

An Animated Guide©: Speed Merges: Formats

Russell Lavery, Contractor, Ardmore, PA

ABSTRACT

The format is a powerful tool for SAS programmers who have to access large data sets and wish to avoid sorting. Formats can be used for subsetting, or as replacement for an if. They can be used to perform the equivalent of a BY merge without sorting the data sets – a great advantage for processors of large data sets.

This presentation concentrates on:

- 1) the structure and contents of the file that Proc Format automatically creates
- 2) how a SAS programmer can use a data step to manually create a file of similar structure and content as a format file
- 3) how to use a format in processing large data sets.

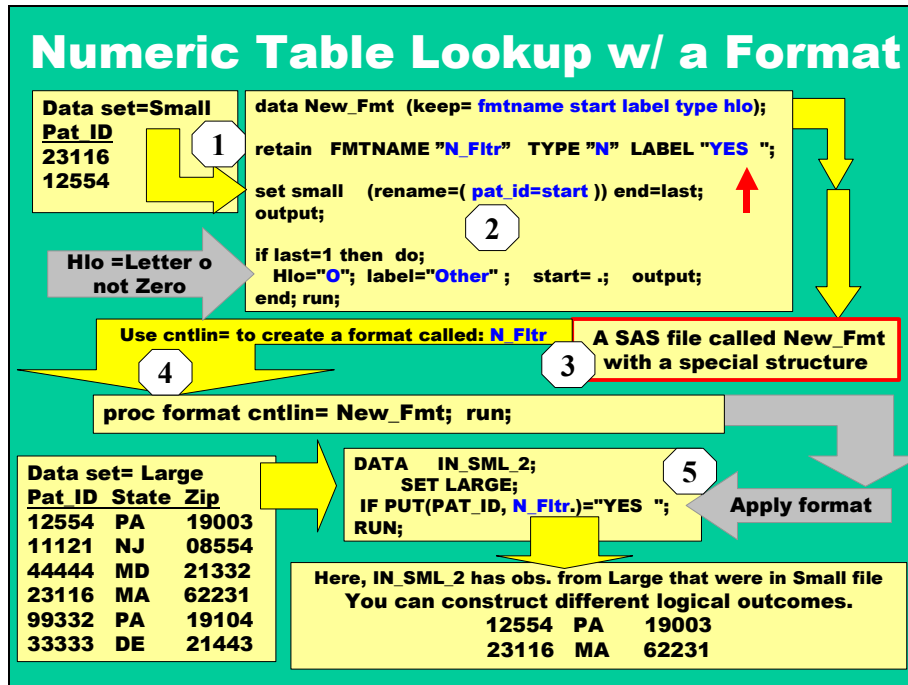


Figure 1

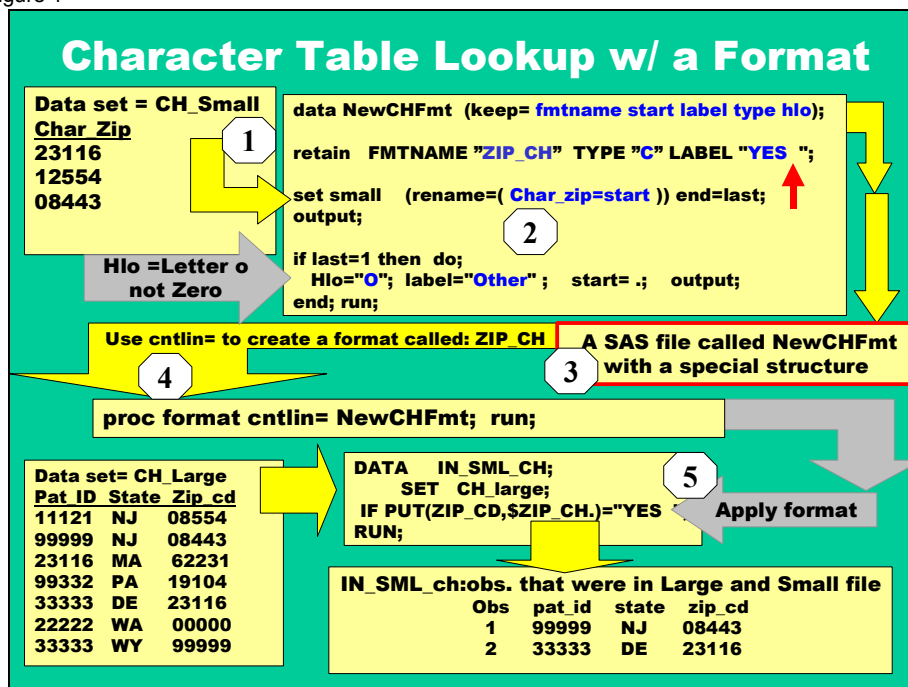


Figure 1a

INTRODUCTION

Figures 1 and 1a show the whole process of using a numeric and a character format as “table lookups”. The actual “format table lookup” is the “if put” statement (5 in Figure 1 or 1a) and is the process of taking a variable from one file and using it to look up information in another file (the word “file” is interchangeable with the word “table” in newer SAS documentation).

Programmers typically learn to do “table lookup” with a “by merge” using two sorted data sets. Use of the “sorted by merge” is a powerful and general technique but, with large files, sorting of data sets should be avoided. Sorting is a resource intensive operation, taking lots of time and lots of disk space. Even with modern computers it may not be possible to sort a very large file.

OVERVIEW OF FORMATS & FORMAT TABLE LOOKUPS

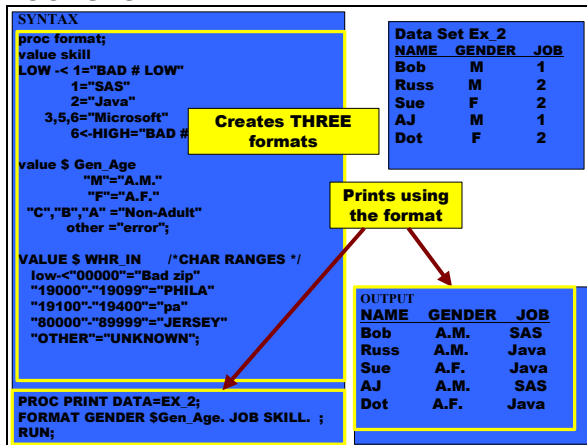


Figure 2

