Overview of Data Management Tasks
(command file=datamgt.sas)

Create the March data set:

To create the March data set, you can read it from the MARCH.DAT raw data file, using a data step, as shown below.

```
data march;
  infile "marflt.dat";
  input flight 1-3
           @4 date mmddyy6.
           @10 time time5.
           orig $ 15-17
           dest $ 18-20
           @21 miles comma5.
           mail 26-29
           freight 30-33
           boarded 34-36
           transfer 37-39
           nonrev 40-42
           deplane 43-45
           capacity 46-48;
format date mmddyy10. time time5. miles comma5.;
run;
```

Or, you can import the March.xls file from Excel, using the Import Wizard (commands not shown).

Make a copy of a SAS data set:

You can use a **set statement** to make a copy of a data set. In the commands below, NEWMARCH is created by making an **exact copy of MARCH** (which we assume was created as a temporary data set in the current SAS session).

```
data newmarch;
   set march;
run;
```

A **set statement** can also be used to make a copy of a **permanent** SAS data set:

```
libname sasdata2 "c:\users\kwelch\desktop\sasdata2";
data sasdata2.bank2;
     set sasdata2.bank;
run;
```

Additional commands can be added to the data step to create new variables, or to modify the data set in other ways.

```
data newmarch;
   set march;
   /*additional SAS statements*/
run;
```

NB: be sure all changes that you wish to make to your new data set are included before the run statement! After the run statement, the data set will be closed, and no additional variables can be added, or changes made to the data set.

Create a subset of data:

You can easily create a subset of your data by using the **set** statement along with a **subsetting if** statement. The subsetting if statement acts as a gateway for allowing observations to be written to a data set. In the examples below, the data set named MARCH15 will contain information on flights only on March 15th, 1990, while the data set named LONDON will contain information on all flights to London, and the data set named LONGFLT will contain information on all flights of 1000 miles or more..

```
data march15;
   set march;
   if date = "15MAR1990"D;
run;

data london;
   set march;
   if dest="LON";
run;

data longflt;
   set march;
   if miles >=1000;
run;
```

NB: The "subsetting if" statement can be used at any place in your data step code. It will only take effect when the data set is written out.

Another way to select cases to be included in a data set is to use an **output** statement. It is important to note that the output statement takes effect immediately (at the point in your code where it is included). Any commands that are added after the output statement will not affect the cases that were output earlier.

WRONG:

```
data london_latemarch;
   set march;
   if dest="LON" and date >="15MAR1990"D then output;
   totpassngrs = boarded + transfer + nonrev;
   pctfull = (totpassngrs/capacity)*100;
run;
```

The data set LONDON_LATEMARCH will contain flights to London on or after March 15th, the new variables TOTPASSNGRS and PCTFULL will be included in the new data set, but they will not have any values in them, because they were defined after the **output** statement. To correct this problem, make the **output** statement the last statement in the data step.

RIGHT:

```
data london_latemarch2;
   set march;
   totpassngrs = boarded + transfer + nonrev;
   pctfull = (totpassngrs/capacity)*100;
   if dest="LON" and date >="15MAR1990"D then output;
run;
```

Delete CASES from a data set:

A **delete** statement can be used to **remove a case or cases** from a data set. When the case is deleted, it is permanently removed from the data set. The delete statement takes effect immediately when it is specified, so deleted cases will not be available for any later programming statements.

```
data shortflt;
   set march;
   if miles >=1000 then delete;
   if date=. then delete;
run;
```

Similar to the output statement, the **delete** statement takes effect at the point in the data step where it is placed.

