/\* PROC SQL is an advanced SAS procedure that helps to run SQL queries to manage and manipulate data.

It includes Introduction of SQL with examples, PROC SQL Joins, conditional statements and useful tips

and tricks of SQL etc.

\* The difference between SAS and SQL terminology is shown in the below.

SAS SQL

Dataset Table

Observation Rows

Variable Columns

\* SYNTAX

PROC SQL;

SELECT column(s)

FROM table(s) | view(s)

WHERE expression

GROUP BY column(s)

HAVING expression

ORDER BY column(s);

QUIT;

The SELECT statement must be specified in the following order:

1.SELECT

2.FROM

3.WHERE

4.GROUP BY

5.HAVING

6.ORDER BY

Note: Only the SELECT and FROM clauses are required. All the other clauses are optional.

EXPLANATION

PROC SQL: calls the SQL procedure

SELECT: specifies the column(s) (variables) to be selected

FROM: specifies the table(s) (data sets) to be queried

WHERE: subsets the data based on a condition

GROUP BY: classifies the data into groups based on the specified column(s)

ORDER BY: sorts the resulting rows (observations) by the specified column(s)

QUIT: ends the PROC SQL procedure

We are going to look at the difference between Non-SQL Base SAS and PROC SQL.

Functions SAS SQL

Drop Columns Drop Select

Rename Column Rename As

Add Row Output Insert Into

Delete Rows Where/ Then delete where/ Delete from

delete duplicate rows nodup distinct

Create a table data create table

sorting proc sort order by

summarize data proc means group by

conditional statements if-then case-when

Displaying Output Proc Print Select

Concatenating Set Outer Union Corr

\*/

data mylib.outdata ;

set SASHELP.BWEIGHT (obs=1000);

run;

\* Removing labels from datasets;

proc datasets library=mylib nolist;

modify outdata;

attrib \_all\_ label='';

quit;

\* PROC SQL STATEMENTS;

\* 1. Selecting all variables from the data set;

proc sql;

select \* from mylib.outdata;

Quit;

\* Asterisk (\*) is used to select all columns (variables) in the order in which they are

stored in the table.;

\* Outdata is the table (data set) from which we need to select the columns (variables) .

It is stored in MYLIB library.;

\* To display the list of columns to the SAS log, use FEEDBACK option in the PROC SQL statement;

proc sql feedback;

select \* from mylib.outdata;

Quit;

\* See log window;

\* 2. Selecting specific variables from the data set;

\* In the SELECT clause, multiple columns are separated by commas.;

proc sql;

select weight,married from mylib.outdata;

Quit;

\* In the SELECT clause, Weight and Married columns (variables) are specified so that we can

select them from OUTDATA table (data set).;

\* 3. Limiting the number of rows;

\* Suppose you want to limit the number of rows (observations) that PROC SQL displays, use the

OUTOBS= option in the PROC SQL statement.;

proc sql outobs=5;

select weight,married from mylib.outdata;

Quit;

\* 4. Renaming a variable in output;

\* Suppose you want to rename a variable, use the column alias AS option in the PROC SQL statement.;

proc sql;

select weight,married as marriage\_flag from mylib.outdata;

Quit;

\* The variable name has been renamed from married to marriage\_flag.

\* 5. Creating a new variable;

\* Suppose you want to create a new variable that contains calculation.;

proc sql;

select weight, (weight\*0.5) as newweight from mylib.outdata;

Quit;

\* A new variable has been created and named newweight which is calculated on the basis

of the existing variable weight.;

\* 6. Referring to a previously calculated variable;

\* The keyword CALCULATED is used to refer a previously calculated variable.;

proc sql;

select weight, (weight\*0.5) as newweight, CALCULATED newweight\*0.25 as revweight

from mylib.outdata;

Quit;

\* 7. Removing duplicate rows;

\* The keyword DISTINCT is used to eliminate duplicate rows (observations) from your query results.;

\* In the following program, we are asking SAS to remove all those cases where in duplicates exist

on combination of both the variables - weight and married.;

proc sql;

select DISTINCT weight, married from mylib.outdata;

quit;

\* The DISTINCT \* implies cases having same values in all the variables as a whole would be removed.;

proc sql;

select DISTINCT \* from mylib.outdata;

quit;

\* 8. Formatting variables;

\* SAS-defined formats can be used to improve the appearance of the body of a report.;

proc sql;

select weight FORMAT= dollar8.2, married from mylib.outdata;

Quit;

\* 9. Sorting data The ORDER BY clause returns the data in sorted order.;

\* ASC option is used to sort the data in ascending order. It is the default option.;

\* DESC option is used to sort the data in descending order;

proc sql;

select MoMAge, weight, married from mylib.outdata

ORDER BY weight ASC, married DESC;

Quit;

/\* 10. Subsetting data with the WHERE clause

Use the WHERE clause with any valid SAS expression to subset data.

List of conditional operators :

1. BETWEEN-AND

The BETWEEN-AND operator selects within an inclusive range of values.

