EFFICIENT DATA TRANSFORMATIONS
USING USER-DEFINED FORMATS AND INFORMATS
Kim L. Kolbe Ritzow
Systems Seminar Consultants, Kalamazoo, MI

Abstract
Regardless of your level of experience, this poster can help you understand the efficiencies that can be gained by transforming your data through user-defined formats and informats using PROC FORMAT. It will make a comparison with other data transformation methods like IF-THEN and SELECT statements in an attempt to show how user-defined formats and informats are a better means for data transformations within our SAS programs.

Background
One of the most common methods of performing data transformations within a SAS program is through IF-THEN statements. IF-THENs can be inefficient if you do not understand when they MUST be used, and when other alternative methods will transform the data more efficiently. This poster will explore the different transformation methods and compare their efficiencies.

The IF-THEN Statements
IF-THEN logic is executable code, which means it will happen as many times as there are observations to process, maybe more, depending on how efficiently you code your IF-THEN statements:

```
DATA CHANGEIT;
  LENGTH REGION $9;
  INFILE 'host-system-path-name';
  INPUT @1 STATE $2. @5 SALES 5.
    @15 STORENAM $25. @50 NOEMP 2.
    @55 DATE DATE7.;
  SET USSALES;
  IF STATE IN ('OH','IL','WI')
    THEN REGION = 'MIDWEST';
  ELSE IF STATE IN ('GA','SC','FL')
    THEN REGION = 'SOUTH';
  ELSE IF STATE IN ('OR','ID','WA')
    THEN REGION = 'NORTHWEST';
  ELSE IF STATE IN ('NY','CT','PA')
    THEN REGION = 'NORTHEAST';
RUN;
```

The above code is not as efficient as it could be because separate IF statements are used to check the same variable's values over and over again. If the data set USSALES contained a million observations, those million observations would be checked FOUR million times. Each separate IF would check the same million observations to see if it fits in to one of the categories, which wastes resources because once a state is found to be in one region, it cannot be in another region. Yet, the way the syntax is coded, we are unintentionally over-checking the data.

Further, this code is inefficient because ORs are used to logically connect conditions together when the IN operator could have produced the same result using less resources. The reason the IN is so much more efficient than the OR is that the IN does not require that the same variable constantly be re-coded, which saves on keystrokes. More importantly, the IN is more efficient because it stops checking the conditions when it finds the first one to be true. The OR operator implies that only one of the conditions has to be found to be true for the condition to be met, yet it actually checks ALL the conditions regardless if its found one to be true or not because it has no way of knowing if later on in the expression an AND or a NOT will be present which could reverse the entire logic. The previous code can be made more efficient by using ELSE/IFs and the IN operator:

```
DATA CHANGEIT;
  LENGTH REGION $9;
  INFILE 'host-system-path-name';
  INPUT @1 STATE $2. @5 SALES 5.
    @15 STORENAM $25. @50 NOEMP 2.
    @55 DATE DATE7.;
  SET USSALES;
  IF STATE = 'OH' OR STATE = 'IL' OR STATE = 'WI'
    THEN REGION = 'MIDWEST';
  IF STATE = 'GA' OR STATE = 'SC' OR STATE = 'FL'
    THEN REGION = 'SOUTH';
  IF STATE = 'OR' OR STATE = 'ID' OR STATE = 'WA'
    THEN REGION = 'NORTHWEST';
  IF STATE = 'NY' OR STATE = 'CT' OR STATE = 'PA'
    THEN REGION = 'NORTHEAST';
RUN;
```

```
PROC PRINT DATA=CHANGEIT;
RUN;
```
PROC PRINT DATA=CHANGEIT;
RUN;

This code is much more efficient than the previous code because it uses ELSE/IFs and IN operators, both of which will stop checking the data when it finds the first condition to be true rather than over-checking the data with separate IFs and logically joining conditions together with the OR operator. This code can be further improved by structuring the IF/ELSE/IFs and the items within the IN list in descending order of probability. This will stop the checking of data sooner when it finds the first condition to be true.

The IN operator DOES NOT take the place of the OR operator. The OR is still needed when checking the value of different variables (i.e.: IF STATE='WI' OR SALES > 500).

Efficiency Rules with IFs

- Always use ELSE/IFs and structure them in descending order of probability
- Always use INs to logically join data together when checking the same variable's value rather than ORs, and structure the items within the list in descending order of probability
- The use of the LENGTH statement to prevent truncation of values is not any more efficient than padding out the length of the character fields, but it is easier to maintain because you only have to change the value in once place rather than on each assignment statement

The SELECT Statement

Another way to transform data is with the SELECT statement. The SELECT statement is not new with Version 6, but people seldom, if ever, actually use it because they are more familiar with using the IF statement for data transformations. The SELECT statement produces the same results as an IF, but because of its rigid syntax structure it forces the code to be more efficient. With the IF, it requires the programmer to know that ELSE/IFs are more efficient than separate IF statements. The SELECT is comparable to ELSE/IFs because it stops checking when it finds the first value to be true.

The use of the LENGTH statement to prevent truncation of values is not any more efficient than padding out the length of the character fields, but it is easier to maintain because you only have to change the value in once place rather than on each assignment statement

PROC PRINT DATA=CHANGEIT;
RUN;

By structuring the WHENs and the values within the WHEN clauses in descending order of probability it can improve the efficiency. Take note that the SELECT uses an END (normally we associate ENDs only with DO loops), and that the OTHERWISE clause is required if you have data that does not fit in to any of the categories (you will get an abend, whereas with the IF, an uncategorized value will simply have a missing value for the newly created variable). The advantage of the SELECT statement over the IF, is that the SELECT statement's syntax forces you to be more efficient without even realizing it.