Keep or Drop VARIABLES:

You can control the variables that are included in a SAS data set by using **keep** and **drop statements** as part of the data step. The keep and drop statements may be given at any point in the data step, and only take effect at the time the data set is written.

```
data march_passngrs;
```

```
   set march;
   keep date time orig dest miles boarded transfer nonrev deplane
capacity;
run;
data march_passngrs2;
   set march;
   drop mail freight;
run;
```

Create new variables using transformations and recodes:

New variables can be created using **transformations** and **recodes** of variables in the data step using assignment statements with SAS **functions**, or **if…then** statements. **Assignment** statements (shown below) are used to create new variables based on the value of previously defined variables, expressions, or constants, and are of the form:

```
newvar = expression;
```

Example:

```
data march_recode;
   set march;

   totpassngrs = boarded + transfer + nonrev;
   totpassngrs2 = sum(boarded,transfer,nonrev);

   empty_seats = capacity - totpassngrs;
   totnonpass  = mail + freight;
   pctfull = (totpassngrs/capacity)*100;

   logpassngrs = log(totpassngrs);
   int_pctfull = int(totpassngrs/capacity)*100;
   rnd_pctfull = round(pctfull,.1);

   if pctfull = 100 then full_flight = 1;
   else full_flight = 0;
   if pctfull = . then full_flight = .;

   if pctfull not=. then do;
      if pctfull < 25 then full_cat = 1;
      if pctfull >=25 and pctfull <50 then full_cat=2;
      if pctfull >=50 and pctfull <75 then full_cat=3;
      if pctfull >=75 then full_cat=4;
   end;

   if dest = "CPH" or dest="FRA" or dest = "LON"
     or dest = "PAR" or dest = "YYZ" then USA = 0;

   if dest in("DFW", "LAX", "ORD", "WAS") then USA = 1;
run;
```

Check the new variables using Proc Means and Proc Freq:

```
title "Check New Variables";
proc means data=march_recode;
run;
proc freq data=march_recode;
  tables full_flight full_cat dest USA;
run;
```

Check New Variables
The MEANS Procedure

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
flight	Flight number	635	447.9086614	275.3102085	114.0000000	982.0000000
date		634	11031.98	8.9801263	11017.00	11047.00
time		635	47952.47	14707.08	24960.00	75960.00
miles		635	1615.25	1338.47	229.0000000	3857.00
mail		634	381.0031546	74.6288128	195.0000000	622.0000000
freight		634	333.9511041	98.1122248	21.0000000	631.0000000
boarded		633	132.3570300	43.4883098	13.0000000	241.0000000
transfer		635	14.4062992	5.3362008	0	29.0000000
nonrev		635	4.1133858	1.9243731	0	9.0000000
deplane		635	146.7842520	45.4289656	18.0000000	250.0000000
capacity		635	205.3795276	27.1585929	178.0000000	250.0000000
totpassngrs		**633**	150.8878357	43.0930520	31.0000000	250.0000000
totpassngrs2		**635**	150.4598425	43.6959260	9.0000000	250.0000000
empty_seats		**633**	54.5244866	34.9192529	0	151.0000000
totnonpass		**633**	715.1927330	124.8981261	341.0000000	1085.00
pctfull		**633**	73.0774908	17.7696598	17.2222222	100.0000000
logpassngrs		**633**	4.9681880	0.3292127	3.4339872	5.5214609
int_pctfull		**633**	8.3728278	27.7198922	0	100.0000000
rnd_pctfull		**633**	73.0764613	17.7693610	17.2000000	100.0000000
full_flight		**633**	0.0837283	0.2771989	0	1.0000000
full_cat		**633**	3.3791469	0.6785651	1.0000000	4.0000000
USA		**632**	0.6819620	0.4660832	0	1.0000000

The two variables, TOTPASSNGRS (n=633) and TOTPASSNGRS2(n=635) have different numbers of cases, because they were created in different ways. TOTPASSNGRS is created using the mathematical operators (+), so the resulting variable is missing if the value of any of the variables in the expression is missing. TOTPASSNGRS2 returns the sum of the non-missing argument variables, so it produces the sum of any non-missing argument variables.

Also note the syntax used to create the new variables, FULL_FLIGHT and FULL_CAT. When using "**Else**" with SAS, all other values, including missing will be included in the Else category. We get around this by setting cases with a missing value for PCTFULL to missing in the resulting variable. We use an **If…then** statement when creating FULL_CAT to be sure this new variable is only created if PCTFULL is not missing (**`if pctfull not=. then do;`**). If you use an **if…**statement, it must be followed by an **end** statement.

