**PROC SQL**

* Proc sql == SAS +SQL
* Syntax:

*proc sql;*

*#Content;*

*Quit*;

//there is no need of **run** in an proc sql statements

//there can be **many** statements in the #Content

* SELECT Statement is generally a query
* More usual proc sql statement

**PROC SQL**;

**Select** column\_name1,column2 ..

From DATASET

<**Where** clause> //Where clause filters the data on which the operation will be done

<**Group** by clause> //groups by the respective cuts

<**Having** clause> // it works along **with group by** clause only and it **filters** the groups c created by the “group by” clause

<**Order** by clause> **;** //Sorts the output

**QUIT;**

* **describe table ===** proc contents ..it does the same things just a way of proc contents via sql

Example:

Proc sql;

***describe table*** *abc***;** //similar to describe there are other statements (other than select w which could be used with proc sql (even proc sql can be used for proc print)

quit;

* proc sql **outobs** (10) /// outobs actually filters the no of outcomes

select \* from abc quit;

* proc sql;

select FirstName, LastName, UserID

from sq.customer(obs=100); //by this method we restrict the no of entries inputed from data

quit;

* **yrdiff(dob, ‘21jan1992’d) //**to take dates difference in select clause one can use (used with select statement
* **calculated** //keyword that can help filtering out on the basis of dynamic variable created in the select clause

proc sql;

select FirstName, LastName, UserID,yrdif(dob,'01jan2019'd) as age

from sq.customer

where age >= 70;

quit;

/\*the output is errored\*/

/\*But if the expression is written in the below format\*/

proc sql;

select FirstName, LastName, UserID,yrdif(dob,'01jan2019'd) as age

from sq.customer

where **calculated** age >= 70;

quit;

* **timepart** //function it works similar to datepart the only thing is it separates out time from date time format
* **insert into** command in proc sql /// inserting rows into an already existing data

proc sql;

insert into data (firstname, lastname, UserID, CreditScore)

select firstname, lastname, UserID, CreditScore

from sq.customer

where CreditScore > 780;

quit;

//ALTER, TABLE, UPDATE, DELETE can be used similar to insert into command

* ***Like* in proc sql //**creates empty data with formats ie columns as in sq.customer

Proc sql;

Create table **like** sq.customer (keep = firstname, lastname etc);

Quit;

* DICTIONARY TABLES

//these tables are dynamically made inbuilt by the current sas session

//we can use these tables to check whether the data sets have same format columns names say

//these are only read only dynamic data (cant be saved are only view only)

**-DICTIONARY.tables**

**- DICTIONARY.columns**

**- DICTIONARY.libnames**

For example:

proc sql;

describe table dictionary.columns; //similar to pro contents in sas

select \*

from dictionary.columns //the **dynamic columns level** detail along with the data names

where libname = "SQ";

quit;

* Coalesce //this function is used in full joins when selecting the non zero column as entry in our query

//coalesce(column1,column2) returns the 1st non null value

//(say if column1 is null….coalesce will return column2 otherwise column1)

* A subquery can either return a value or a column (it can never return more than one column)

(always first calculate the subquery and then use its value in the main query)

* #**CREATE VIEW**

Proc sql;

**Create view** sq.totalcustomer as //in this create view there is no data but only query is defined which could be called or referenced in various other queries

Select state , count(\*) as Totalcustomer

From sq.customer

Group by State;

Quit;

//benefits of a view …. No temporary data is stored anywhere…..can be called multiple times in various other queries //it is similar to %Let

* **SET Operators in proc sql**

A Set operator basically appends the intermediate queries to finally present the outcome from more than 1 tables without appending the tables (just appends the summary of the original tables and not the data appending)

There are majorly 4 set operators.

1. **Intersect**
2. **UNION**
3. **EXCEPT**
4. **OUTER UNION** (only available in sas and is generally not in sql)

The first three operators outputs unique one line entry in the output . we can use ALL keyword for keeping even the repetitive results..

Also the first 3 operators align the output in the sequence of columns . ie it appends assuming the 1st column of query 1 is same as 1st column of query 2

The UNION set operator works in a different order than the INTERSECT and EXCEPT operators. The UNION set operator first combines results sets, then removes duplicate rows. INTERSECT and EXCEPT remove duplicate rows, then combine result sets. If the two intermediate result sets have a different number of columns, then SAS extends one table with null columns so that the two intermediate result sets will have the same number of columns. If result set 1 is extended with null columns, then the name of the column in result set 2 will be used in the final results. In these cases, SAS writes a note in the log.

For example

proc sql;

select \*

from sq.salesphone (rename = (PhoneResp = Resp))

**outer union corr**

select \*

from sq.salesemail (rename = (EmailResp = Resp))

order by 1, Resp ;

quit;

In the above code outer union is used (which does not removes duplicates ) And **corr** actually appends on the basis of column name

#Using intersect //which actually selects the common part and also removes duplicates

proc sql;

select \*

from sq.salesemail

**intersect**

select \*

from sq.salesphone;

quit;

Remember **queries are appended** not the data, so we can be safe from altering the available data.

**User defined macro variables in proc sql**

1. **%Let** variable = value

The value can be a text or a number

Lets say %Let dummy= Mommy

And want the text “Mommy”

Then we can write it as ”&dummy” (Note: we always need to use **double quotes** whenever we are calling a macro variable inside a text)

1. By storing a query in one macro variable by which we will be able to use that summarized value in any text etc

Syntax:

Proc sql;

Select sum(Income)

into :macro\_variable\_name

From data;

Quit;

We can use the macro variable created in numerous queries and text

We use “**&**” operator to refer to the value of the macro variable

For eg in the above query we refer the macro variable name by

**&macro\_variable\_name**

Whenever we use it in text always use double quotes and not single quotes.

Note : we can make more than one macro variables from one query for eg

Proc sql;

Select sum(Income), count(\*)

into :macro\_variable\_name1 , :macro\_variable\_name2

From data;

Quit;

//value of count \* will be assigned to macro\_variable\_name2.