Simple Proc Sql Code;

```
proc sql;
select make, type, origin, enginesize,
  msrp-invoice as Difference
from sashelp.cars
where enginesize le 2
order by enginesize, make;
```

*(or use **order by 4,1;** or **enginesize, 1;** or **4,make;** as per the column references in the select clause)*

Querying Multiple tables; (Simple Join)

```
proc sql;
select patdata.obs, patdata.sex, patdata.age, patientdata.ht, patientdata.wt
from newlib.patdata, newlib.patientdata
where patdata.obs=patientdata.obs
order by age;
```

(column references required cz columns exist in more than one table)

Summarizing Data;

```
proc sql;
select customer, sum(revenue) as Earnings
from sashelp.electric
group by customer;
```

Simple Create Table statement;

```
proc sql;
create table work.summary as     (as keyword is used to create a table from the results of a query)
select customer, sum(revenue) as Earnings
from sashelp.electric
group by customer;     (does not create a report only a table)

proc sql;
select*from work.summary;     (this step is used to create a report)
```

Having Clause;

```
proc sql;
select customer, sum(revenue) as Earnings, year
from sashelp.electric
group by customer, year
having earnings > 100     (column alias used in having clause)
order by customer;
```

Feedback Option;

```
proc sql feedback;     (debugging tool letting us see exactly what is being submitted to sql processor)
select*from sashelp.shoes;
```

Distinct Option;

```
proc sql;
select distinct region, stores, product     (eliminates duplicate rows from all columns)
from sashelp.shoes
order by 1;
```

Between-And & Contains operator;

```
proc sql;
select * from sashelp.shoes
where sales between 100000 and 300000
    and product not contains 'Dress'
order by 7;
```

IN operator and outobs= option;

```
proc sql outobs=50;     (restrict the no. of rows that are displayed but not that are read)
select * from sashelp.shoes
where region in ('Africa', 'Asia', 'Canada')
    and stores not in (1,2,3,4);
```

Is missing or Is null;

```
proc sql;
select * from sashelp.column
where format is missing;     (alternatively we can use 'is null')
```

```
proc sql;
select * from sashelp.column
where informat is not missing;     (alternatively we can use is not null)
```

Like Operator;

```
proc sql;
select name, team, league, division, position
from sashelp.baseball
where name like '% B_l%';     ( % and _ are called wildcard characters)
```

Calculated keyword;

```
proc sql;
select make, model, type, origin,
      msrp-invoice as Discount,
      calculated Discount/msrp*100 as PercentDisc
from sashelp.cars
where calculated discount < 2000;
```

Enhancing Query Output;

```
title 'Discount below $2000';
proc sql;
footnote 'Worldwide data';
select make, model, type, origin label='Country',
    msrp-invoice as Discount format=dollar8.2,
    calculated Discount/msrp*100 as PercentDisc format=4.2
from sashelp.cars
where calculated discount < 2000;
```

Adding a new column;

```
title 'Discount below $2000';
proc sql outobs=50;
footnote 'Worldwide data';
select make, model, type, origin label='Country', msrp, invoice,
    'Difference in:', 2016,     (new columns added)
    msrp-invoice format=dollar6.
from sashelp.cars;
```

Summazrizing and grouping data;

```
proc sql;
select make, type, msrp-invoice as discount, avg(horsepower) as avghp,
    avg(calculated discount) as avgdisc
from sashelp.cars
group by make, type;
```

```
proc sql;
select avg(inventory) as stocks
from sashelp.shoes;
```
(produces avg of all rows in inventory column)

```
proc sql;
select product, avg(inventory) as stocks
from sashelp.shoes
group by product;
```
(produces group in this case product wise avg of inventory)

```
proc sql;
select product, avg(inventory) as stocks
from sashelp.shoes;
```
(in this case total inventory avg is shown for each row which is the same)

```
proc sql;
select species, sum(length1, length2, length3) as total
from sashelp.fish
group by species;
```
(same as length1+length2+length3)