Example : where salary between 4500 and 6000;

2. CONTAINS or ?

The CONTAINS or ? operator selects observations by searching for a specified set of characters within the values of a character variable

Example : where firstname contains "DE"; OR where firstname ? "DE";

3. IN

The IN operator selects from a list of fixed values.

Example : where state = 'NC' or state = 'TX';

The easier way to write the above statement would be to use the IN operator

where state IN ('NC','TX');

4. IS MISSING or IS NULL

The IS MISSING or IS NULL operator selects missing values.

Example : where dateofbirth is missing OR where dateofbirth is null

5. LIKE

The LIKE Operator is used to select a pattern.

a) underscore(\_) - any single char

b) percent sign(%) - any number of char

\*/

\* 11. Multiple Conditions / Criteria - Case-When;

PROC SQL;

SELECT WEIGHT,

CASE

WHEN WEIGHT BETWEEN 0 AND 2000 THEN 'LOW'

WHEN WEIGHT BETWEEN 2001 AND 3000 THEN 'MEDIUM'

WHEN WEIGHT BETWEEN 3001 AND 4000 THEN 'HIGH'

ELSE 'VERY HIGH'

END AS NEWWEIGHT

FROM mylib.outdata;

QUIT;

\* The END is required when using the CASE.

The following operators can be used in CASE expression:

• All operators that IF uses (= , <, >, NOT, NE, AND, OR, IN, etc)

• BETWEEN AND

• CONTAINS or ‘?’

• IS NULL or IS MISSING

• = \*

• LIKE

;

\* 12. Aggregating or summarizing data;

\* Use GROUP BY clause to summarize data. Summary functions are used on the SELECT statement to

produce summary for each of the analysis variables.;

proc sql;

select momage, COUNT(married) AS marriage from mylib.outdata GROUP BY momage;

Quit;

\* The summary functions available are listed below:

AVG/MEAN

COUNT/FREQ/N

SUM

MAX

MIN

NMISS

STD

VAR

T (t value)

USS (Uncorrelated Sum of Square)

CSS (Correlated Sum of Square)

RANGE

;

\* 13. Subsetting data in the groups

In order to subset data when grouping is in effect, the HAVING clause must be used. The

variable specified in having clause must contain summary statistics.;

proc sql;

select momage, weight, COUNT(married) AS marriage from mylib.outdata

GROUP BY momage, weight HAVING marriage > 3;

Quit;

\* Important Point - The WHERE clause cannot be used to subset aggregated data. To subset data

with the GROUP BY clause you must use HAVING clause.;

/\* 14. Creating a new data set as output

The CREATE TABLE statement can be used to create a new data set as output instead of a

report produced in output window.

SYNTAX

PROC SQL;

CREATE TABLE table-name AS

SELECT column(s)

FROM table(s) | view(s)

WHERE expression

GROUP BY column(s)

ORDER BY column(s);

QUIT;

\*/

proc sql;

create table health AS select weight, married from mylib.outdata

ORDER BY weight ASC, married DESC;

Quit;

\* 15. Limiting the number of rows in the new created data set

Suppose you want to limit the number of rows (observations) that PROC SQL produces in the data

set, use the INOBS= option in the PROC SQL statement.;

proc sql INOBS=50;

create table health AS select weight,married from mylib.outdata;

Quit;

\* Difference between INOBS= and OUTOBS=

INOBS controls how many records are read from the dataset and OUTOBS controls how many

records are written. Run the following program and see the difference. Both returns

different results.;

\* OUTOBS=Example;

proc sql outobs=2;

select age, count(\*) as tot from sashelp.class group by age;

quit;

\* INOBS= Example;

proc sql inobs=4;

select age, count(\*) as tot from sashelp.class group by age;

quit;

/\* 16. Counting unique values by a grouping variable

Suppose you are asked to calculate the unique number of age values by gender columns using

SASHELP.CLASS dataset

You can use PROC SQL with COUNT(DISTINCT variable\_name) to determine the number of unique

values for a column.

\*/

PROC SQL;

CREATE TABLE TEST1 as SELECT Sex, Count(distinct Age) AS Unique\_count FROM sashelp.class GROUP BY Sex;

QUIT;

/\* 17. Count the number of missing values

You can use NMISS() function to compute the number of missing values in a variable. The COUNT()

function returns the number of non-missing values in a variable.

\*/

data temp;

input id;

cards;

1

2

.

4

5

.

;

run;

proc sql;

select nmiss(id) as N\_missings, count(id) as N, calculated N\_missings + calculated N as total

from temp;

quit;

/\* 18. KEEP and DROP some variables

Suppose you need to keep all the variables from SASHELP.CARS except variables 'MODEL' and 'MAKE'.

The DROP= option is used to drop these two variables. Similarly, we can use KEEP= option to keep

specific variables. These DROP= and KEEP= Options are not native SQL language. It only works in SAS.

\*/

proc sql;

create table saslearning (drop= make model) as select \* from sashelp.cars;

quit;

/\* 19. Delete Rows from a Table

You can use DELETE FROM statement to remove records (rows) from a dataset. \*/

proc sql;

delete from mylib.outdata where momage > 0;

quit;

\* In this case, we are deleting all records having momage greater than 0 from outdata dataset.

