

By Statements and Non-rectangular File Structures in SAS

1) Processing non-rectangular files in SAS

- recall the standard Execution Phase in a SAS data step; execution proceeds as an implicit loop over the observations being read, with each observation being processed as follows:
 - starts from DATA statement
 - a record is read
 - additional statements in the DATA step are executed on *that observation*
 - at end of DATA step, the observation is output, processing returns to the top of that step, and all information in the program data vector is reset
 - iteration counter (`_N_`) is updated
 - flow continues until the end of the input file is reached
- this execution sequence is especially well-suited for processing “rectangular” files—files with the same record layout for each observation and with no relationships among the observations
- data steps and other SAS procedures can be written to accommodate non-rectangular files
- a key tool in processing these alternative data structures is BY-group processing
- this class will cover
 - the general use of the BY statement in DATA steps and other procedures
 - alternative data structures
 - general processing techniques for these structures
 - methods for sorting and merging files
 - methods for traversing other file structures

2) BY-group processing¹

- refers to techniques in SAS for working with data that are ordered or grouped in terms of the values of one or more variables
- requires a BY statement

```
BY <DESCENDING> <ordering_var1>  
  <<DESCENDING> <ordering_var2> ...  
  <<DESCENDING> <ordering_var_n>>> <by_options>;
```

where *ordering_var1* is the first variable that is used to order or group the data, *ordering_var2* is the second variable that is used to order or group the data, etc.

- for multiple variables, the ordering or grouping is assumed to be nested
- for instance, in a data set with two ordering variables, SAS expects that observations are ordered or grouped by *ordering_var2* within groups of *ordering_var1*
- the first step in any BY-group processing is to order or group the data; this is generally done through the SORT procedure (though there are other ways that a data set may be ordered)
- BY variables, values, and groups

¹ See <http://support.sas.com/documentation/cdl/en/lrcon/59522/HTML/default/a002261727.htm>.

- BY variable(s): is(are) the ordering variable(s) listed in the BY statement
- BY value: is the value of an ordering variable within a BY group
- BY group: is the set of observations defined by a particular BY value or combination of BY values
- BY-group processing treats all of the observations within a BY-group as a group
- the actual processing varies, depending on the procedure or the context
- DATA step processing
 - a BY statement causes a DATA step to look for changes in the values of the BY variables
 - when SAS encounters the first instance of a BY value, it sets a temporary FIRST logical (binary 0/1) indicator to 1
 - when SAS encounters the last instance of a BY value, it sets a temporary LAST logical indicator to 1
 - because there can be multiple BY variables, there need to be multiple FIRST and LAST indicators; for a given BY *variable*, the corresponding indicators would be *FIRST.variable* and *LAST.variable*
 - consider the following sequence from a data set that is sorted according to BY variables *i* and *j* and that contains an additional variable *x*

<u>N</u>	initial variables			variables created by BY processing			
	i	j	x	FIRST.i	FIRST.j	LAST.i	LAST.j
1	100	2	25002	1	1	0	0
2	100	2	25009	0	0	0	1
3	100	5	25837	0	1	0	0
4	100	5	26387	0	0	0	0
5	100	5	26287	0	0	0	0
6	100	5	26053	0	0	1	1
7	200	2	26135	1	1	0	0
8	200	2	25915	0	0	0	1
9	200	3	26248	0	1	0	0
10	200	3	26730	0	0	0	1
11	200	4	25880	0	1	0	0
12	200	4	25911	0	0	0	1
13	200	5	26184	0	1	1	1
14	300	1	26282	1	1	1	1

- processing can then be done based on the FIRST and LAST variables
- match merging
 - when a BY statement and MERGE statement are used together in the same DATA step; the BY statement causes the MERGE to perform “match merging”
 - match merging combines observations from two data sets that share the same values of the BY variables

