CORRECTING AND UPDATING EBCDIC FILES WITH SAS76

Marilyn Bodow, College of Medicine and Dentistry of New Jersey

ABSTRACT

For an environment with no advanced file edit facilities and where files are stored in EBCDIC format, SAS76 was chosen as the most convenient and flexible language to correct and update an ongoing data base. Advanced coding features were used to accomplish these objectives with minimum effort to the user and with the easiest possible portability of code among similar applications.

The corrections program, controlled by a programmer, requires that each record be assigned a unique key field for record identification. One of two options occurs:

1. Undesirable records are deleted with IF statements.

2. Records with specific column errors are overwritten with suspended PUT statements.

The update program is user controlled. The user submits updates of a specific subset of fields. The program checks to insure that the proper number of fields appear and that all characters are numeric. Cards failing this screening are listed and deleted. Those passing are used to update the old master file. Finally, the original master file record and the updated version are listed.

This SAS76 coding used in conjunction with IBM generation data sets and a case structure screening utility program (written in PL/I) establishes a complete maintenance facility for a data base of moderate activity.

INTRODUCTION

Although SAS has been developed primarily for statistical analysis applications, its wide range of flexible statements together with its many powerful functions have been key factors for choosing SAS as a programming language to correct and update an ongoing data base. This paper discusses a system designed to build and maintain a data base of moderate activity, concentrating on the innovative coding features used in two SAS programs: the correction program and the update program.

BACKGROUND

Before highlighting these features, a brief description of the type of data, the environment, and the system flow is appropriate. Input data, which typically concerns medical diagnoses, can be contained on several cards. However, for the purpose of this system each card is treated as a separate record.

All files are stored externally on disk in EBCDIC card image format as IBM generation data sets. For those unfamiliar with the terminology, IBM generation data sets are defined as a group of related data sets catalogued under the same generalized file name with a machine-generated qualifier appended to the name and automatically incremented for each run. In this system three generations (copies of the data base) are always kept, the current version and the previous 2 versions. This arrangement is useful for short term backup purposes. If a file is damaged while being updated an earlier version can be retrieved easily.

To insure that each record can be uniquely identified, each card containing raw data is assigned a stamp by the computer at the time of initial submission. This stamp consists of the run date, run time, and a sequence number and is preserved in columns 83-99 of each record stored on disk.

The data is screened for duplicate records and separated into either a master file or a rejects file by a PL/I utility program. Records with unique key fields are loaded into the master file which constitutes the data base. Key fields are a group of variables appearing on each record that together identify each record. For example, a patient identification number and a card number can be used. Complete duplicates are deleted - the original record is loaded into the data base - while all semi-duplicate records (cards containing the same key field values but different data field values) are put into a rejects file. Periodically the rejects file is corrected and then added to the data base via the PL/I screening utility. Schematic diagrams of the system are illustrated in Figures 1 and 2.

THE CORRECTIONS PROGRAM

The corrections program, controlled by a programmer, operates on the rejects file. Since this file houses semi-duplicate records the same key fields appear on more than one card. Therefore, the stamp, assigned to each record, acts as a unique record identifier enabling the program to accomplish its two functions: deleting undesirable records and overwriting records with specific column errors. In this program all corrections are specified by column locations, thus eliminating the need to define variables. This concept allows for maximum portability of the programming among

