\* COMBINING TABLES VERTICALLY WITH PROC SQL;

data dat1;

input x y;

cards;

1 6

1 6

1 7

6 4

7 6

8 7

;

run;

data dat2;

input x z;

cards;

1 5

4 2

3 4

6 4

6 5

5 8

;

run;

/\* 1. UNION Operator

It displays all rows from both the tables and removes duplicate records from the combined

dataset. By adding ALL keyword, it allows duplicate rows in the combined dataset.

Important Point

a) UNION is performed by position not by column name. Hence, common columns in each SELECT

statement should be in the same order. If CORR keyword is included, PROC SQL matches the

columns by name.

b) ALL Keyword - ALL keyword allows duplicates in the concatenated dataset.

c) CORR Keyword - CORR keyword tells SAS to match the columns in table by name and not

by position. Columns that do not match by name are excluded from the result table, except

for the OUTER UNION operator

\*/

\* UNION Operator;

proc sql;

create table out7 as

select \*

from dat1

UNION

select \*

from dat2;

quit;

Proc print data=out7;

run;

proc sql;

create table out8 as

select \*

from dat1

UNION ALL

select \*

from dat2;

quit;

Proc print data=out7;

run;

proc sql;

create table out9 as

select \*

from dat1

UNION CORR

select \*

from dat2;

quit;

Proc print data=out9;

run;

/\* 2. OUTER UNION CORR

It appends (concatenates) two tables. It is equivalent to SET statement in Data Step.

It allows duplicates in the concatenated table. The ALL keyword is not required with

OUTER UNION.

\*/

proc sql;

create table out10 as

select \*

from dat1

OUTER UNION CORR

select \*

from dat2;

quit;

Proc print data=out10;

run;

/\* 3. Except Operator

It returns unique rows from the first query that are not found in the second query.

(Non matched Rows). It removes duplicate records (where all columns in the results are

the same) - row 2nd in table1.

\*/

\* Except Operator;

proc sql;

create table out1 as

select \*

from dat1

EXCEPT

select \*

from dat2;

quit;

Proc print data=out1;

run;

\* Except ALL - It allows duplicate records in the combined dataset and does not remove duplicates.;

proc sql;

create table out2 as

select \*

from dat1

EXCEPT ALL

select \*

from dat2;

quit;

Proc print data=out2;

run;

/\* Except CORR - It displays only columns that have the same name (or common) in both the tables.

Except CORR - It returns all unique rows in the first table (based on the common column) that do not

appear in the second table.

\*/

proc sql;

create table out3 as

select \*

from dat1

EXCEPT CORR

select \*

from dat2;

quit;

Proc print data=out3;

run;

\* 4. INTERSECT Operator - It selects unique rows that are common to both the tables.;

proc sql;

create table out5 as

select \*

from dat1

INTERSECT

select \*

from dat2;

quit;

Proc print data=out5;

run;

\* PROC SQL : INSERT INTO;

data temp;

set sashelp.class;

run;

Proc print data=temp;

run;

/\* 1. Insert Rows based on Column Position

With the VALUES clause and INSERT statement, we can assign values to columns by their positions. In

the example below, "Sam" would be added to the first column, "M" added to the second column, "28"

added to the third column and so on. Multiple VALUES clauses implies multiple rows to be added into

the table.

\*/

PROC SQL;

INSERT INTO temp

VALUES ("Sam","M",28,75,100)

VALUES ("Sam2","M",58,55,70);

QUIT;

\* Log Window: NOTE: 2 rows were inserted into WORK.TEMP.;

Proc print data=temp;

run;

/\* 2. Insert Rows based on Column Name

We can also define columns and values assigned to them only. Values of all the columns that are

not defined would be assigned missing.

\*/

PROC SQL;

INSERT INTO temp (name,age)

VALUES ("Sam",20);

QUIT;

\* Log window: NOTE: 1 row was inserted into WORK.TEMP.;

Proc print data=temp;

run;

/\* 3. Insert Rows with a Query

We can also add rows with a query. In the example below, we are appending rows to the table by

extracting data from the other table.