- SAS supports “many-to-one” and “one-to-many” match merging; in other words, one of the merged data sets must only contain one observation per BY-value combination
- when a SAS data set is created by match merging, the resulting data set is automatically sorted according to the BY statement
- statistical and reporting procedure processing
 - a BY statement in a statistical or reporting procedure causes the procedure to calculate separate statistics or list separate output for each BY group
 - if an OUTPUT statement is used, only one data set is created; however, this data set
 - ◊ will contain the BY variables and
 - ◊ will have separate observations for each BY group
 - the processing for several procedures for BY and CLASS groups is similar
- SORT procedure processing
 - a BY statement in a SORT procedure causes the data set to be arranged in order of the BY variables
 - the syntax for the SORT procedure is

```
PROC SORT <options>;
  BY <DESCENDING> <ordering_var1>
     <<DESCENDING> <ordering_var2> ...
     <<DESCENDING> <ordering_varn>>>;
```

- a BY statement must accompany the SORT procedure
- if the DESCENDING option is used in front of an ordering variable, the data set is sorted in DESCENDING (reverse) order of that variable
 - ◊ the default is to sort the data in ascending order
 - ◊ DESCENDING must be used in front of every BY variable that is to be sorted in DESCENDING order
- the SORT procedure can be very time- and resource-intensive
 - ◊ the procedure reorders the data set
 - ◊ it also completely recopies the data set
 - ◊ you want to program carefully to use as few SORTs as possible

3) Database structures

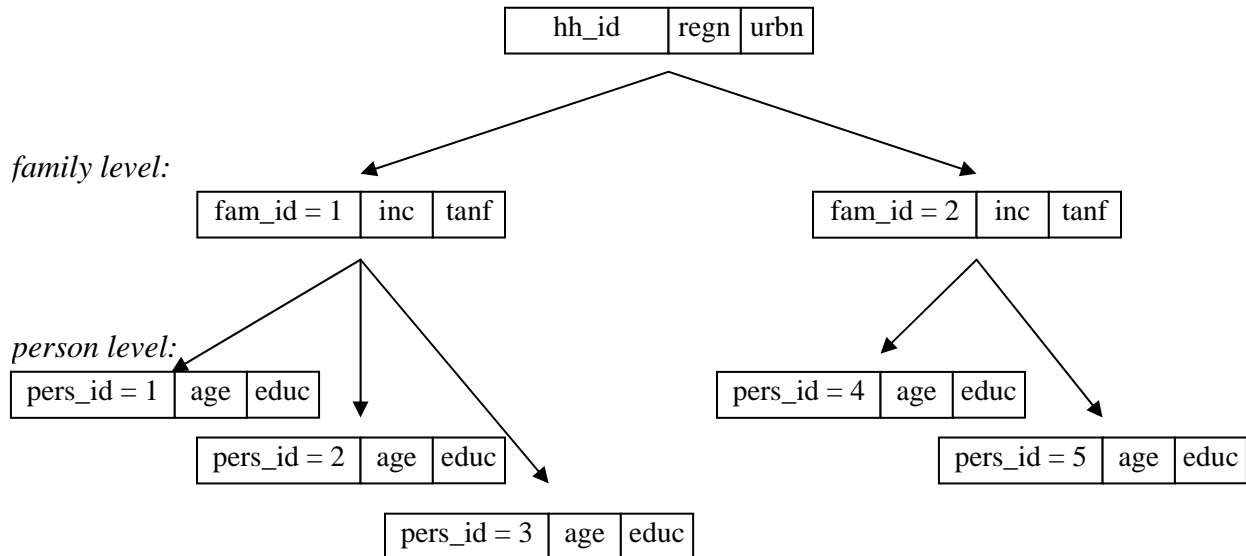
- database structures refer to the organization of a database’s observations and records
- in a “rectangular” or “flat” structure, all of the observations have the same record layout and refer to the same types of units
 - suppose that we had a data set in which each observation had a unique identifier, *obs_id*, and accompanying variables *x*, *y*, and *z*; a diagram of the record structure would be

obs_id	x	y	z
--------	---	---	---

- as you can see, the structure for a single observation is flat; when the observations are stacked on each other, the structure becomes a rectangle