The new variable USA is created from the character variable, DEST. The **in** operator is used to shorten the syntax for setting up the value of USA=1.

Sort Cases:

Use **Proc Sort** to sort a data set. Once sorted, a data set remains sorted, and any later analyses can be done either for the entire data set, or for subgroups by including a **by** statement in a procedure. A separate analysis will be done for each of the "by" groups.

```
proc sort data=march_recode;
  by USA;
run;


title "Descriptive Statistics by US vs Non-US Destinations";
proc means data=march_recode;
  by USA;
run;
```

Sorting by more than one variable:

You can sort by several variables, as shown in the example below. Proc sort organizes the data so that the first variable represents the slowest changing index (i.e., cases will be sorted first by DATE, and then by levels of DEST within DATE).

```
proc sort data=march_recode;
     by date USA;
run;
title "Descriptive Statistics by Date and Destination";
proc means data=march_recode;
     by date USA;
run;
```

Using the Tagsort Option:

Sorting is one of the more computationally intensive operations. It requires a lot of hard drive space, which can be a problem, especially for data sets with many observations and a large number of variables. You can be more efficient in sorting if you use the **tagsort** option. This method sorts only the key variables and then rebuilds the dataset by pulling up the appropriate observation and attaching all the rest of the variables. The tagsort method often takes longer to sort a data set, but uses less hard drive space.

```
proc sort data=march_recode tagsort;
     by date dest;
run;
```

Creating a New Sorted Data Set:

If you wish to create a new data set, and maintain the input data set in its original order, you can use the **out=** option on the Proc Sort statement, as shown below:

```
proc sort data=march_recode out=sortdat;
     by totpassngrs;
run;
```

Getting Rid of Duplicate Cases for the Same ID – Using the Nodupkey Option:

Proc Sort provides an easy way to get rid of duplicate cases having the same values of the key variables. Use the **nodupkey** option on the Proc Sort statement, as shown below. Check the log to see how many duplicates were deleted. The original data set will not be affected.

```
proc sort data=march_recode out=sortdat2 nodupkey;
   by date dest;
run;
```

Getting Rid of Duplicate Records for the Same ID – Using the Noduprec Option:

You can also ask SAS to eliminate any cases that are duplicates for all variables using the **noduprec** option, as shown in the code below.

```
proc sort data=march_recode out=sortdat3 noduprec;
   by date dest;
run;
```

Selecting cases for analysis:

Cases can be selected for a given analysis by using a **where** statement.

Selecting cases based on values of a character variable:

When selecting cases based on the value of a character variable, **be sure to enclose the value or values in quotes**, as shown below:

```
title "Flights to Los Angeles";
proc print data=march_recode;
    where dest = "LAX";
    var flight dest totpassngrs;
run;
```

If you wish to select the observations based on a missing value for a character variable, use quotes around a blank " " (the missing value for character variables).

```
title "Missing Destination";
proc print data=march_recode;
    where dest = " ";
    var flight dest totpassngrs; run;
```

Selecting cases based on values of a numeric variable:

Cases used in an analysis may be selected based on the values of a numeric variable. The Boolean operators (<, >, <=, >=, =, ~=) may be used to get the desired case selection, as shown below. **Do not use quotes when specifying the value of a numeric variable.**

```
title "Flights Less than 30 Percent Full";
proc print data=march_recode;
    where pctfull < 30;
    var dest date pctfull;run;
```

Those with PCTFULL missing are also included in this case selection, because missing is evaluated as less than any numeric value.

```
Flights Less than 30 Percent Full
Obs    dest          date    pctfull
 99    ORD     03/05/1990    28.0952
102    WAS     03/05/1990    17.2222
235    DFW     03/12/1990    27.2222
390    WAS     03/19/1990    29.4444
421    LAX     03/21/1990       .
451    WAS     03/22/1990    18.3333
512    WAS     03/25/1990    23.8889
```

The syntax below can be used to exclude the missing values from those cases printed:

```
title "Flights Less than 30 Percent Full";
proc print data=march_recode;
    where pctfull not=. and pctfull < 30;
    var dest date pctfull;
run;
```