Formats are very useful SAS tools. Generally, formats determine how data is displayed. The data in the SAS data set is stored as the original value (in figure 2 they are 1,2,M,F) and SAS formats automatically “convert” these values to the formatted value when data is displayed. Formats can convert one character to another, a character to a number, a number to a character or a number to a different number. Formats can also be used to print using a template.

Figure 2 shows the normal creation, and typical use, of character and numeric formats. The large box on the left contains the syntax for a Proc Format and a Proc Print that applies those formats. A data set is in the upper right hand corner and the output of the Proc Print, applying the formats to that data set, is in the lower right.

Formats are usually created with a Proc Format and are often used to make output more readable. Formats can also be used to save disk space. In Figure 2, SAS can store a C in the SAS data set rather than the longer string “Non-Adult”.

When a format is used in a Proc (see Figure 2), the format has a temporary effect that lasts only while that Proc is executing. The line “FORMAT GENDER \$Gen_age. JOB SKILL.,” in the Proc Print in Figure 2 temporarily links the values of Gender and JOB to the formats \$Gen_age and Skill. That link is broken when that Proc Print finishes. A Proc Print run immediately following the one in our example would display a first line of “BOB M 1” (unless a format were applied to that Proc as well). However, if a format is applied to a variable, as part of data step syntax, it becomes permanently associated with that variable and will be automatically used in future output.

Use of “format table lookup” (Figures 1 and 1a) is a valuable programming technique because it can be used without sorting either data set. “Format table lookups” generally run much faster, and smaller, than “sorted by merges”.