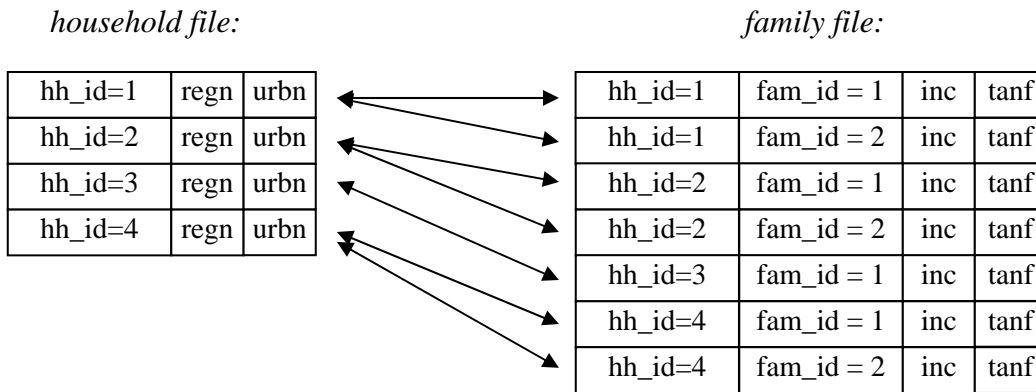
- in non-rectangular structures, there are different types of records referring to different types of organizations
- in a hierarchical structure, the relationship between the different types of records can be described in terms of a tree; the tree structure reflects nesting of logical relationships
 - a good example of a hierarchical structure is the Current Population Survey, which has records describing households, families within households, and persons within families
 - for a given household, assume that there is a household identifier, *hh_id*, and household descriptors, *regn* (region of residence) and *urbn* (urban residence)
 - for a given family, assume that there is a family identifier that is unique within the household, *fam_id*, and family descriptors, *inc* (family income) and *tanf* (family TANF receipt)
 - for a given person, assume that there is a person identifier, *pers_id*, and personal descriptors, *age* and *educ*

household level:



- the advantage of this structure is that it does not repeat information across lower levels, e.g., information for the household is only stored once; information for each family is stored once
- the shortcoming is that these types of files can be hard to work with
- in a relational structure, data are stored in several different flat files
 - each flat file corresponds to a type of observation
 - the relationships between the records and files are determined by key, or index, variables
 - for example, the CPS data *could* be stored in three different files
 - ◊ a household file with variables *hh_id*, *regn*, and *urbn*
 - ◊ a family file with variables *hh_id*, *fam_id*, *inc*, and *tanf*
 - ◊ a person file with variables *hh_id*, *fam_id*, *pers_id*, *age*, and *educ*

- ◇ families can be linked or matched to households through *hh_id*
- ◇ people can be linked to families through *fam_id* and linked to households through *hh_id*
- ◇ consider the relationship between the household and family files



- relational structures are more flexible than hierarchical structures; new relationships can be added through additional key variables
- rectangular files can be drawn or created from non-rectangular files
 - for example, with the information above we could construct person-level files that also contain all of the available family and household information for each person
 - creating rectangular files from non-rectangular files is the main way that we create analysis data sets in SAS

4) Match-merging in SAS

- files that are related by key variables can be match-merged in SAS
- the files should be sorted in order of the key variables
 - you can use PROC SORT
 - you can work with a permanent file that had previously been sorted
 - you can create a file that is sorted
 - if a file is sorted, this will be indicated in output from a CONTENTS procedure, so you can use PROC CONTENTS to check the sorting
- merging is a form of reading, so, it would take the place of a SET or INPUT statement in a DATA step
- the syntax for merging two SAS data sets would be

```

MERGE    <SAS_data_set_1> <(data_set_options)>
         <SAS_data_set_2> <(data_set_options)>;
BY       <DESCENDING> <ordering_var1>
         <<DESCENDING> <ordering_var2> ...
         <<DESCENDING> <ordering_var_n>>> <by_options>;