The where statement can also be used with "**between**" to restrict cases used in an analysis. The example below will print those cases with percent full from 25 to 35:

```
title "Flights Between 25 and 35 Percent Full";
proc print data=march_recode;
    where pctfull between 25 and 35;
run;
```

If you wish to select observations based on a missing value for a numeric variable, use a period to indicate missing, as shown in the example below.

```
title "Cases Where Number of Passengers is Missing";
proc print data=march_recode;
    where totpassngrs = .;
    var flight dest totpassngrs;
run;
```

You can also select cases using a combination of character and numeric variables in the **where** statement:

```
title "Flights less than 60 percent full to London";
proc print data=march_recode;
    where (pctfull < 60) and (dest="LON") ;
    var flight dest totpassngrs capacity pctfull;
run;
```

Selecting cases based on dates:

You can select cases for a procedure based on dates, by using a SAS **date constant**. Note that the date constant is specified in quotes with the day as a two-digit number, followed by a three-letter abbreviation for the month, followed by a 2 or 4-digit number for the year. A letter D (either upper or lower case) must appear after the quote to let SAS know that this is a date.

```
title "Flights on March 7th, 1990";
proc print data=march_recode;
    where date = "07MAR90"D;
run;
```

You can also use "**where … between**" with dates to specify a range of dates:

```
title "Flights March 7th to March 9th , 1990";
proc print data=march_recode;
    where date between "07MAR90"D and "14MAR90"D;
run;
```

You can use the same method for selecting observations based on missing values for a date variable as for a numeric variable, because dates are stored as numeric values in SAS.

```
title "Cases with Missing Date";
proc print data=march_recode;
    where date = .;
    var flight dest date;
run;
```

Combining SAS Data Sets

Stack Data Sets Vertically (adds new cases):

You can use the **set** statement to combine data sets vertically. It is not necessary for the data sets being combined to have their variables in the same order, or even for them to have the same variables. However, it is critical that if the same variable does appear in both data sets, it should be of the same **type** (either character or numeric) in both.

If a variable is present in one data set and not in the other, the values for that variable will be missing for all cases for the data set that did not have it. The order of variables in the resulting data set will reflect the order of the first data set listed.

In the example below, the data set BOYS has different variables, which are also in a different order, than the variables in the data set GIRLS. The length of the variable Name in the BOYS dataset will be 8 characters, which is the default, as shown in the output from Proc Contents.

```
data boys;
     input name  $ sex $ age height teacher $;
     cards;
Tom       M 12 62 Smith
Bob       M 13 57 Green
Joe       M 11 59 Green
Harry     M 12 53 Green
William   M 13 60 Smith
John      M 11 57 Smith
Richard   M 11 55 Green
     ;
title "Boys Dataset";
proc contents data=boys varnum;
run;
```

Boys Dataset

The CONTENTS Procedure

Data Set Name	WORK.BOYS	Observations	7
Member Type	DATA	Variables	5
Engine	V9	Indexes	0
Created	Thu, Sep 08, 2011 12:51:33 PM	Observation Length	40
Last Modified	Thu, Sep 08, 2011 12:51:33 PM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_32		
Encoding	wlatin1 Western (Windows)		

Variables in Creation Order

#	Variable	Type	Len
1	**name**	**Char**	**8**
2	sex	Char	8
3	age	Num	8
4	height	Num	8
5	teacher	Char	8

Using a Length Statement:

We use a Length statement for the variable Name in the GIRLS dataset, so that we can accommodate the longest name that occurs. Note that you can use any length (Up to 32767 characters) that you want to specify for a character variable; just be sure that it is long enough to accommodate the longest value that will occur within the variable. Note that the Length statement needs to follow immediately after the data statement.

```
data girls;
     length name $ 10;
     input name $ age sex $ teacher $;
     cards;
Sharice  13 F  Smith
Mary     12 F  Smith
Ellen    11 F  Green
Carol    11 F  Green
Chris    13 F  Smith
Claire   12 F  Green
Raye     13 F  Smith
Wilhelmina 12 F Green
     ;
title "Girls Data";
proc contents data=girls varnum;
run;
```