Count function;

```
proc sql;
select species, count(*) as count
from sashelp.fish
group by species;      (shows each group's count)

proc sql;
select count(*) as count
from sashelp.fish;      (shows total count)

proc sql;
select count(weight) as count
from sashelp.fish;    (counts non-missing values in the specified column)

proc sql;
select count(distinct species) as types
from sashelp.fish;      (counts only distinct values of species)
[ proc sql;
select distinct species
from sashelp.fish;  ]    {this step lists all distinct column values}
```

Non-correlated Subquery;

```
proc sql;
select cause, day, substr(process,9,1) as process, avg(count) as avgcount
from sashelp.failure
group by day, cause
having avg(count) >
(select avg(count)
 from sashelp.failure);
```

```
proc sql;
select * from sashelp.mon111
where t in
(select year
 from sashelp.macrs10
 where year in (1, 2, 3, 4, 5));
```

```
proc sql;
select t, s391, s409
from sashelp.mon111
where t < any
(select year
 from sashelp.macrs10
 where year = 5);
```

```
proc sql;
select t, s391, s409
from sashelp.mon111
where t > all
(select year
 from sashelp.macrs10
 where year = 5);
```

Correlated subquery;

```
proc sql;
select name, address
from  newlib.testsql1
where 'Mumbai' =
(select location
 from newlib.testsql2
 where testsql1.id=testsql2.id);
```

Noexec option;

```
proc sql noexec;
select species, weight
from sashelp.fish
where weight ge 200
order by species, weight;
```

(useful for syntax checking, to know whether any errors are present in syntax)

Validate keyword;

```
proc sql;
validate    (similar to noexec option but only applicable to select statement following it)
select species, weight
from sashelp.fish
where weight ge 200
order by species, weight;
```

Exists and not exists;

```
proc sql;
select *
from  newlib.numpad4
where exists
(select * from newlib.numpad3
 where numpad4.x = numpad3.x);

proc sql;
select *
from  newlib.numpad4
where not exists
(select * from newlib.numpad3
 where numpad4.x = numpad3.x);
```

Combining tables horizontally;

```
proc sql;
select * from newlib.testsql1, newlib.testsql2;
```
(Cartesian product – each row in table one is combined with every row in table 2) {doesn't include a where statement hence Cartesian product}

Inner Join;

```
proc sql;
select * from newlib.testsql1, newlib.testsql3
where testsql1.id = testsql3.idnum;
```
(all columns showed in output including duplicates)

(valid syntax – same data type)

```
proc sql;
select testsql3.idnum, name, age, mother, father
from newlib.testsql1, newlib.testsql3
where testsql1.id = testsql3.idnum;
```
(duplicate columns removed)

Or

```
proc sql;
select testsql1.id, name, age, mother, father
from newlib.testsql1, newlib.testsql3
where testsql1.id = testsql3.idnum;
```
(duplicate columns removed)

```
proc sql;
select testsql1.*, mother, father     (all columns selected from testsql1)
from newlib.testsql1, newlib.testsql3
where testsql1.id = testsql3.idnum;
```

similarly,

```
proc sql;
select testsql3.*, name, age     (all columns selected from testsql3)
from newlib.testsql1, newlib.testsql3
where testsql1.id = testsql3.idnum;
```

```
proc sql;
select * from
newlib.testsql4, newlib.testsql5     (testsql5 contains two different values for same id)
where testsql4.id = testsql5.id;
```

Table alias;

```
proc sql;
select t1.id, name, surname
from newlib.testsql1 as t1,      (or we can use just t1 and t2 without preceding as)
    newlib.testsql2 as t2
where t1.id=t2.id;
```

```
proc sql;
select t1.id,
    substr(name,1,1) || '. '|| surname as fullname,
    age,
    location
from newlib.testsql1 as t1, newlib.testsql2 as t2
where t1.id=t2.id
and location='Mumbai'
order by age;
```

```
proc sql;
select address,
    count(surname) as actnum,
    avg(age) as avgage
from newlib.testsql1 as t1, newlib.testsql2 as t2
where t1.id=t2.id
group by address
order by address;
```

