Class 28 (Proc Freq, Joins & Call Routine Statement)

In the below example, A Proc Freq (Cross list) works similar to an excel Pivot which counts actlevel (High, Low & MOD)per the 2 gender groups F & M and gives a grand total count at the bottom.

```
Proc freq data =sasuser.admit;

Tables sex*actlevel/crosslist;
Run;
```

Output:

The FREQ Procedure

Table of Sex by ActLevel

Sex	Act Level	Frequency	Percent	Row Percent	Column Percent
F	HIGH	4	19.05	36.36	57.14
	LOW	4	19.05	36.36	57.14
	MOD	3	14.29	27.27	42.86
	Total	11	52.38	100.00	
M	HIGH	3	14.29	30.00	42.86
	LOW	3	14.29	30.00	42.86
	MOD	4	19.05	40.00	57.14
	Total	10	47.62	100.00	
Total	HIGH	7	33.33		100.00
	LOW	7	33.33		100.00
	MOD	7	33.33		100.00
	Total	21	100.00		

Proc Freq (Weight Statement)

Statement Weight when used in Proc Freq works like a sum function, instead of counting it adds the values of variable written against it per the Tables variable.

Example:

```
Data Survey;
Input City Response Count;
cards;
1 1 35
1 0 65
2 1 40
2 0 60
3 1 25
3 0 75
;run;
```

```
proc freq data = Survey;
tables response;
weight Count;
run;
```

Output:

```
The FREQ Procedure
```

Response	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0	200	66.67	200	66.67
1	100	33.33	300	100.00

The output gives the Frequency per response however gives the sum of Count variable.

Types of Data

Discrete Data – They always have distinct & finite values. (Pin code, phone number, room number etc.)

Continuous Data – They are based on scale, they can take infinite values. (Height, Age, Time)

Character values are always discrete and finite values.(Gender, Color etc.)

E.g. of a Discrete and continuous variable with weight statement:

Gender is a discrete character value and Fee is a continuous numeric value:

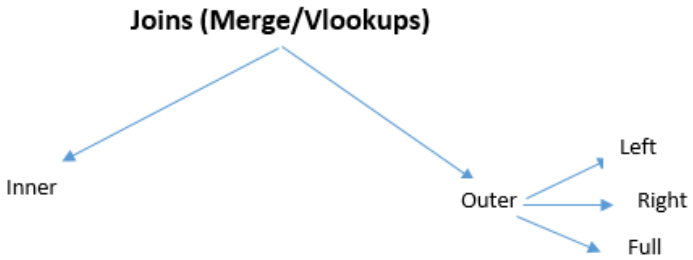
```
proc freq data =sasuser.admit;
tables sex;
weight fee;
run;
```

Output:

```
The FREQ Procedure
```

Sex	Frequency	Percent	Cumulative Frequency	Cumulative Percent
F	1418.35	52.79	1418.35	52.79
M	1268.6	47.21	2686.95	100.00

The output has been grouped by gender (F & M) with sum total of variable Fee.



Features of SQL join

- Needs a Primary key
- Primary key variable and any other valriable can be different (doesn't require to be renamed like in Base SAS)
- Doesn't require sorting
- Does not overwrite common variables between tables

Q -Why SQL doesn't require sorting?

A - Because of Cartesian product

Concept of Cartesian Product: Cartesian product result-set contains the number of rows in the first table, multiplied by the number of rows in second table.

A	B
ID	ID
1	3
2	4

Proc sql;

Select * from A,B;

Quit;

Output

ID	ID
1	3
1	4
2	3
2	4

Join is a sub-set of Cartesian product and Cartesian Product is a superset.