FIGURE 1
DATA MANAGEMENT FOR REGISTRIES
OR INFREQUENTLY-USED DATA BASES

SCHEMATIC DIAGRAM SHOWING INITIAL DATA ENTRY

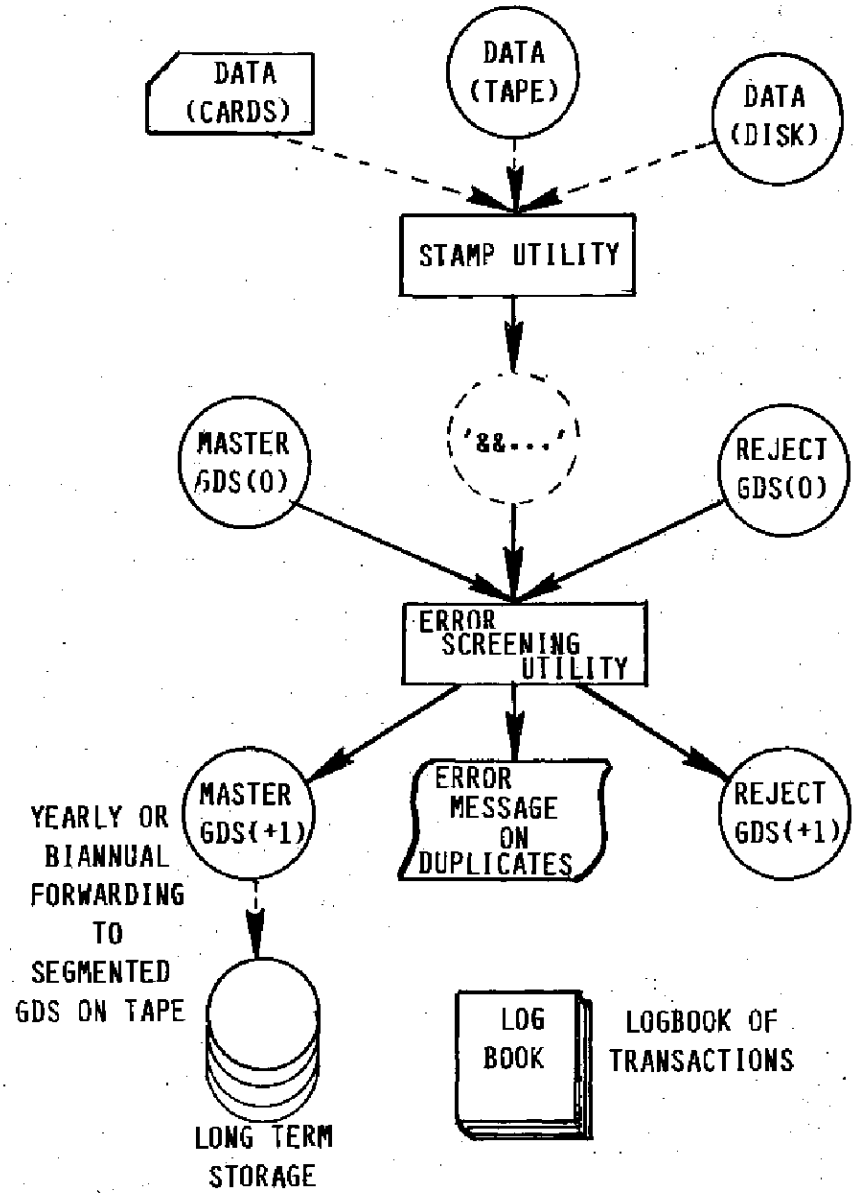
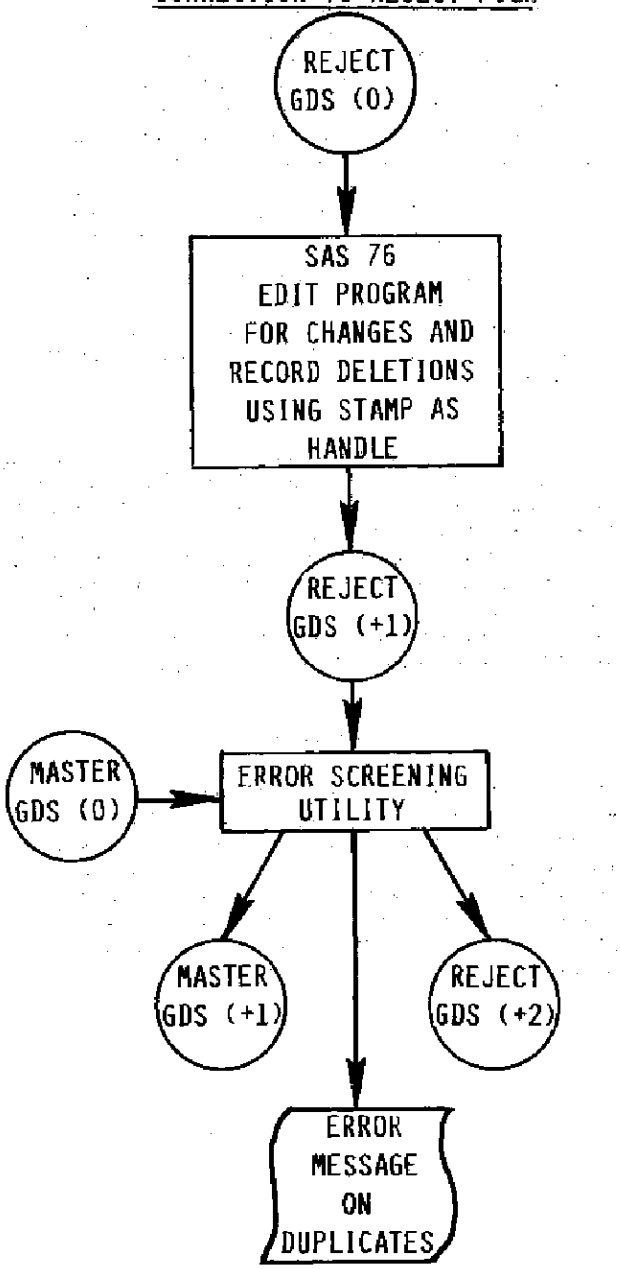


FIGURE 2
DATA MANAGEMENT FOR REGISTRIES
OR INFREQUENTLY-USED DATA BASES
SCHEMATIC DIAGRAM SHOWING
CORRECTION TO REJECT FILE



projects.