In Figure 1, we only want to return two subjects from the file called Large and it would be easy to create an appropriate format with:

```
Proc Format ;
  Value Filter 12554,23116 ="Yes  ";
Run;
```

In Figure 2 we built a small format, by typing values into a Proc Format statement. However, if the format had thousands of levels, manually creating the syntax of a Proc Format would be tedious. This paper concentrates on using programming steps to create a format and to avoid typing in the format syntax (see Figures 1 & 1a: sections (1) (2) (3) (4)).

The “format table lookup” technique involves: a) building a SAS data set that has the same structure as a SAS format file, b) loading that file into a format catalog and c) using the format in a Proc or a put.

A large portion of the speed of the “format table lookup” comes from the fact that that is it a memory resident technique and avoids disk access. Theoretically there is no limit on the number of levels in a SAS format, however when the “format table lookup” executes the whole format must fit in ram memory.

When the format is too large to be loaded into ram, a programmer often tries an _IORC_ merge. An _IORC_ merge is generally slower than a “format table lookup”, but is not a RAM resident technique and faster than a “sorted by merge”.

The goal of this paper is to demonstrate how to programmatically use a “format table lookup” in replacement for a “sorted by merge” and to explore a mistake that programmers occasionally make with the “format table lookup”.

Especially important to understanding the “format table lookup” process is understanding the structure of the file you create (see red bordered box (3) in Figure 1 and the gold box in Figure 3). The process that is diagrammed in Figure 1 illustrates the use of a “small” file (1) as a source of information for a data step (2). The Data step creates an output file (3) that has the same variables and structure as a SAS format file. That output file (3) is loaded into the SAS format catalog (4) using the command “Proc Format Cntlin=filename;”. Finally, the format is used (5), in a second data step, as part of an “if statement” to select observations. The heart of this paper is the manual creation of the file (3) with the structure of an entry in the SAS format catalog and then loading that file into the SAS format catalog.

There are three side comments to be made about Figures 1 and 1a. First, constructing (3) (4) the file to be used in the cntlin= statement requires an understanding of the intended use of the put (5) as well as the format file structure. It is best to conceptualize the whole project (as shown in Figures 1 and 1a), and not just parts of it. Second, the “if statement” shown in Figure 1 could be modified to give different logical results. Finally, the “YES ” in the retain statement has two blanks inside the quotes. This is not required in these examples. This is a simple fix used in situations to allow the string “Other” be printed without truncation.

THE FORMAT CATALOG ENTRY

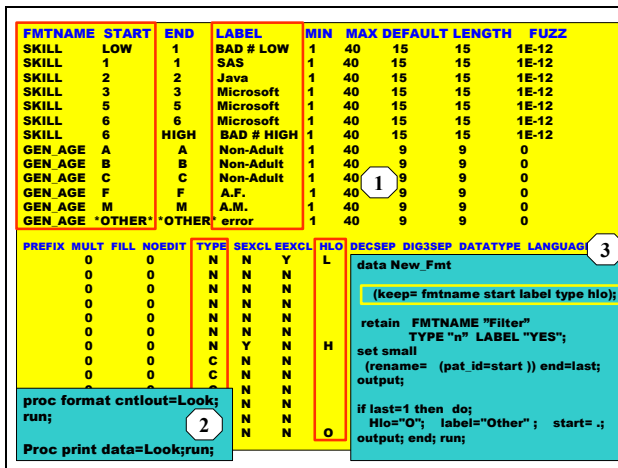


Figure 3

Normally, we do not need to think how SAS processes our requests to create and apply formats to data values. When we run Proc Format syntax, SAS automatically creates a format catalog entry out of the information we type in our Proc Format code.

UNLOADING THE FORMAT TO A SAS FILE

The contents of a SAS created catalog entry are shown in Figure 3. SAS allows us to copy the information from the format catalog into a SAS data file by using the command

```
Proc Format cntlout=sas_data_file;
Select format;
Run;
```

That command is shown in the blue box (2) in Figure 3. The gold area (1) in Figure 3 shows what we would see if we executed the commands in the blue box (2) in Figure 3. The blue box (2) is a request to copy the contents of the current format catalog into a file called Look and to print the file. The gold box shows the listing that results from printing Look. The gold box shows what SAS stored in the format catalog when we execute the Proc Format syntax shown in Figure 2.