Left join;

```
proc sql;
select t1.id, name, mother, father
from newlib.testsql1 as t1
left join newlib.testsql3 as t3
on t1.id = t3.idnum;
```

Right join;

```
proc sql;
select t3.idnum, mother, address
from newlib.testsql3 as t3
right join newlib.testsql1 as t1
on t1.id = t3.idnum;
```

Full Join;

```
proc sql;
select *
from newlib.testsql3 as t3
full join newlib.testsql4 as t4
on t4.id = t3.idnum;
```

Inner join with outer join style syntax;

```
proc sql;
select * from
newlib.testsql1
inner join newlib.testsql3
on testsql1.id = testsql3.idnum;
```

Coalesce function; (same as match merge)

```
proc sql;

select coalesce (testsql2.id, testsql4.id) as Id, surname, father

from newlib.testsql2

full join newlib.testsql4

on testsql2.id = testsql4.id;
```

Except set operator;

```
proc sql;

select * from newlib.numpad1

except    (displays only the unique rows from the first table that are not found in the second table, duplicate rows are eliminated, matching rows from both tables are eliminated)

select * from newlib.numpad2;
```

```
proc sql;

select * from newlib.numpad1

except all    (all values from table one displayed except the matching rows in two tables)

select * from newlib.numpad2;
```

```
proc sql;

select * from newlib.numpad1

except corr    (displays only the same named columns, with unique values only from first table that do not have a matching value in the common column)

select * from newlib.numpad2;
```

proc sql;

select * from newlib.numpad1

except all corr *(displays only the common column, with unique values from table one including duplicate values from common column, matching values are eliminated)*

select * from newlib.numpad2;

Interesect set operator;

proc sql;

select * from newlib.numpad1

intersect *(displays only the matching rows in two tables, no duplicate rows shown)*

select * from newlib.numpad2;

proc sql;

select * from newlib.numpad1

intersect all *(duplicates of matching rows also included)*

select * from newlib.numpad2;

proc sql;

select * from newlib.numpad1

intersect corr *(only common column and matching rows across the tables displayed with no duplicates)*

select * from newlib.numpad2;

proc sql;

select * from newlib.numpad3

intersect all corr *(only common column, matching rows and also matching duplicate rows displayed)*

select * from newlib.numpad4;

Union set operator;

```
proc sql;
select * from newlib.numpad1
union     (duplicate rows and matching rows from second table eliminated, rest all rows displayed)
select * from newlib.numpad2;
```

```
proc sql;
select * from newlib.numpad1
union all    (displays all rows, rows are not sorted)
select * from newlib.numpad2;
```

```
proc sql;
select * from newlib.numpad1
union corr    (unique values only from both tables for only the common column are displayed)
select * from newlib.numpad2;
```

```
proc sql;
select * from newlib.numpad1
union corr all     (all values for only the common column displayed, rows are not sorted)
select * from newlib.numpad2;
```

Outer Union;

```
proc sql;
select * from newlib.numpad1
outer union     (doesn't overlay columns, displays all rows but are not sorted)
select * from newlib.numpad2;

proc sql;
select * from newlib.numpad1
outer union corr     (overlays common column, rows are not sorted)
select * from newlib.numpad2;
```

Create table;

```
proc sql;
create table work.discount
(Destination char (3),
 BeginDate num format = date9.,
 EndDate num format = date9.,
 FlyHours num);
proc sql;
describe table work.discount;     (can also be written without the preceding proc sql; statement)

proc sql;
create table work.cars
like sashelp.cars;     (crate an empty table with the same columns as another table)
```

```
proc sql;
create table work.cars (drop = drivetrain enginesize)
like sashelp.cars;
```

Or

```
proc sql;
create table work.cars
like sashelp.cars (drop = drivetrain enginesize);     (keep= option can also be used)
```

```
proc sql;
create table fish as     (used for creating a table from a query result)
select species, weight, height, width
from sashelp.fish
where weight > 200
order by species, weight;
```