\*/

proc sql;

insert into temp

select \* from sashelp.class

where age > 15;

quit;

Proc print data=temp;

run;

/\* 4. Create Sample Data with PROC SQL

The DATALINES statement with an INPUT statement in DATA STEP is used to read data that you enter

directly in the program. In PROC SQL, you can do the same with CREATE TABLE and INSERT INTO statement.

\*/

proc sql;

create table list

(ID num(10), Gender char(1),Salary num,

DateOfBirth num informat=date7. format=date7.);

insert into list

values(12345,'F',42260,'21JAN01'd)

values(23456,'M',61090,'26JAN54'd);

quit;

Proc print data=list;

run;

/\* 5. Add Constraints in the Table

We are adding constraints that values of ID variable should be unique (Primary Key), "area" variable

contain only two values - USA and India, samplesize should be greater than 0.

\*/

proc sql;

create table example

(ID num(15),

samplesize num,

area char(15) NOT NULL,

constraint prim\_key primary key(ID),

constraint samplesize check(samplesize gt 0),

constraint area check(area in ('USA', 'India')));

quit;

\* Let's insert two rows;

proc sql;

insert into example

values(12345,42260,'India')

values(12345,61090,'USA');

quit;

\* It returns error due to duplicate values in a variable that have a constraint of primary key.;

\* 6. Create a blank table - We can create a blank table by copying the structure of existing table;

PROC SQL;

CREATE TABLE EXAMPLE2 LIKE TEMP;

QUIT;

/\* 7. See the structure of table

The DESCRIBE table is an alternative to PROC CONTENTS. It displays the structure of table - how table

was created and format of variables. \*/

PROC SQL;

DESCRIBE TABLE EXAMPLE2;

QUIT;

/\* PROC SQL : ALTER TABLE AND UPDATE COLUMN;

The ALTER TABLE statement is used to add new columns, delete existing columns or modifying the

format of columns.

The UPDATE statement is used to modify existing column values in a table.

\*/

data temp;

set sashelp.class;

run;

\* ALTER TABLE Syntax

ALTER TABLE table-name

ADD CONSTRAINT constraint-name constraint-definition

ADD column-definition

DROP CONSTRAINT constraint-name

DROP column(s)

DROP FOREIGN KEY constraint-name

DROP PRIMARY KEY

MODIFY column-definition

;

\* Example 1 : Adding Columns;

\* In the following program, we are adding 3 columns - Section as character variable, TotalMarks

as numeric variable, DateOfBirth as Date format variable. The new columns would be blank.;

PROC SQL;

ALTER TABLE temp

ADD Section CHAR (10), TotalMarks NUM (8), DateOfBirth num informat=date7. format=date7.;

QUIT;

Proc Print Data=temp;

run;

\* Example 2 : Add Values in New Columns;

\* The UPDATE statement is used to add or update values in columns. In this case, we are updating

rows wherein age is less than 15.;

PROC SQL;

UPDATE temp SET Section='Section A', TotalMarks=100, DateOfBirth='22OCT99'D where age < 15;

QUIT;

Proc Print Data=temp;

run;

\* Example 3 : Conditional Update Statement;

\* We are adding 5 to column Height if age is less than or equal to 15. If age is greater than 15,

height should be added by 10. In other words, we are using IF THEN ELSE conditions in UPDATE statement.;

PROC SQL;

UPDATE temp

SET Height =

CASE WHEN age <= 15 THEN Height + 5

WHEN age > 15 THEN Height + 10

ELSE HEIGHT

END;

QUIT;

Proc Print Data=temp;

run;

\* Example 4 : Update Multiple Columns;

\* We can update multiple columns with UPDATE statement like the programs written below;

PROC SQL;

UPDATE temp SET Section='SectionB', DateOfBirth='22OCT02'D where age<15;