Figure 3. Excerpt From Corrections Program

```
 1.   DATA DELETE; INFILE IN;
 2.   INPUT @1 REC $CHAR99. @83 STAMP
        $CHAR17.;
 3.   IF STAMP='780219 2212 05032' THEN
        DELETE;
 4.   IF .......
      .
      .
      .
      .
10.   DATA FIX; SET;
11.   FILE OUT;
12.   PUT REC $CHAR99. @;
13.   IF STAMP='780219 2212 05033' THEN
        PUT @13 '0' @15 ' ' @16 '1210' @;
14.   IF .......
      .
      .
      .
      .
20.   PUT @81 ' ';
```

The first DATA paragraph simply deletes unwanted records by means of a series of IF statements. The next DATA paragraph reads in the remaining records, corrects them, and puts the corrected version of each record into an external OS file. In Figure 3 FILE OUT (statement 11) refers to the next generation of the rejects file. This second paragraph illustrates several novel coding techniques. By using a series of suspended PUT statements records are corrected. Statement 12 writes each original record leaving the pointer on that line. Note that SAS requires that PUT statement variables be formatted in the same way as the INPUT variables in order to preserve leading blanks in the $CHARw. format. Again, the stamp is used to identify individual records and the IF statements select those records to be corrected. When writing to an external file with a suspended PUT statement as in statement 12 several anomolies can occur. In order to eliminate unwanted intermittently spaced blank records from appearing in the file, all IF statements (statement 13 .....) must end with a trailing @. Without statement 20 the file produced may have multiple records concatenated and/or may be missing entire records.

THE UPDATE PROGRAM

In the application illustrated in this paper, one or more of 5 numeric diagnoses are frequently changed. SAS's UPDATE function can update a data base easily. The update program is completely controlled by a lay user. The user merely inserts the correction data cards in the appropriate place and submits the deck. Each card must contain an identification number, 5 diagnoses, and a checking field. Each variable is separated by at least one blank and must appear in a specific order.

If any of the diagnoses is to remain unchanged, the user punches a "." in its appropriate place.

Figure 4. Excerpt From Error Checking Routine

```
1.  DATA DIAGCHK (KEEP=ID DIAG1 DIAG2
       DIAG3 FDIAG MDIAG);
2.  INFILE IN MISSOVER;
3.  INPUT ID_A $ D1 $ D2 $ D3 $ FD $ MD
       $ CHK_A $ @1 ID DIAG1 DIAG2 DIAG3
       FDIAG MDIAG @1 CARD $CHAR80.;
4.  IF ID NOT=.
       AND (DIAG1 NOT=. OR (DIAG1=. AND
                            D1=' '))
       AND (DIAG2 NOT=. OR (DIAG2=. AND
                            D2=' '))
       AND (DIAG3 NOT=. OR (DIAG3=. AND
                            D3=' '))
       AND (FDIAG NOT=. OR (FDIAG=. AND
                            FD=' '))
       AND (MDIAG NOT=. OR (MDIAG=. AND
                            MD=' '))
       AND CHK_A='#' THEN GO TO GOOD;
```

The first section of the program is designed to check each card to insure that the correct number of variables are given and that all values are numeric. Since the updating is done directly to the data base, it is essential that the initial step be to eliminate all cards containing bad data. Ordinarily, SAS continues reading cards until all variables are found. This could have disastrous consequences in the data base since variables could be updated with wrong values if the user neglected to supply the correct number of fields on each card. Thus the SAS MISSOVER option is employed forcing SAS to read only one card per observation.

To insure that all variables on each card are assigned a value the checking field called CHK_A was created. This variable should have a value of '#'. If in fact CHK_A does equal '#' then all variables have been accounted for and have acceptable values.

The numeric check relies on SAS's "@" symbol in the INPUT command. Statement 3 in Figure 4 illustrates how the variables are first declared as characters ($ is used to permit free formatting of the cards) and then overlayed with the new names that are declared as numerics. This dual declaration of variables allows the programmer full control over error messages to the lay user. The actual screening, accomplished with a single IF statement, assures that no erroneous characters appearing in the input data are transmitted to the program. For example, in Figure 4, diagnosis 2 is acceptable if its declared numeric value is not missing or if both its declared numeric value and its declared character value are missing.

The rest of the program does the actual updating of the data base using the UPDATE function and those cards passing the screening. The resulting printout is a series of messages to the user explicitly listing all bad correction data cards, all cards that have identification numbers not on the master file, and finally, both the original and revised form of all records that actually have been updated (see Figure 5).

CONCLUSION

The two SAS programs when combined with IBM generation data sets and an error screening utility program provide a complete data base maintenance facility, requiring minimal project-specific programming. The system has been successfully applied to a genetics research project and can be easily adapted to similar applications.

FIGURE 5

SCHEMATIC DIAGRAM OF UPDATE PROGRAM