Girls Data

The CONTENTS Procedure

Data Set Name	WORK.GIRLS	Observations	8
Member Type	DATA	Variables	4
Engine	V9	Indexes	0
Created	Thu, Sep 08, 2011 12:53:38 PM	Observation Length	40
Last Modified	Thu, Sep 08, 2011 12:53:38 PM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_32		
Encoding	wlatin1 Western (Windows)		

Variables in Creation Order

#	Variable	Type	Len
1	name	Char	10
2	age	Num	8
3	sex	Char	8
4	teacher	Char	8

Stacking the data using SET (BOYS First):

We now use a Set statement to stack the two datasets. This will make the boys dataset be first in our output dataset, followed by the GIRLS dataset. The characteristics of the variables will be taken from the BOYS data. This means that the length of Name will be too short. To avoid this, we use a Length statement when stacking the two datasets. Again, be sure the Length statement follows the data statement, immediately.

```
data allkids;
     length $ name 10;
     set boys girls;
run;

title "printout of allkids dataset";
title2 "with boys first in the data set";
proc print data = allkids;
run;
```

printout of allkids dataset
with boys first in the data set

Obs	name	sex	age	height	teacher
1	Tom	M	12	62	Smith
2	Bob	M	13	57	Green
3	Joe	M	11	59	Green
4	Harry	M	12	53	Green
5	William	M	13	60	Smith
6	John	M	11	57	Smith
7	Richard	M	11	55	Green
8	Sharice	F	13	.	Smith
9	Mary	F	12	.	Smith
10	Ellen	F	11	.	Green
11	Carol	F	11	.	Green
12	Chris	F	13	.	Smith
13	Claire	F	12	.	Green
14	Raye	F	13	.	Smith
15	Wilhelmina	F	12	.	Green

Stacking the data using SET (GIRLS First):

We again use a Set statement to combine the two data sets, but with the GIRLS dataset being first. In this case, we don't need to use a length statement, because the GIRLS dataset has a length of 10 for the variable Name, so SAS will pick up the characteristics of the variable Name from the GIRLS dataset. However, it is probably safer to specify a length here, too, so that the lengths of the character variables will match when combining the data.

```
data allkids2;
     set girls boys;
run;

title "printout of allkids2 data set";
title2 "with girls first in the data set";
proc print data = allkids2;
run;
```

printout of allkids2 dataset
with girls first in the data set

Obs	name	age	sex	teacher	height
1	Sharice	13	F	Smith	.
2	Mary	12	F	Smith	.
3	Ellen	11	F	Green	.
4	Carol	11	F	Green	.
5	Chris	13	F	Smith	.
6	Claire	12	F	Green	.
7	Raye	13	F	Smith	.
8	Wilhelmina	12	F	Green	.
9	Tom	12	M	Smith	62
10	Bob	13	M	Green	57
11	Joe	11	M	Green	59
12	Harry	12	M	Green	53
13	William	13	M	Smith	60
14	John	11	M	Smith	57
15	Richard	11	M	Green	55

Notice that the order of the variables in the final data set is changed, depending on which data set was listed first in the set statement, but the values in both data sets are the same.

Merge Data Sets Horizontally (adds new variables):

SAS data sets can be merged horizontally in a number of ways. This method of combining data sets allows you to match based on some key variable(s) such as ID or household. You can merge based on the values of one or more variables. Note that if you merge, based on the value of a character variable, be sure that you specify a LENGTH statement as the first statement in your data step, so that the lengths of the variables in both datasets can be accommodated. **You must first sort the data sets that are being merged by the key variable(s), and then merge by the same key variable(s).**