Insert into;

```
proc sql;
insert into newlib.nums
set x=7, a='d'
set x=8, a='h';      (using set clause)
```

```
proc sql;
insert into newlib.nums (x,a)     (also valid without column names)
values (9,'n')
values (12,'l');
```

Create Index;

```
proc sql;
create unique index month     (simple index)
on work.enso1 (month);
```

Or

```
proc sql;
create unique index month
on work.enso2;
```

Create an index during data step;

```
options msglevel = i;
data enso1 (index = (month /unique));     (simple index)
set sashelp.enso;
run;
```

```
data enso3 (index = (month year /unique));    (simple index)
set sashelp.enso;
run;

proc sql;
create unique index annum    (composite index)
on work.enso3 (month, year);

data enso2 (index =(timeline =(month year)));    (composite index)
set sashelp.enso;
run;

data enso4 (index = (month readings = (year pressure)));    (month is simple and readings is composite)
set sashelp.enso;
run;

proc sql;
create index species
on work.fish;

proc sql;
create index company
on work.cars (make, origin);
```

```
proc datasets library = work nolist;
modify enso1;
index create month;
quit;
```

Delete or create an index on existing dataset;

```
proc datasets library = work nolist;
modify enso1;
index delete month;
quit;
```

```
proc datasets library = work nolist;
modify enso1;
index create month;
quit;
```

```
proc datasets library = work nolist;
modify enso4;
index delete month;
index delete notations;
index create month;     (simple index)
index create readings= (year pressure);     (composite index)
quit;
```

```
options msglevel=i;
```
(used to determine whether SAS uses an index or not)

```
proc print data = enso4;
where month = 25;
run;
```

Determining index usage;

```
options msglevel=i;
proc sql;
select * from enso1
```
(or enso2)

```
where month = 10;
```
(where clause remains the same for both simple and composite indexes)

IDXWHERE=;

```
proc sql;
select * from enso1 (idxwhere=no)
where month = 150;
```
(forces SAS to not use an index)

```
proc sql;
select * from enso1 (idxwhere=yes)
where month = 150;
```
(forces SAS to use an index)

IDXNAME=;

```
proc sql;
create index year      (second index named year created on column named year after creating composite index on columns month and year)
on enso2 (year);

proc sql;
select * from enso2     (selects index year )
where year = 2.5;

proc sql;
select * from enso2 (idxname=annum)     (forces SAS to use the index annum)
where month = 150;
```

Dropping an index;

```
proc sql;
drop index annum
from enso2;
```

Copying an indexed data set to a new library;

```
proc datasets library = work;
copy out = newlib;
select fish;
quit;

proc copy out = newlib in = work;
select vehicles;
run;
quit;

proc copy out = newlib in = work
move;
select vehicles;
run;
quit;
```

Change a datasets name;

```
proc datasets library = newlib nolist;
change fish = seafish;
quit;
```

Change a variables name;

```
proc datasets library = work;
modify fish;
rename Species = FishType;
quit;
```

Proc sql views;

```
proc sql;
create view work.vehicles as
select make, type, origin, cylinders
from sashelp.cars;

proc sql;
select * from work.vehicles
where cylinders>=4;

proc sql;
describe view work.vehicles;

proc sql;
create view newlib.testdatax as
select * from testdata;    (no libref required for views in the same library as the dataset)
```

Using libname statement;

```
proc sql;
create view testdatay as
select * from testdata
using libname newlib '/folder/myfolders/newfolder';
```

Drop view;

```
proc sql;
drop view newlib.testdatax;
```

Inobs & outobs;

```
proc sql inobs=20 outobs=10;     (outobs limits the no. of rows in the output)
select make, origin              (inobs limits the no. of rows to be read from each table)
from sashelp.cars;
```

Using the number option;

```
proc sql outobs=25 number;     (adds column named row to the output while numbering the rows)
select make,origin
from sashelp.cars;
```

