| F      | W      | 30000  | 10980.95 |
| F      | M      | 18000  | 10980.95 |
| M      | M      | 23000  | 12007.14 |
| M      | M      | 23000  | 12007.14 |
| M      | M      | 12300  | 12007.14 |
| M      | M      | 40000  | 12007.14 |

Salary and Avgсал columns have been shown to illustrate the different averages for the 2 groups.

### Style 3

Let's consider the last output:

### OUTPUT

| GENDER | STATUS | SALARY | AVGSAL   |
|--------|--------|--------|----------|
| F      | M      | 30000  | 10980.95 |
| F      | M      | 15000  | 10980.95 |
| F      | SEP    | 18000  | 10980.95 |
| F      | S      | 30000  | 10980.95 |
| F      | S      | 13000  | 10980.95 |
| F      | M      | 15000  | 10980.95 |
| F      | M      | 13000  | 10980.95 |
| F      | M      | 15000  | 10980.95 |
| F      | W      | 30000  | 10980.95 |
| F      | M      | 18000  | 10980.95 |
| M      | M      | 23000  | 12007.14 |
| M      | M      | 23000  | 12007.14 |
| M      | M      | 12300  | 12007.14 |
| M      | M      | 40000  | 12007.14 |

What if we were only concerned with the last rows of gender?

We wish to calculate the average salary and average number of cars owned for each value of the *gender* column; moreover, we are only interested in those who earn more than £10,000 per year.

How do we alter the SQL so that only 2 rows result?

To do this, we need to apply summary functions to all items on the SELECT list:

The MAX and MIN statistics can be applied to character variables.

### PROGRAM EDITOR

\*Q02E20 One row reporting on statistics for each group;

```
select avg(salary) label='Average Salary'
              format=8.2,
       avg(cars) label='Average Number of Cars'
              format = 3.1,
       gender
from saved.demograf
where salary > 10000
group by gender;
```

The HAVING option needs to be replaced by a WHERE clause, so that the SELECT acts on rows, not groups. Otherwise all groups would be held, resulting in all rows.

### OUTPUT

| Average<br>Salary | Average<br>Number<br>of Cars | GENDER |
|-------------------|------------------------------|--------|
| 19700.00          | 1.4                          | F      |
| 20383.33          | 1.3                          | M      |

### Style 4

Although there is no summary function on the SELECT list, the HAVING clause does have a summary function, and the GROUP BY can group the data for calculation:

### PROGRAM EDITOR

\*Q02E21 Group on the sum function values with having statement;

```
select gender, status, salary
from saved.demograf
group by gender
having salary>avg(salary) ;
```

Each salary is compared to each gender's average salary.

#### OUTPUT

| GENDER | STATUS | SALARY |
|--------|--------|--------|
| F      | M      | 30000  |
| F      | M      | 15000  |
| F      | SEP    | 18000  |
| F      | S      | 30000  |
| F      | S      | 13000  |
| F      | M      | 15000  |
| F      | M      | 13000  |
| F      | M      | 15000  |
| F      | W      | 30000  |
| F      | M      | 18000  |
| M      | M      | 23000  |
| M      | M      | 23000  |
| M      | M      | 12300  |
| M      | M      | 40000  |

#### Style 5

Because there is no summary function on either the SELECT list or the HAVING clause, no grouping can occur, and the GROUP BY is translated into an ORDER BY clause:

#### PROGRAM EDITOR

\*Q02E22 Group By translated as Order By since no sum function exists;

```
select gender, status, salary
from saved.demograf
group by status
having salary>10000;
```

#### LOG

WARNING: A GROUP BY clause has been transformed into a ORDER BY clause because neither the SELECT clause nor the optional HAVING clause of the associated table-expression referenced a summary function.

### OUTPUT

| GENDER | STATUS | SALARY |
|--------|--------|--------|
| M      | M      | 12300  |
| M      | M      | 23000  |
| F      | M      | 13000  |
| M      | M      | 40000  |
| F      | M      | 30000  |
| F      | M      | 15000  |
| F      | M      | 15000  |
| F      | M      | 15000  |
| F      | M      | 18000  |
| M      | M      | 12000  |
| M      | M      | 23000  |
| F      | S      | 13000  |
| F      | S      | 30000  |
| M      | SEP    | 12000  |
| F      | SEP    | 18000  |

### Nested Subqueries

The result of a query may be embedded inside further queries; embedded queries are termed SUBQUERIES or INNER queries. They produce either single results or a set of values which are then part of the main query.

The results of a subquery are typically used as part of a WHERE or HAVING clause.

The inner query is evaluated before the outer query.

### Example

Let's examine the average profit for each CPU type:

### PROGRAM EDITOR

```
*Q02E23  Nest query within query, the inner evaluated first;

select cpu,
avg(retail-wholesale) as profit
from saved.computer
group by cpu;
```

### OUTPUT

| CPU   | PROFIT   |
|-------|----------|
| 286   | 194.4444 |
| 386DX | 285      |
| 386SX | 200      |
| 486DX | 250      |
| 486SX | 266.6667 |

Clearly some CPU types are more profitable, some less so.

### Comparison to overall figures

How do we compare the average profit for each CPU type to the overall average profit for all CPU types?