IMPORTANT VARIABLES

The variables in the red boxes are the most important variables to a Format and are the variables that MUST be a part of the file you create for a format table lookup. Remember that the values in the gold box were created by the Proc Format in Figure 2. To save space, the format whr_in was not shown here.

The blue box (3) shows code we might employ to create a file with a structure similar to that shown in Figure 3. Note that the variables in keep statement of blue box (3) are the important variables in the SAS format catalog. The kept variables are: fmtname, start, label, type and hlo. We will discuss all the variables in the format catalog, but need to pay special attention to these 5.

To create familiarity with the structure of the cntlin file, several format files will be shown in this paper. Accordingly, Figure 4 shows the use a Proc Format (A) to create a format and a "Proc Format Cntlout=show" to copy the format information to a SAS file, called show, that is then printed. Note the use of the select in (B) to limit the amount of information copied to the SAS file show.

The information stored in the format catalog, shown in Figure 4, is described below:

(1) the name of the format. This name is used when the format is

applied.

(2) (3) The start and end variables determine a range. Values in a range will be converted to the value of label (4) on the same row of the file. If start and end are the same value, that row defines one specific value to be mapped to the value of label. In the format Paidup, in figure 4, the value N is both the start and end of its range. In this format, only N will be mapped to "Owes".

While figure 4 shows a single value range, ranges can be defined in character formats. Character ranges must be based on the sorting order for character variables on your operating system. Figure 5 shows a numeric range where start and end are different. For more examples, see the included SAS program in the appendix.

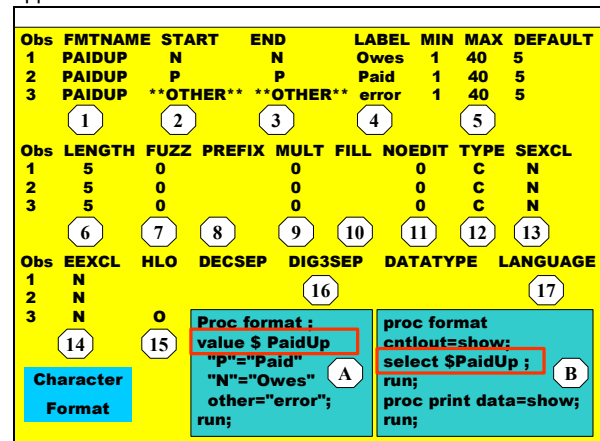


Figure 4

(4) label contains the value that will be displayed when the format is applied. Values between start and finish (subject to modification by fuzz, sexcl and eexcl) are displayed using the value in label.

(5) min, max and default define the minimum, maximum and default length of the format.

(6) length is the length of the format

(7) fuzz is not used in character formats but can be useful if the format is numeric. If a numeric value is to be formatted, and is the result of a SAS calculation, the small imprecision in computer calculations can cause problems. For example $Y=3 * 1/3$ might not return a value that is exactly one. Fuzz specifies that values within the value of Fuzz of start and finish can be considered as "matching" that range. Dangerously, Fuzz will interact with Sexcl and Eexcl.

(8) prefix specifies a prefix for the formatted value and an example is shown in (9) below.

(9) mult is a timesaver. It specifies a number by which the value to be formatted is to be multiplied, before the format is applied. An example is: picture million low-high='00.0M' (prefix='\$' mult=.00001); This code would format 1600000 as \$1.6M:

(10) fill specifies a character that completes the formatted value. It is used in picture formats and not often in table lookup.

(11) noedit specifies that numbers are message characters. This variable is used in picture formats and not often in table lookup. An example is: picture miles 1-1000='0000' 1000<-high='>1000 miles'(noedit);

(12) type is valued with either "c" or "n", indicating that the format should be applied to character or numeric variables

(13) (14) sexcl and eexcl indicate if the start and/or end of the range, defined in (2) (3), are excluded from that range. This will be illustrated later.