Q – How to create a table with this data?

```
Proc sql;
Select * from A,B;
Create table C as select a. ID, b.ID as ID1 from a,b;          *Qualifying a column*
Quit;
```

Output

ID	ID1
1	3
1	4
2	3
2	4

Inner Join

A		B	
ID	Name	ID	Sal
1	A	1	100
2	B	7	200

Base SAS code

```
Data C;
Merge A (In=x)    B (In=y);
By ID;
If x and y;
Run;
```

SQL code

```
Proc sql;
Select A.* , B,* from A,B where A.ID=B.ID;
Quit;
```

Output

ID	Name	Sal
1	A	100

Can also be written as

Proc sql;

Select A.*, B,* from A **inner join** B **on** A.ID=B.ID;

Quit;

If the primary key name is different (table A = ID, table B = PID)

Proc sql;

Select A.*, B,* from A, B where ID=PID;

Quit;

```
Data A;
input ID Name$ Age;
cards;
1 A 55
2 B 88
; run;

data B;
Input ID Sal Age;
cards;
7 100 20
1 200 90
;run;

Proc Sql;
select a.ID,a.Name,b.age,sal from A, B where A.ID =B.ID;
quit;
```

Output:

ID	Name	Age	Sal
1	A	90	200

From the above example, it is proved that SQL doesn't require sorting because it needs **Cartesian product** which gets optimized by the where clause, therefore it only picks those rows where primary key of one table is equal to primary key of another table.

****Here the Age variable was not overwritten (Table B age by Table A age), the Age variable is 90 because we specified Age as B.age, hence it picked age as 90.

Note: SQL never overwrites common variable between the tables, we would need to specify the table name with variable to get the same.

More Examples of inner join:

City wise count of designation using **group by**:

```
Data A;
Input ID city $;
Cards;
1 G
2 G
3 N
; run;

Data B;
Input Id des $;
Cards;
1 VP
2 VP
3 AVP
7 VP
;run;

proc sql;
select city,des,count (*) as Count from a,b where a.id =b.id
group by city,des;
quit;
```

Output:

City	des	Count
G	VP	2
N	AVP	1

Same query using in-line view concept:

```
proc sql;
```

```
Select City,des,Count(*) as Count from
(
select a.id,city,des from a,b where a.id =b.id
)group by city, des;
quit;
```

Scenario 2

City names changed from G to GUR and N to Noi (**using Case, When-Then**), City wise, Designation count of non-absconding using **in-line view** with **group by**

Here we are joining three Tables to get our desired output:

Code

```
Data A;
input ID city $;
cards;
1 G
2 G
3 N
; run;


Data B;
input id des $ abs;
cards;
1 VP 0
2 VP 0
3 AVP 0
7 VP 1
;run;

Data dep;
input id dep$;
cards;
1 HR
2 HR
3 IT
4 IT
;run;


proc sql;
select case when city ="G" then "Gur" else "Noi" end as
city,des,dep,Count from
(
select city,des, dep, count (*) as Count from a,b,dep where a.id
=b.id=dep.id and abs =0
```

```
group by city,des,dep);
quit;
```

Output:

```
City  des       dep          Count
-----------------------------------
Noi   AVP       IT               1
Gur   VP        HR               2
```

How to Sort data row-wise

Any numeric and character values can be sorted row-wise with the help of Call Routine statements as **Call SortN** and **Call sortC**.

What is a **Call Routine**?

A **CALL routine** alters variable values or performs other system functions. **CALL routines** are similar to functions, but differ from functions in that you cannot use them in assignment statements or expressions. All **SAS CALL routines** are invoked with **CALL** statements.

Call SortN and Call SortC

The values of variable are sorted in ascending order by the **CALL SORTN** routine. Comparisons. The **CALL SORTN** routine is used with numeric variables, and the**CALL** SORTC routine is used with character variables.

Practical E.g.

```
Data A;
input ID1 ID2 ID3 ID4;
Call SortN (of ID1-ID4);**SortN(of ID4 - ID1) descending order**
cards;
1 5 3 1
3 2 1 17
1 2 31 11
;run;

Proc Print data =A;
run;
```

Output:

Obs	ID1	ID2	ID3	ID4
1	1	1	3	5
2	1	2	3	17
3	1	2	11	31

E.g. for Character values

```
Data A;
input Name1 $ Name2 $ Name3$ Name4$;
Call SortC (of Name1-Name4);
cards;
C A R K
H B A N
S O F A
;run;

Proc Print data =A;
run;
```

Output:

Obs	Name1	Name2	Name3	Name4
1	A	C	K	R
2	A	B	H	N
3	A	F	O	S