We need to use the HAVING option and a subquery:

#### **PROGRAM EDITOR**

```
*Q02E24  Compare each type to overall stats regardless of type;

  select cpu,                                < OUTER QUERY
         avg(etail-wholesal) as profit
  from saved.computer
  group by cpu
  having profit >
         (select avg(etail-wholesal)        < INNER QUERY
          from saved.computer);              < or SUBQUERY
```

#### **OUTPUT**

##### **CPUs with Higher Profit**

| CPU   | PROFIT  |
|-------|---------|
| 386DX | 285     |
| 486DX | 250     |
| 486SX | 266.667 |

#### **OUTPUT**

##### **Result of INNER QUERY**

233.3333

One way to accomplish this in traditional database processing steps would be as follows:

### PROGRAM EDITOR

```
data new;
  set saved.computer;
  profit = retail - wholesal;
run;
proc summary data=new;
  class cpu;
  var profit;
  output out=new2 mean=meanprof;
run;
data new3;
  retain totavprf;
  set new2;
  if _type_ = 0 then totavprf = meanprof;
  if meanprof > totavprf;
run;
proc print data=new3;
run;
```

### OUTPUT

| OBS | TOTAVPRF | CPU   | TYPE | FREQ | MEANPROF |
|-----|----------|-------|------|------|----------|
| 1   | 233.333  | 386DX | 1    | 10   | 285.000  |
| 2   | 233.333  | 486DX | 1    | 4    | 250.000  |
| 3   | 233.333  | 486SX | 1    | 3    | 266.667  |

Here the power of the SQL query is seen, one query replacing 4 steps.

This further example shows an inner query that results in a list of values which is further compared using an IN operator.

### PROGRAM EDITOR

```
*Q02E25 Mutually exclusive subset using in option;

select cpu,
       disk,
       retail - wholesale as profit
from saved.computer
where cpu in
  (select distinct cpu
   from saved.computer
   where retail < 1200) ;
```

Inner query result:

### OUTPUT

| CPU   |
|-------|
| 286   |
| 386SX |
| 386DX |

Full query result:

**OUTPUT**

|  | CPU    | DISK | PROFIT |
|--|--------|------|--------|
|  | 286    | 20   | 200    |
|  | 286    | 40   | 200    |
|  | 286    | 100  | 200    |
|  | 386SX  | 40   | 200    |
|  | etc... |      |        |
|  | 386DX  | 60   | 400    |
|  | 386DX  | 120  | 250    |
|  | 386DX  | 200  | 400    |

**Correlated Subqueries**

This term means using a subquery that depends on values in the outer query.

**Schematic**

Select name and date from A, but only those whose height is above 1.3:

| Table A |       |         | Table B |        |        |
|---------|-------|---------|---------|--------|--------|
| code    | name  | date    | code    | height | weight |
| aa      | jimmy | 24dec92 | aa      | 1.2000 | 76     |
| bb      | jack  | 12oct91 | ee      | 1.3300 | 51     |
| cc      | sally | 03aug65 | bb      | 1.3550 | 68     |
| dd      | suzie | 14feb78 | ff      | 1.2200 | 58     |

select name, date for each row of a:

from A

where 1.3 <

(select height from B where A.code=B.code)

**Process**

|        |           |           |          |
|--------|-----------|-----------|----------|
| row 1: | a.code=aa | b.code=aa | match    |
|        | 1.3       | < 1.2 ?   | no       |
| row 2: | a.code=bb | b.code=bb | match    |
|        | 1.3       | < 1.35 ?  | yes      |
| row 3: | a.code=cc | b.code=   | no match |
| row 4: | a.code=dd | b.code=   | no match |

The inner query cannot be evaluated directly, so the initial evaluation takes place for all rows in the table.

**Example**

Records are kept on the suppliers of computer equipment pertaining to delivery targets, product quality and technical support provided.

Here is the data:

#### PROGRAM EDITOR

```
select * from saved.compsupp;
```

#### OUTPUT

| SUPPLIER          | DELIVERY | PRODUCT | SUPPORT |
|-------------------|----------|---------|---------|
| FLOPPY COMPUTERS  | 8        | 4       | 6       |
| KETCHUP COMPUTERS | 7        | 7       | 8       |

Ratings are out of 10, 1=poor, 10=excellent.

We would like to assess which CPU type is the most profitable, and take into account the current quality ratings of our suppliers.

The outer query provides statistics on the most profitable CPU's, while the inner query chooses only those suppliers whose average rating over all quality measures is above 6.5.

#### PROGRAM EDITOR

```
*Q02E26 Outer query checks profit, inner check suppliers quality;
select supplier label='supplier',
       cpu,
       avg(retail-wholesale)
         label='Average Profit' format=7.2,
       std(retail-wholesale)
         label='Standard Deviation' format=5.2
  from saved.computer
 where 6.5 <
       (select mean(delivery,product,support)
        from saved.compsupp
        where computer.supplier=compsupp.supplier)
 group by supplier,cpu;
```

#### OUTPUT

| Supplier          | CPU   | Average Profit | Standard Deviation |
|-------------------|-------|----------------|--------------------|
| KETCHUP COMPUTERS | 286   | 187.50         | 25.00              |
| KETCHUP COMPUTERS | 386DX | 275.00         | 98.74              |
| KETCHUP COMPUTERS | 386SX | 200.00         | 0.00               |
| KETCHUP COMPUTERS | 486DX | 233.33         | 57.74              |
| KETCHUP COMPUTERS | 486SX | 400.00         |                    |