(15) hlo is an important variable. It can take values High (H). Low (L) or other (o). Hlo is valued by entering Low, High or Other on the left hand side of the equal sign in the value statement of a Proc Format command (see Figure 2). If other is the value in the Proc Format syntax, SAS also places the string "Other" in the start and end variables. It is important to note that where Hlo is 0 the

values in start and finish are only for reading convenience of programmers. If there is a letter "o" in Hlo, SAS ignores values in start and finish.

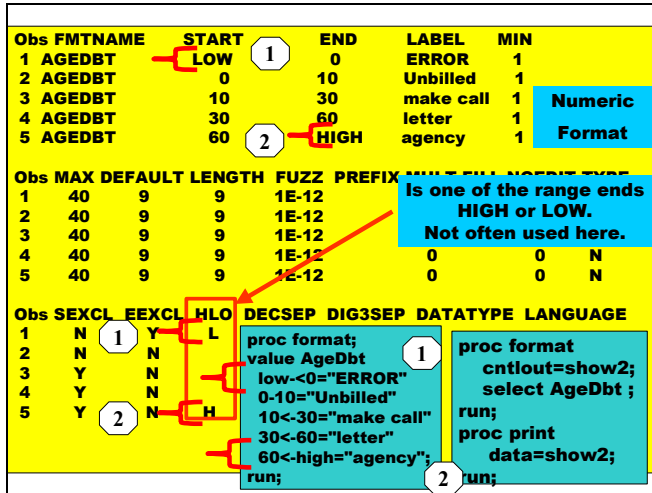


Figure 5

Figure 5 shows the Proc Format syntax that creates a format called AgeDbt and the contents of the SAS format catalog created by that syntax.

The first line of the value statement is worth noting. The range runs from Low (lower than any number that SAS can hold in a variable – but not including missing) up to, BUT NOT INCLUDING, zero.

The <- notation in the value statement specifies that low is in the range but 0 is not in the range. 0 will be mapped to the string "Unbilled". Similarly, 30 will be mapped to "make call", not to "letter".

For obs 1 in Figure 5, the variable Sexcl (Starting Value Excluded) is valued N (1) because the Starting value for the range (Low) is not Excluded from the range. The variable Eexcl (Ending Value Excluded) is valued Yes. The ending value of the range (the zero) is not part of the range. (Note: The final line in the value statement (2) has little practical value, it was constructed to show the use of High and to make Eexcl have a value of Yes.)

SAS code is attached to this paper that creates, and recalls from the catalog, several other formats. Due to space considerations, these files are not shown in this paper. The reader is encouraged to explore the attached code at his/her convenience

The final section of this paper shows, via a hypothetical project, a problem associated with the creation of the file to be loaded into the format catalog.

THE PROJECT & THE (SOMETIMES) PROBLEM

Figure 6 shows an overview of the process. The idea is to select, from a large file of hospital records (containing many many records), the patient zip codes for the patients who visited doctors Smith, Huan and Lee.

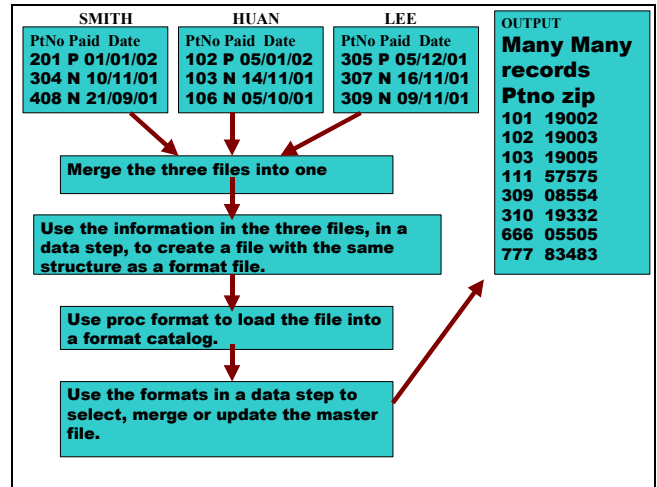


Figure 6

It is desired that any patient not associated with one of the three physicians should be labeled "other". The creation of the "other" category sometimes causes problems and deserves special study.