Double option;

```
proc sql outobs=25 double;     (double spaces the output, affects only listing output)
select make,origin
from sashelp.cars;
```

Stimer option;

```
proc sql outobs=10 stimer;     (timing information for each statement is written to the SAS log)
select make,origin
from sashelp.cars;
select cylinders, horsepower
from sashelp.cars;
quit;
```

Using the Reset option;

```
proc sql outobs=5;
select make, origin
from sashelp.cars;
reset number;
select msrp, invoice
from sashelp.cars;
```

Dictionary option;

```
proc sql;

describe table dictionary.tables;
```

```
proc sql;

select libname, memname, memtype, nobs

from dictionary.tables

where libname = 'NEWLIB';
```

Flow option;

```
proc sql flow=10 15 outobs=25;     (specifies the width of character columns to float between 10 and 15 spaces)

select make, model

from sashelp.cars;
```

```
proc sql;

create table cars4 as

select make, origin, msrp

from sashelp.cars

where make = 'Porsche';
```

Combining data sets vertically;

Using filename= statement;

```
filename fishes ('/folders/myfolders/rawdata/fish.dat'
                                    '/folders/myfolders/rawdata/fish1.dat'
                                    '/folders/myfolders/rawdata/fish2.dat');
data work.fishing;
infile fishes;
input species $ weight height;
run;
```

(above statement combines raw data sets vertically or concatenates the raw data sets)

Using infile statement with filevar= option;

```
data saltfish;
do Fish = 1, 2, 3;
nextfile = "/folders/myfolders/rawdata/fish" || compress(put(Fish,2.)) || ".dat";
do until (lastobs);
infile temp filevar = nextfile end = lastobs;
input species $ weight height;
output;
end;
end;
stop;
run;
```

```
proc means data = sashelp.cars;
class make type drivetrain;
var msrp;
output out = cars mean = invoice;
run;

proc means data = sashelp.cars;
class make type drivetrain;
var msrp;
types () make*type type*drivetrain;
output out = cars2 mean = msrp;
run;

proc means data = sashelp.cars mean;
class make type drivetrain;
var msrp;
ways 1 2;
output out = cars3;
run;
```

Creating a table with integrity constraints;

```
proc sql;
create table employees
        ( id char(4) primary key,
          name char(10),
          contact num,
          age num,
          gender char(1) not null check (gender in ('M','F')));
quit;
```

```
proc sql;
describe table constraints employees;
```

Specifying a constraint;

```
proc sql;
create table discount
        ( destination char(3),
          date num format = date9.,
          discount num,
          constraint disc check (discount le .5),
          constraint dest not null (destination));    (constraints have been given names as disc & dest)
quit;
```

```
proc sql undo_policy = none;

insert into discount

values ('ewr', '03mar2016'd, .15)

values ('jfk', '02mar2016'd, .55);


proc sql;

describe table constraints discount;
```

Updating rows in a table;

```
data cars;

set sashelp.cars;

run;


proc sql;

update cars

set invoice = invoice * 1.05

where make = 'Audi';
```
(without the where statement all rows in the table will be updated, where statement updates only a subset of rows)

Updating rows using a case expression;

```
proc sql;
update cars1
set invoice = invoice *
        case
                when make = 'Acura'
                        then 1.10
                when make = 'BMW'
                        then 1.25
                when make = 'Jaguar'
                        then 1.50
        else 1      (without else clause all cases not in when clause will receive a missing value)
end;
proc sql;
update cars1
        set invoice = invoice *
                case make    (used in update only when the set clause contains equals {=} operator)
                        when 'Dodge'
                                then 1.15
                        when 'Ford'
                                then 1.35
                        when 'GMC'
                                then 1.45
                else 1
        end;
```

Using case expression in a select statement;

```
proc sql;
select make, type, origin,
        case
                when make = 'Audi'
                        then '5-Star'
                when make = 'BMW'
                        then '4-Star'
                else '3-Star'
        end as Safety_Rating
from cars1;
```