UPDATE temp SET Section='SectionA', DateOfBirth='22OCT99'D where age>=15;

QUIT;

Proc Print Data=temp;

run;

\* Example 5 : Modify the column attributes;

\* We can modify the column format with MODIFY statement.;

PROC SQL;

ALTER TABLE temp

MODIFY totalmarks DECIMAL(8,2) format=8.2;

quit;

Proc Print Data=temp;

run;

\* Example 6 : Delete Columns;

PROC SQL;

ALTER TABLE temp DROP totalmarks, section;

QUIT;

Proc Print Data=temp;

run;

\* Example 7 : Adding NOT NULL Constraint;

\* We are preventing missing values in a column using NOT NULL Contraint.;

PROC SQL;

ALTER TABLE TEMP

ADD CONSTRAINT NOT\_NULL\_WEIGHT NOT NULL(WEIGHT);

QUIT;

Proc Print Data=temp;

run;

\* Example 8 : Allowing only UNIQUE values;

\* We are not allowing duplicate values in a column.;

PROC SQL;

CREATE TABLE TEMP3

(ID NUM UNIQUE,

STATE CHAR(20));

QUIT;

Proc Print Data=temp3;

run;

\* Example 9 : Creating a Primary Key;

\* The PRIMARY KEY constraint uniquely identifies each record in a table.;

PROC SQL;

ALTER TABLE TEMP3

ADD CONSTRAINT PRIM\_KEY PRIMARY KEY (ID);

QUIT;

Proc Print Data=temp3;

run;

\* INTERMEDIATE PROC SQL TUTORIAL;

\* Example 1 : Count Cases Where Condition is TRUE;

data xyz;

input x y$ z$;

cards;

1 23 Y

1 24 N

1 25 Y

2 21 Y

2 22 Y

3 25 N

3 36 Y

;

run;

proc sql noprint;

create table tt as

select x,

sum(case when z= "Y" then 1 else 0 end) as z\_Y,

sum(case when z= "N" then 1 else 0 end) as z\_N

from xyz

group by x;

quit;

Proc Print data = tt;

run;

\* Example 2 : Creating trend variables;

data example1;

input ID Months Revenue Balance;

cards;

101 1 3 90

101 2 33 68

101 3 22 51

101 4 3 90

101 5 33 65

101 6 22 54

102 1 100 18

102 2 58 62

102 3 95 97

102 4 100 18

102 5 58 65

102 6 95 92

;

\* Task : Calculate total revenue and total balance accumulated in the first 3 months;

proc sql noprint;

create table output1 as

select ID,

sum(case when 1 <= Months <= 3 then Revenue else . end) as Rev\_1\_3,

sum(case when 1 <= Months <= 3 then Balance else . end) as Bal\_1\_3

from example1

group by ID;

quit;

Proc Print data = output1;

run;

\* Example 3 : Extract First and Last Observation within a Group;

\* PROC SQL : Alternative to First. Statement;

proc sql;

create table output2 (drop=n) as

select \*, monotonic() as n

from example1

group by ID

having min(n) = n;

quit;

Proc Print data=output2;

run;

\* PROC SQL : Alternative to Last. Statement;

proc sql;

create table output2 (drop=n) as

select \*, monotonic() as n

from example1

group by ID

having max(n) = n;

quit;

Proc Print data=output2;

run;

\* Example 4 : Select All Excluding 1 Variable;

proc sql;

create table want (drop=x) as

select a.\*,b.\*

from dat1 a, dat2 (rename=(id=x)) b

where a.id = b.x;

quit;

Proc Print data=want;

run;

/\* Example 5 : Sub Query - ANY and ALL Operators

1. ANY operator selects values that pass the comparison test with any of the values that are

returned by the sub-query.

2. ALL operator selects values that pass the comparison test with all of the values that are returned

by the sub-query.