The example below shows how to merge two data sets for the same people. The dataset, EXAM contains data for a hypothetical group of people on a physical exam. The data set LAB contains information for the *some of the same people* on their laboratory results.

```
data exam;
     input id examdate mmddyy10. sex age height weight sbp dbp;
     format examdate mmddyy10.;
     cards;
1 10/18/2000 1 25 72 156 128 89
2 05/29/2000 1 33 68 168 145 96
3 02/21/2000 1 47 65 182 152 98
4 06/17/2000 1 29 69 190 139 91
5 01/11/2000 2 37 62 129 145 93
6 08/15/2000 2 42 64 156 133 94
;
```

```
data lab;
    input id hgb;
    cards;
1  13.2
4  12.1
3  14.5
6  12.8
12 13.0
;
proc sort data=exam;
   by id;
run;
proc sort data=lab;
   by id;
run;

data exam_lab;
    merge exam lab;
    by id;
run;

title "Printout of Exam_lab Data Set";
proc print;
run;
```

Printout of Exam_lab Data Set

Obs	id	examdate	sex	age	height	weight	sbp	dbp	hgb
1	1	10/18/2000	1	25	72	156	128	89	13.2
2	2	05/29/2000	1	33	68	168	145	96	.
3	3	02/21/2000	1	47	65	182	152	98	14.5
4	4	06/17/2000	1	29	69	190	139	91	12.1
5	5	01/11/2000	2	37	62	129	145	93	.
6	6	08/15/2000	2	42	64	156	133	94	12.8
7	12	.	.	.	.	.	.	.	13.0

By default, SAS will include all observations from both data sets in the merged data. Notice in the above example, ID numbers 2 and 5 are in the EXAM data set, but not in the lab data set, while ID number 12 is in the LAB data set, but not in the EXAM data set. However all of these cases are in the merged EXAM_LAB data set.

You can control the observations that get written to the final data set, using the **in=** data set option. This creates a **temporary variable** that indicates whether a case is in a particular data set or not. Then you can control which observations get written out, using **subsetting if** statements. The examples below show three different ways this could be done.

```
/*How to include only cases that are in both data sets*/

data exam_lab2;
     merge exam(in=a) lab(in=b);
     by id;
     if a and b;
run;

title "Exam_lab2 Data Set Includes Only Those";
title2 "In Both Data Sets";
proc print data=exam_lab2;
run;
```

Exam_lab2 Data Set Includes Only Those
In Both Data Sets

Obs	id	examdate	sex	age	height	weight	sbp	dbp	hgb
1	1	10/18/2000	1	25	72	156	128	89	13.2
2	3	02/21/2000	1	47	65	182	152	98	14.5
3	4	06/17/2000	1	29	69	190	139	91	12.1
4	6	08/15/2000	2	42	64	156	133	94	12.8

```
/*How to include cases that are in EXAM, regardless of Lab
Data*/

data exam_lab3;
     merge exam(in=a) lab(in=b);
     by id;
     if a;
run;

title "Exam_lab3 Data Set Includes Those";
title2 "In Exam Data, Regardless of Lab Data";
proc print data=exam_lab3;
run;
```

Exam_lab3 Data Set Includes Those
In Exam Data, Regardless of Lab Data

Obs	id	examdate	sex	age	height	weight	sbp	dbp	hgb
1	1	10/18/2000	1	25	72	156	128	89	13.2
2	2	05/29/2000	1	33	68	168	145	96	.
3	3	02/21/2000	1	47	65	182	152	98	14.5
4	4	06/17/2000	1	29	69	190	139	91	12.1
5	5	01/11/2000	2	37	62	129	145	93	.
6	6	08/15/2000	2	42	64	156	133	94	12.8

```
/*How to include cases that are in LAB, regardless of Exam
Data*/
```

```
data exam_lab4;
     merge exam(in=a) lab(in=b);
     by id;
     if b;
run;

title "Exam_lab4 Data Set Includes Those";
title2 "In Lab Data, Regardless of Exam Data";
proc print data=exam_lab4;
run;
```

Exam_lab4 Data Set Includes Those
In Lab Data, Regardless of Exam Data

Obs	id	examdate	sex	age	height	weight	sbp	dbp	hgb
1	1	10/18/2000	1	25	72	156	128	89	13.2
2	3	02/21/2000	1	47	65	182	152	98	14.5
3	4	06/17/2000	1	29	69	190	139	91	12.1
4	6	08/15/2000	2	42	64	156	133	94	12.8
5	12	.	.	.	.	.	.	.	13.0