```

- each of the input data sets would need to include and be arranged by the ordering variables
- because MERGE is a type of reading command, you can use all of the input data set options with it
- MERGE will produce a data set with at least one observation for each read combination of BY values
- if MERGE can't find a match, it will read in the variables and values from the available record and pad the unmatched information with blanks
- MERGE continues reading data until both contributing files are exhausted
- the IN= data set option can be used to create a temporary variable that indicates whether a particular data set contributed an observation to the merged data set
- example,
 - consider the three files from our previous example; suppose that
 - ◊ the household file is a SAS data set, *hhdata*, that is sorted by *hh_id* and
 - ◊ the family file is a SAS data set, *famdata*, that is sorted by *hh_id* and *fam_id*
 - code to merge the data sets is

```
DATA thfmrg;
  MERGE   hhdata (IN=hhin)
         famdata (IN=famin);
  BY hh_id;

  IF hhin EQ 0 THEN
    PUT 'No household match for HH ' hh_id ', family ' fam_id;
  IF famin EQ 0 THEN
    PUT 'No family match for HH ' hh_id;
```

- assuming that every family matches to a household and that there are no households without families, the MERGE should produce one record per family (same length as the original family data set) but with all of the family and household variables (*hh_id*, *fam_id*, *inc*, *tanf*, *regn*, and *urbn*) in each record
- make sure that you include a BY statement when using the MERGE; if you don't, SAS will conduct one-to-one merging, linking the first records in each data set, the second records, etc.

5) Working with a sorted data set

- computing and storing simple statistics for BY groups is straightforward
 - these statistics can be computed by either PROC MEANS or PROC SUMMARY using a BY statement
 - for example, consider the person-level data set from above
 - ◊ suppose that it is stored in a SAS data set, *persdata*, that is sorted by *hh_id* and *fam_id*
 - ◊ we could calculate the highest education attained among the family members, the age of the youngest person in the family, and the age of the oldest person in the family with the code

```

PROC SUMMARY DATA=persdata;
  BY hh_id fam_id;
  VAR age educ;
  OUTPUT  OUT=perssum
          MAX=fmoldest fmmosted
          MIN(age)=fmyngest;

```

- note: the data set *perssum* would contain one record per family; this file could be merged directly (one-to-one) with *famdata* or merged one-to-many with *persdata*
- it is also possible to accomplish the same thing using a DATA procedure
 - to do this, we need to introduce one additional DATA step command, the RETAIN statement
 - in a normal execution phase, SAS assigns missing values to (erases the values in) all of the variables when it returns to the DATA statement at the end of an observation iteration
 - the RETAIN statement keeps SAS from doing this
 - the syntax for the statement is

```
RETAIN <list_of_variables>;
```

- the list of variables would not be overwritten with missing codes when SAS returns to the top of the DATA step
- now consider the following code that reads *persdata*, assuming that *persdata* is sorted by *hh_id* and *fam_id*

```

DATA perssum;
  SET persdata;
  BY hh_id fam_id;

  IF FIRST.fam_id THEN DO;                /* Initial obs. for family */
    fmoldest = age;                        /* Initialize family values */
    fmyngest = age;                        /* information for first pers. */
    fmmosted = educ;                       /* read */
  END;
  ELSE DO;                                  /* Subsequent obs. for fam. */
    fmoldest = MAX(fmoldest, age);         /* Compare ages and educs */
    fmyngest = MIN(fmyngest, age);        /* to current high/low values*/
    fmmosted = MAX(fmmosted, educ);
  END;

  RETAIN fmoldest fmyngest fmmosted;

  IF LAST.fam_id;                          /* Output last obs. for family */

```

- the above example is very artificial (because the same thing can be produced using the SUMMARY procedure); however, it illustrates how BY-group processing works
- BY-group processing within a single data set is very useful when working with longitudinal data
- if the data in two files are “grouped” but not sorted, you would use the following BY statement:

```
BY <ordering_var1> <<ordering_var2> ...> NOTSORTED;
```