\*/

data jansale;

input sale id;

cards;

100 1

105 2

108 3

110 4

;

run;

data febsale;

input sale id;

cards;

120 1

105 2

118 3

117 4

;

run;

proc sql;

select \*

from jansale

where sale < all (select sale from febsale);

quit;

proc sql;

select \*

from jansale

where sale < any (select sale from febsale);

quit;

\* PROC SQL SELF JOIN TUTORIAL;

\* Example 1 : Find out Manager

Suppose you have data for employees. It comprises of employees' name, ID and manager ID. You need to

find out manager name against each employee ID.;

data example2;

input Name $ ID ManagerID;

cards;

Smith 123 456

Robert 456 .

William 222 456

Daniel 777 222

Cook 383 222

;

run;

proc sql;

create table want as

select a.\*, b.Name as Manager

from example2 as a left join example2 as b

on a.managerid = b.id;

quit;

proc print data=want;

run;

\* Example 2 : Find out Manager of Manager;

\* Suppose you have data for employees. It comprises of employees' name, ID and manager ID. You need to

find out manager of manager's name against each employee ID.;

data example22;

input Name $ ID ManagerID;

cards;

Smith 123 456

Robert 456 777

William 222 123

Daniel 777 .

Cook 383 456

;

run;

proc sql;

create table want as

select a.Name, a.ID, a.managerid, b.ManagerID as ManagerofManagerID

from example22 a left join example22 b

on a.managerid = b.id;

quit;

Proc print data=want;

run;

proc sql;

create table want2 as

select a.\*, b.Name as ManagerofManagerName

from want as a left join want as b

on a.ManagerofManagerID = b.id;

quit;

Proc print data=want2;

run;

\* Example 3 : Find out Grand Son;

data example22;

input Parent $ Child $;

cards;

Aanad Vikas

Ankur Ram

Vikas Chandan

Chandan Dan

Ram Kush

;

run;

proc sql;

create table want as

select a.Parent as GrandParent, b.Child as GrandChild

from example22 a left join example22 b

on a.child = b.parent;

quit;

Proc print data=want;

run;

\* SAS SQL : FIND RECORDS ONLY EXIST IN ONE TABLE BUT NOT OTHER;

data dataset1;

input name $;

cards;

Dave

Ram

Sam

Matt

Priya

;

run;

data dataset2;

input name$;

cards;

Ram

Priya

;

run;

\* In SQL, there are multiple ways to solve this problem. The methods are listed below -;

/\* Method I - NOT IN Operator

The simplest method is to write a subquery and use NOT IN operator, It tells system not to include

records from dataset 2.

\*/

proc sql;

select \* from dataset1

where name not in (select name from dataset2);

quit;

/\* Method II - LEFT JOIN with NULL Operator

In this method, we are performing left join and telling SAS to include only rows from table 1 that do

not exist in table 2.\*/

proc sql;

select a.name from dataset1 a

left join dataset2 b

on a.name = b.name

where b.name is null;

quit;

/\* How it works -

In the first step, it reads common column from the both the tables - a.name and b.name. At the second

step, these columns are matched and then the b.name row will be set NULL or MISSING if a name exists

in table A but not in table B. At the next step, WHERE statement with 'b,name is null' tells SAS to

keep only records from table A.

\*/

\* NODUPKEY WITH PROC SQL;

\* This tutorial explains how to remove duplicates by a column but returns all the columns.;

data readin;

input ID Name $ Score;

cards;

1 David 45

1 David 74

2 Sam 45

2 Ram 54

3 Bane 87

3 Mary 92

3 Bane 87

4 Dane 23

5 Jenny 87

5 Ken 87

6 Simran 63

8 Priya 72

;

run;

Proc Print data=readin;

run;

\* Solution -Suppose you want to remove duplicates based on name but returns all the variables.;

proc sql noprint;

create table tt (drop = row\_num) as

select \*, monotonic() as row\_num

from readin

group by name

having row\_num = min(row\_num)

order by ID;

quit;

Proc Print data=tt;

run;