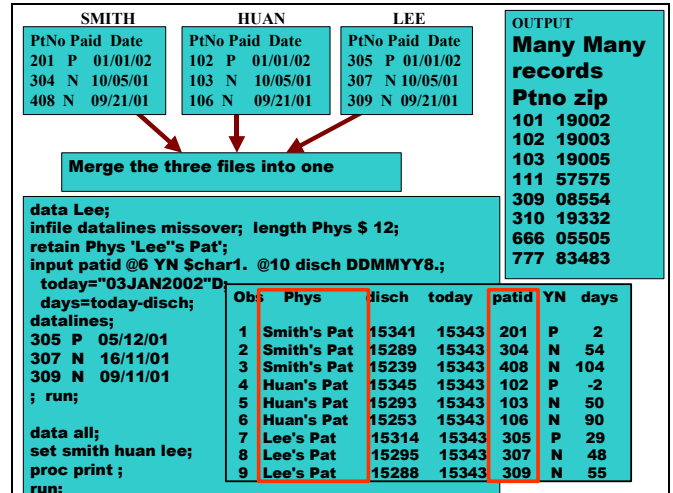


Figure 7

Figure 7 shows the creation of one of the data sets and the code to assemble the three files into one.

Figure 8 shows the code (with the label statement commented out as an intentional error) that would create a file called nifty1 that we could load into the format catalog.

Note, while label has been commented out of the code, the last line of the file nifty1 (shown in the gold box) does hold a value for label. That last value was "retained" in the PDV from the previous line. This unintentional retain, and a default SAS action, has occasionally caused some problems with "format table lookups".

SAS knows that a line with an "o" in Hlo should be describing "Other" observations. An "o" in Hlo, causes SAS to ignore values in the variables start and end. Unintentionally retaining the 309 onto the last line of Nifty1 does not cause problems. When Hlo is o SAS knows values in start and end must be errors.

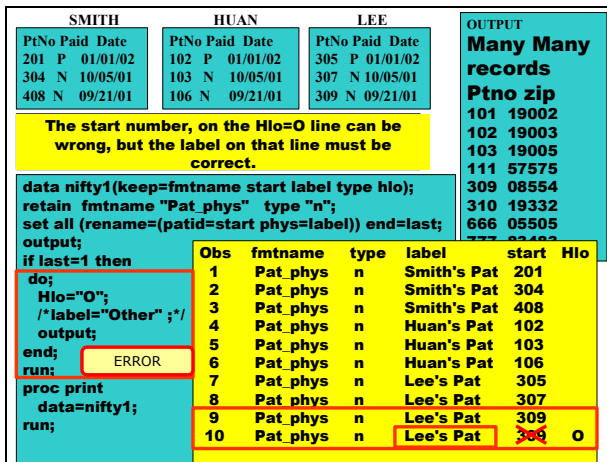


Figure 8

However, there is no logic that SAS can perform on the variable Label when Hlo is 0. Some programmers have expected that SAS, in the presence of an "o" in Hlo, would ignore values in start, end AND label. This idea is incorrect! The value of Label must be controlled by the programmer.

If the file Nifty1 (created in Figure 8) were used in a table lookup, erroneous classifications would result. All patients who were not in any of the physician files will be assigned to Dr. Lee as is shown in Figure 9. The value for label (in the small red box in Figure 8) is an error and needs to be fixed.