How to merge data sets when the variable names are the same:

When you merge two datasets, SAS requires that the names be different (except for the variables used as Key variables in the merge). If the names are not different in the two datasets, you will. overwrite the values of the variables in the first dataset with the values in the second dataset. If the two data sets that you wish to merge have the same variable names, this can be handled by using the rename dataset option for either one or both of the datasets.

```
data oldsal;
     input name $ idnum sex $ age salary jobcat year;
     cards;
 Roger  518 M 45 7677 2 1989
 Martha 321 F 28 5000 1 1989
 Zeke   444 M 33 6075 1 1989
 Barb  1728 F 40 9023 2 1989
 Bill   993 M 36 7739 3 1989
 Sandy 1002 F 29 6161 3 1989
     ;

data newsal;
     input name $ idnum salary jobcat year;
     cards;
 Hank    108 11138 1 1995
 Fred    519 10035 2 1995
 Zeke    444  9697 1 1995
 Martha  321  7987 2 1995
 Sandy  1002  6995 2 1995
```

```
 Bill    993 12400 3 1995
 Roxy    773 10119 2 1995
     ;

/*merging by idnum*/
proc sort data=oldsal;
    by idnum;
run;

proc sort data=newsal;
    by idnum;
run;


data combine1;
    merge oldsal(rename=(salary=salary89 jobcat=jobcat89))
          newsal(rename=(salary=salary95 jobcat=jobcat95));
    by idnum;
    drop year;
run;

title "printout of combine1 data set";
title2 "matching by id number";
title3 "all cases that were in either data set are included";
proc print data=combine1;
run;
```

printout of combine1 data set
matching by id number
all cases that were in either data set are included

Obs	name	idnum	sex	age	salary89	jobcat89	salary95	jobcat95
1	Hank	108		.	.	.	11138	1
2	Martha	321	F	28	5000	1	7987	2
3	Zeke	444	M	33	6075	1	9697	1
4	Roger	518	M	45	7677	2	.	.
5	Fred	519		.	.	.	10035	2
6	Roxy	773		.	.	.	10119	2
7	Bill	993	M	36	7739	3	12400	3
8	Sandy	1002	F	29	6161	3	6995	2
9	Barb	1728	F	40	9023	2	.	.

You can control the observations that are written to the final data set, using **in=** data set options for this type of merge also.

```
/*merging by idnum, but keeping only cases that are in both
datasets*/

data combine2;
    merge oldsal(in=a rename=(salary=salary89 jobcat=jobcat89))
          newsal(in=b rename=(salary=salary95 jobcat=jobcat95));
    by idnum;
    if a and b;
    totsal = sum (salary89,salary95);
    format salary89 salary95 totsal dollar12.;
    drop year;
run;
proc print data=combine2;
    title "printout of combine2 data set";
    title2 "matching by id number";
    title3 "and only including cases that are in both data
sets";
run;
```

printout of combine2 data set
matching by id number
and only including cases that are in both data sets

Obs	name	idnum	sex	age	salary	jobcat	salary95	jobcat95	totsal
1	Martha	321	F	28	5000	1	7987	2	$12,987
2	Zeke	444	M	33	6075	1	9697	1	$15,772
3	Bill	993	M	36	7739	3	12400	3	$20,130
4	Sandy	1002	F	29	6161	3	6995	2	$13,156

Include Cases from EXAM regardless of whether they are in LAB or not:

If you want to include only cases from the EXAM dataset, regardless of whether they appear in the LAB dataset or not, you can use the following syntax.

```
/*INCLUDE CASES THAT ARE IN EXAM, WHETHER THEY
   ARE IN LAB OR NOT*/

data exam_lab3;
     merge exam(in=a) lab(in=b);
     by id;
     if a;
run;

title "Exam_lab3 Data Set Includes Those";
title2 "In Exam Data, Regardless of Lab Data";
proc print data=exam_lab3;
run;
```

Note that in the output below, two cases are included for people who were in the EXAM dataset, but had no lab values.