Log shows '478 rows were deleted from MYLIB.OUTDATA'.;

/\* 20. Sub Query

Find employee IDs who have records in table file1 but not in table file2.\*/

data file1;

input ID age;

cards;

1 24

2 34

3 45

4 67

;

run;

data file2;

input ID age;

cards;

1 25

3 46

4 62

;

run;

Proc SQL;

Select ID from file1 Where ID not in (select ID from file2);

Quit;

\* 21. Sub Query - Part II;

\* Find employee IDs whose age is in the average age +/- 10 years.;

Proc SQL;

Select id from file1

where age between (select Avg(age) from file1) - 10 and (select avg(age) from file1)+10;

Quit;

/\* PROC SQL : JOINS

Advantages of PROC SQL Joins over Data Step Merging

a) PROC SQL joins do not require sorted tables (data sets), while you need to have two

data sets sorted when using MERGE statement

b) PROC SQL joins do not require that common variable have the same name in the data

sets you are joining, while you need to have common variable name listed in BY option when using MERGE statement.

c) PROC SQL joins can use comparison operators other than the equal sign (=).

d) PROC SQL can handle many to many relationship well whereas Data Step Merge do not.

\*/

/\* 1. Cross Join / Cartesian product

The Cartesian product returns a number of rows equal to the product of all rows

(observations) in all the tables (data sets) being joined. For example, if the first table

has 10 rows and the second table has 10 rows, there will be 100 rows (10 \* 10) in the

merged table (data set).

Create these two data sets into SAS \*/

Data A;

Input ID Name$ Height;

cards;

1 A 1

3 B 2

5 C 2

7 D 2

9 E 2

;

run;

Data B;

Input ID Name$ Weight;

cards;

2 A 2

4 B 3

5 C 4

7 D 5

;

run;

\* Cartesian Product : SQL Code;

PROC SQL;

Create table dummy as

Select \* from A as x cross join B as y;

Quit;

Proc Print data=dummy;

run;

/\* Cartesian or Cross Product

Key takeaways

1. Since the first data set has 5 rows and the second data set has 4 rows, there are

20 rows (5 \* 4) in the merged data set.

2. The 'as' keyword (aka alias) is used to assign a table a temporary name.

3. Since the ID values of the first data set is different than the ID values of the

second data set, the ID given in the joined data set is misleading.

\*/

/\* 2. Inner Join

It returns rows common to both tables (data sets). If we select \* keyword in the query,

the final merged file would have number of columns equal to (Common columns in both the

data sets + uncommon columns from data set A + uncommon columns from data set B).

\*/

PROC SQL;

Create table dummy as

Select \* from A as x, B as y

where x.ID = y.ID;

Quit;

Proc Print data=dummy;

run;

/\* Inner Join

Since the above case is of type INNER JOIN, it returns values 5 and 7 from the variable

ID in the combined table as these two values are common in both the datasets A and B

\*/

\* Another way to write the above code ;

PROC SQL;

Create table dummy as

Select \* from A as x inner join B as y

On x.ID = y.ID;

Quit;

Proc Print data=dummy;

run;

\* Both the codes produce same result.;

\* 3. Left Join - It returns all rows from the left table with the matching rows

from the right table.;

PROC SQL;

Create table dummy as

Select \* from A as x left join B as y

On x.ID = y.ID;

Quit;

Proc Print data=dummy;

run;

\* Left Join

Since the above case is of type LEFT JOIN, it returns all rows from the table

(dataset) A with the matching rows from the dataset B.;

\* 4. Right Join

It returns all rows from the right table that do not match any row with the left-hand

table, and the matched rows from the left-hand table.;

PROC SQL;

Create table dummy as

Select \* from A as x right join B as y

On x.ID = y.ID;

Quit;

Proc Print data=dummy;

run;

\* Right Join

Note : The right-hand table ID values are missing in the merged table.

To add the missing right hand table ID values to a right join, you can use the SQL

COALESCE function. The COALESCE function returns the first non-missing argument.;

proc sql;

create table dummy as

select coalesce (x.ID,y.ID) as ID, coalesce (x.name,y.name) as name,height,weight

from a as x right join b as y

on x.id = y.id;

quit;

Proc Print data=dummy;

run;

\* Right Join with Coalesce

Since the above case is of type RIGHT JOIN, it returns all rows from the table

(dataset) B with the matching rows from the dataset A.;

/\* 5. Full Join

It returns all rows from the left table and from the right table.

The FULL JOIN suffers the same difficulty as the RIGHT JOIN. Namely, the common variable

values are lost from the right-hand data set. The COALESCE function can solve this difficulty.

\*/

proc sql;

create table dummy as

select coalesce (x.ID,y.ID) as ID, coalesce (x.name,y.name) as name,height,weight

from a as x full join b as y

on x.id = y.id;

quit;

\* Since the above case is of type FULL JOIN, it returns all rows from the dataset A and B.;