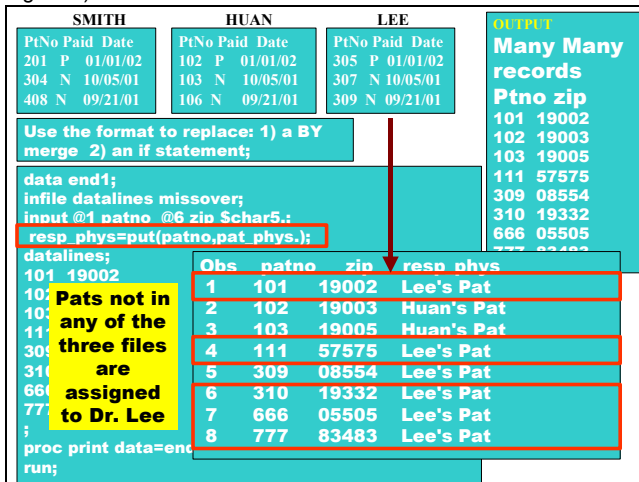


Figure 9

In Figure 9 we see the incorrect assignment of patients 101,111,310,666 and 777 to Dr. Lee.

The fix for the problem is to specify the value for label when the last line is written. That fix is illustrated in Figure 10. The reader is encouraged to run the included SAS code and verify that correct classification will result.

When the final line of the file is created, a value for Label must be written to the file. Remember, the "o" in Hlo will override an entry in start but will not override an entry in label. Figure 10 shows a file with the correct syntax and correct values for Label and Start. Note the red boxes in Figure 10

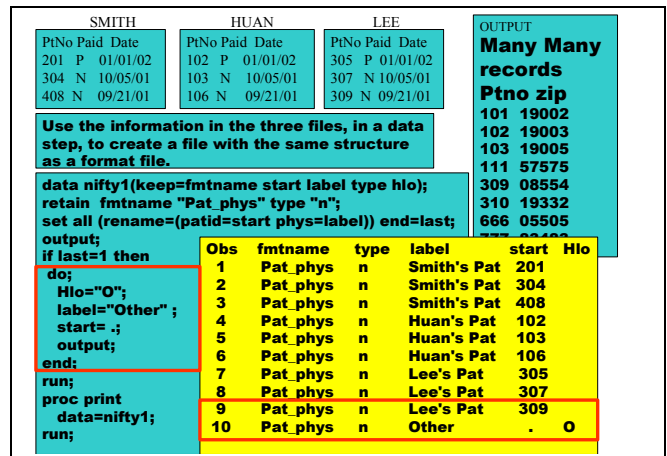


Figure 10

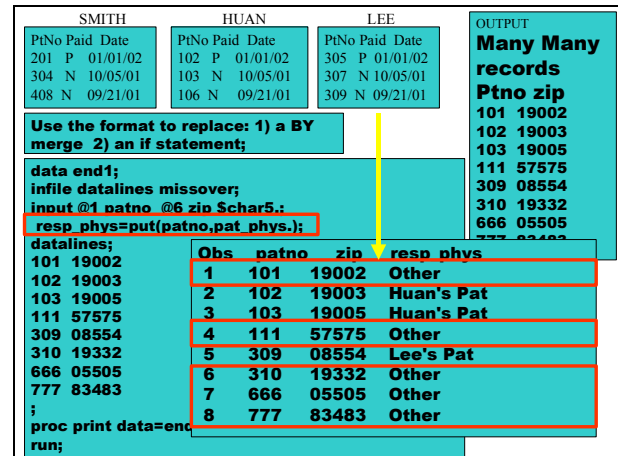


Figure 11

Figure 11 shows the result of using the file created in Figure 10 and correct assignment of patients to their physician and the assignment of "Other" to patients who did not visit any of our three doctors.

CONCLUSION

Figure 1 and Figure 1A are models for the use of character and numeric formats in a "format table lookup". Use of a data step and the cntlin= option in a Proc Format is a great time saver for the creation of large formats.

Programmatically creating formats makes is easy to use formats, a fast technique, for table lookup.

REFERENCES

Several years of proceedings are available online for free. A MS Access® database listing online SUGI/NESUG articles is available from the author. It can be searched for articles on formats, IORC, etc. that can then be downloaded from the web. Email rlavery@about-consulting.com for a copy.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Russell Lavery, Independent Contractor
 9 Station Ave. Apt 1 Ardmore, PA 19003
 610-645-0735 # 3 Email: russell.lavery@att.net