OR;

```
proc sql;
select make, type, origin,
        case make
                when 'Audi'
                        then '5-Star'
                when 'BMW'
                        then '4-Star'
                else '3-Star'
        end as Safety_Rating
from cars1;
```

Delete rows from tables;

```
proc sql;
delete from cars1
where cylinders < 4;     (to delete all rows remove the where clause)
```

Alter table statement;

```
proc sql;
        alter table cars1
                add Safety_Rating char (6)     (to populate new columns use the update statement)
                drop length, origin
                modify msrp format = dollar10.2,
              invoice format = dollar10.2;
```

Drop table;

```
proc sql;
drop table work.cars;
```

OR;

```
proc sql;
drop table cars1;
```

Combining data horizontally;

Using if/then-else statement;

```
data test;
set newlib.testdata;
if sex = 'm' then Group = 'A';
else if sex = 'f' then group = 'B';
else Group = ' ';
run;
```

(combines lookup values that are not stored in SAS data sets, same as creating a new variable)

Making use of index while combining data;

```
libname newlib '/folders/myfolders/newfolder';

options msglevel = i;
data combtest (index = (id/unique));
set newlib.testsql2;
run;

proc sql;
create unique index id
on newlib.testsql2 (id);
```

```
data combi;
set newlib.testsql4;
set newlib.testsql2 key = id;     (key= uses the index defined on lookup data set)
run;
```

Updating the master data set;

```
proc sort data = newlib.testsql2 out = testsql2;
by id;
run;

proc sort data = newlib.testsql4 out = testsql4;
by id;
run;

data testsql4;
update testsql4 testsql2;     (in this case testsql4 is master ds and testsql2 is transaction ds)
by id;     (by variable must contain all unique values in master; multiple values allowed in transaction)
run;
```

Audit trail;

```
proc datasets nolist;
audit capinfo;
initiate;
quit;
```

```
proc datasets nolist;

audit capinfo;

terminate;

quit;
```

Generation data sets;

```
proc datasets nolist;

modify cargorev (genmax=4);     { No message is written to the log when you specify the GENMAX= option.} [n=0 is the default]

quit;


proc print data=year(gennum=4);     /*absolute reference*/

run;


proc print data=year(gennum=-1);     /*relative reference*/

run;


proc contents data=year(gennum=-1);     /*relative reference*/     (information about a specific generation using the GENNUM= option with PROC CONTENTS)

run;
```

Modify datasets using modify statement;

```
data capacity;

modify capacity sasuser.newrtnum;     (capacity is master and newrtnum is transaction)

by flightid;

run;
```

Using key=; option (index) to modify a dataset;

```
data cargo99;
set sasuser.newcgnum
(rename = (capcargo = newCapCargo  cargowgt = newCargoWgt  cargorev = newCargoRev));
modify cargo99 key=flghtdte;
capcargo = newcapcargo;
cargowgt = newcargowgt;
cargorev = newcargorev;
run;
```

Integrity constraints using data step;

```
proc datasets nolist;
modify capinfo;
ic create PKIDInfo=primary key(routeid)  message='You must supply a Route ID Number';
ic create Class1=check(where=(cap1st<capbusiness  or capbusiness=.))
message='Cap1st must be less than CapBusiness';
quit;

proc datasets;
modify capinfo;
ic delete pkidinfo;
ic delete class1;
quit;
```

Proc Transpose;

```
proc transpose data=sasuser.ctargets
out=work.ctarget2
name=Month      (to rename _name_)
prefix=Ctarget;     (to rename col1, col2, ….. , coln)
[by year;]     we can use by statement if required
run;    (does not produce the output, print statement is required)
proc print data=work.ctarget2 label;
label Month=MONTH;
run;
```

OR;

```
proc transpose data=sasuser.ctargets
out=work.ctarget2 (rename=(col1=Ctarget1 col2=Ctarget2 col3=Ctarget3))
name=Month;    (the first transposed variable is _name_, rest all are col1, col2,…….,coln)
run;
```