Exam_lab3 Data Set Includes Those In Exam Data, Regardless of Lab Data

Obs	id	examdate	sex	age	height	weight	sbp	dbp	hgb
1	1	10/18/2000	1	25	72	156	128	89	13.2
2	2	05/29/2000	1	33	68	168	145	96	.
3	3	02/21/2000	1	47	65	182	152	98	14.5
4	4	06/17/2000	1	29	69	190	139	91	12.1
5	5	01/11/2000	2	37	62	129	145	93	.
6	6	08/15/2000	2	42	64	156	133	94	12.8

Merging Data from a Table with a Dataset having multiple obs per group

Having the ability to specify which dataset(s) provide cases when merging allows a lot of freedom when combining SAS datasets. For example, combining information on patients who live in several counties with information from the county can be easily accomplished, as shown in the example below.

First, we import the AgeStudy data, and check it using Proc Print:

```
PROC IMPORT OUT= WORK.AgeStudy
            DATAFILE= "AgeStudy.xls"
            DBMS=EXCEL REPLACE;
     SHEET="Sheet1";
RUN;
title "AgeStudy Data";
proc contents data=agestudy;
run;
proc print data=agestudy;
run;
```

AgeStudy Data

Obs	County	Name	Gender	Age
1	Washtenaw	Jim	m	54
2	Washtenaw	Bob	m	49
3	Washtenaw	Susan	f	29
4	Wayne	Sally	f	35
5	Muskegon	Robert	m	53
6	Osceola	Jill	f	59
7	Newaygo	Phil	m	37
8	Wayne	Roger	m	56
9	Washtenaw	Gipper	m	74
10	Muskegon	Joey	m	58

Now, we import the Census data, and take a brief look at it.

```
PROC IMPORT OUT= WORK.Census
            DATAFILE= "MI_Census_2000.xls"
            DBMS=XLS REPLACE;
     RANGE="Sheet1";
RUN;

title "Census Data";
proc contents data=census;
run;
proc print data=census(obs=12);
run;
```

Census Data

Obs	County	Census_ 2000
1	Alcona	11719
2	Alger	9862
3	Allegan	105665
4	Alpena	31314
5	Antrim	23110
6	Arenac	17269
7	Baraga	8746
8	Barry	56755
9	Bay	110157
10	Benzie	15998
11	Berrien	162453
12	Branch	45787

Now, merge the two datasets, but keep only cases that occur in the Age Study Dataset. Note that each case in Washtenaw County will be matched with the data for Washtenaw County in the Census datasets. Also note that we used a Length statement to be sure the variable County would have sufficient length to accommodate all county names. The "IF" statement restricts the merged dataset, so that it contains only observations that were originally in the AGESTUDY dataset. Counties that occur in the Census data, but not in the Agestudy data will not be included.

```
proc sort data=census;
by county;
run;

proc sort data=agestudy;
by county;
run;

data study_census;
  length county $ 16;
  merge agestudy(in=instudy) census(in=incensus);
  by county;
  if instudy;
run;
```

```
title "Printout of Merged Data";
proc print data=study_census;
format Census_2000 comma12.;
run;
```

Note that every observation in Muskegon County gets the Census for Muskegon County. The same is true for the cases in Washtenaw County, etc. In general, SAS will always fill in the values for matching Key variables, regardless of the dataset in which the duplicates occur. However, SAS does not allow duplicates of the same values of key variables to occur in both datasets.

Printout of Merged Data

Obs	county	Name	Gender	Age	Census_2000
1	Muskegon	Robert	m	53	170,200
2	Muskegon	Joey	m	58	170,200
3	Newaygo	Phil	m	37	47,874
4	Osceola	Jill	f	59	23,197
5	Washtenaw	Jim	m	54	322,895
6	Washtenaw	Bob	m	49	322,895
7	Washtenaw	Susan	f	29	322,895
8	Washtenaw	Gipper	m	74	322,895
9	Wayne	Sally	f	35	2,061,162
10	Wayne	Roger	m	56	2,061,162