Efficiency Rules with SELECT

- Structure WHENs in descending order of probability
- Structure values within the WHEN clauses in descending order of probability
- Be sure to always code an OTHERWISE to prevent an abend when uncategorized data are encountered

The CASE Statement

The CASE statement within PROC SQL is another way in which we can transform data if the data comes from a SAS data set. Since PROC SQL cannot read a flat file directly, the Data Step is better at transforming data when it comes from a flat file, and SQL, when the data comes from a SAS data set:
PROC SQL;
SELECT SALES,
       CASE
           WHEN STATE IN ('OH','IL','WI')
               THEN 'MIDWEST'
           WHEN ('GA','SC','FL')
               THEN 'SOUTH'
           WHEN ('OR','ID','WA')
               THEN 'NORTHWEST'
           WHEN ('NY','CT','PA')
               THEN 'NORTHEAST'
           END AS REGION
FROM USSALES;

Efficiency Rules with the CASE

- Structure WHENs in descending order of probability
- Structure values within the WHEN clauses in descending order of probability
- A data set can be created in lieu of a report by using the CREATE TABLE statement prior to the SELECT statement

PROC FORMAT for Data Transformations
PROC FORMAT is one of the most efficient ways to do straight-forward transformation of your data:

PROC FORMAT;
   VALUE $REGFMT
   'OH','IL','WI' = 'MIDWEST'
   'GA','SC','FL' = 'SOUTH'
   'OR','ID','WA' = 'NORTHWEST'
   'NY','CT','PA' = 'NORTHEAST';
RUN;

PROC PRINT DATA=USSALES;
   FORMAT STATE $REGFMT.;
RUN;

PROC FORMAT has a number of advantages over the previously discussed methods. First, a new variable DOES NOT have to be physically created to contain the value. Formats make data LOOK LIKE they contain the value, but the value is still actually stored as 'OH', 'IL'. This saves space by not making the data set wider through adding a variable to it and the existing variable can maintain its internal length, which in this case is a 2-byte character ('OR') rather than a 9-byte character ('NORTHEAST').

Another advantage of using formats for data transformations is that they can be "lifted" out of the program and stored in a central location. This file can then be %INCLUDED into your SAS to program to make it look like it is actually coded within the program. The advantage of doing this is that the formats can exist in one location, but can be referenced in several locations within your own programs. This allows updating and maintenance to occur in one place, rather than in every program where the transformations are needed, which will result in more accurate and consistent reporting.

Why User-Defined Formats are Efficient in Transforming Data:

- Do not require the creation of additional variables on the data set
- Does not change the way the value is stored, it is for appearances only, so a value usually can be stored as a smaller value, but "look like" something larger
- Do not have to worry about the length of the value being assigned
- Can be "lifted out" and used in more than one program
- Oftentimes eliminates the need for a Data Step (anytime you can reduce the number of Data and PROC Steps in a program, the more efficient it will be)

Obviously, user-defined formats can not be used for all types of data transformations. It can do simple transformations of character and numeric data, but it cannot transform data based on complicated logic (i.e.: IF STATE='OH' AND SALES > 500).

Using User-Defined INFORMATS for Data Transformations

If you are never going to reference the data coming in by its original value ('OH', 'IL', etc.) and you prefer the variable to actually contain the value 'NORTHEAST', 'MIDWEST', etc., then one of the best ways to accomplish this is through a user-defined informat.

The ability to code user-defined informats is new starting with Version 6 and it gives us the ability to transform the data upon reading it in:
PROC FORMAT;
INVALUE $REGFM
'OH','IL','WI'='MIDWEST'
'GA','SC','FL'='SOUTH'
'OR','ID','WA'='NORTHWEST'
'NY','CT','PA'='NORTHEAST';
RUN;

DATA CHANGEIT;
LENGTH STATE $9;
INFILE 'host-system-path-name';
INPUT @1 STATE $REGFM2. @5 SALES 5.
@15 STORENAM $25. @50 NOEMP 2.
@55 DATE DATE7.;
RUN;

PROC PRINT DATA=CHANGEIT;
RUN;

If you would like to keep the variable's original value and would also like the ability to read the variable in with the user-defined format, it is possible to read the same field in two different ways:

INPUT @1 STATE $REGFM2. @1 STATENUM $2.;

User-defined informat are an effective way to perform simple transformation of our data values as the values are being read in. The values will be physically stored as 'NORTHEAST', 'SOUTH', etc. User-defined informat can not perform transformation that require complex logic (i.e.: IF STATE='WI' AND SALES > 500).

Why User-Defined Informats are Efficient in Transforming Data:

- Data can be transformed at the point of being read in, which reduces the number of times in which the data are passed
- Does not require the creation of additional variables on the data set
- Can be "lifted out" and used in more than one place

In Summary
There are many ways in which data can be transformed within our SAS programs. This poster has discussed the various methods and the efficiencies that relate to each method.

Choosing the best data transformation method is determined by the source of the data. If the data come from a flat file then use the following methods (in descending order of preference): user-defined informat, user-defined formats, IF/ELSE/IFs and the IN operator or the SELECT statement. If the data come from a SAS data set then use the following methods (in descending order of preference): User-defined formats called in a SAS PROC, PROC SQL, IF/ELSE/IFs and IN operators, and the SELECT statement.

The version, operating system, and platform in which an application is running under may all affect performance. As always, try a few methods and benchmark their efficiency to determine which transformation method is best suited for your particular application.

Trademark Notice
SAS is a registered trademark of the SAS Institute Inc., Cary, NC, USA and other countries.

Any questions or comments regarding the paper may be directed to the author:

Kim L. Kolbe Ritzow
Systems Seminar Consultants
Kalamazoo Office
927 Lakeway Drive
Kalamazoo, MI 49001
Phone: (616) 345-6636
Fax: (616) 345-5793