Using SAS® For Data Extraction And Analysis

Robert W. Ingram (E-mail: ringram@cba.ua.edu), University of Alabama

Abstract

Data extraction and analysis are common uses of computers by accountants and auditors. This paper provides an introduction to SAS® and a tutorial exercise in using SAS for data extraction and analysis purposes. It provides example programs for extracting data from a relational database and for such common accounting purposes as computing and aging receivables, computing accounting ratios, grouping, outputting, describing, and filtering data, identifying duplicate and missing observations, and identifying dollar unit samples. The exercise can be used to introduce students to SAS and to data extraction and analysis techniques.

Introduction

A common use of technology by accountants is for accessing and evaluating computerized accounting data. Large databases are repositories of accounting data in many organizations. Consequently, the ability to extract, combine, and analyze these data efficiently is important for many accounting related tasks. The extraction and analysis tasks often are complicated by the need to access data from a variety of computer platforms and operating systems. Accordingly, accountants need tools that are capable of handling large amounts of data without having to expend significant resources to retrieve and reformat the data.

The SAS®[1] package of programs from the SAS Institute provides a widely-used and relatively simple solution for the extraction and analysis problem. This paper provides a classroom exercise for introducing accounting students to SAS programs and how they can be used for data extraction and analysis. The exercise can be used as a tutorial for students to gain experience in using the computer to extract, manipulate, and analyze data for tasks commonly encountered by accountants.[2]

The next section of this paper describes some advantages of using SAS for data extraction and analysis. The second section explains basic components of the SAS language. The third section provides an exercise that uses SAS to access a relational database, to merge files from the database, and to perform a series of data analysis tasks, typical of those performed by auditors. The last section provides a brief summary and suggestions for future research.

Reasons for Using SAS

SAS is used in a variety of business and education settings. It is readily available to students in many colleges, as well as to auditors and managers in many firms and companies. It is used by 3.5 million users at over 31,000 sites in over 120 countries (Delwiche and Slaughter,

Readers with comments or questions are encouraged to contact the authors via email.

1998). Consequently, accountants can use software that already exists in many organizations or can load core programs on a laptop, link to a client network, and read client data without having to recomputed, reformat, or otherwise manipulate the data.

SAS is capable of reading almost any data from almost all operating systems and platforms, from spreadsheets to relational databases, without the need to convert the data to an intermediate form. It is capable of handling unlimited numbers of observations, subject only to the memory capacity of the computer used for the analysis task. Programs written in SAS for one operating system are portable to other systems. SAS runs off of laptops, client-server systems, and mainframes. In addition, SAS is a common application in business environments.

SAS provides a large number of pre-programmed procedures that are easily implemented for examining most types of data analysis issues. These range from simple descriptive statistics to complex comparisons. Most of the analysis issues can be handled by a core set of SAS programs that are easily learned. One of the major advantages of SAS is that the user can control many attributes of these programs rather than having to rely on a limited set of predetermined techniques. Accountants who discover unexpected problems in a particular data environment, often can create data manipulation and analysis procedures on an ad hoc basis to deal with those problems without the need to recollect and reformat data.

SAS Basics

SAS can be run from a graphical interface in Windows and Unix (x-windows) environments. Nevertheless, the system relies on a simple programming language that can be used in almost any environment. The SAS language is not complicated but provides programming capability to meet a large variety of user needs. It is this combination of being relatively simple but enormously flexible that makes SAS a popular tool.

SAS programs are composed of statements. The statements are not case sensitive, are not sensitive to word or line spacing, and each statement ends in a semi-colon. SAS reads from one line to the next until it encounters a semi-colon. Most SAS statements fit into one of two categories, DATA steps and PROC (procedure) steps. DATA steps identify the location of data files and define variables and their formats. PROC steps describe how the data will be manipulated and analyzed. The first record is read from the file and processed. Then, the next record is read and processed until the end of the file is reached.

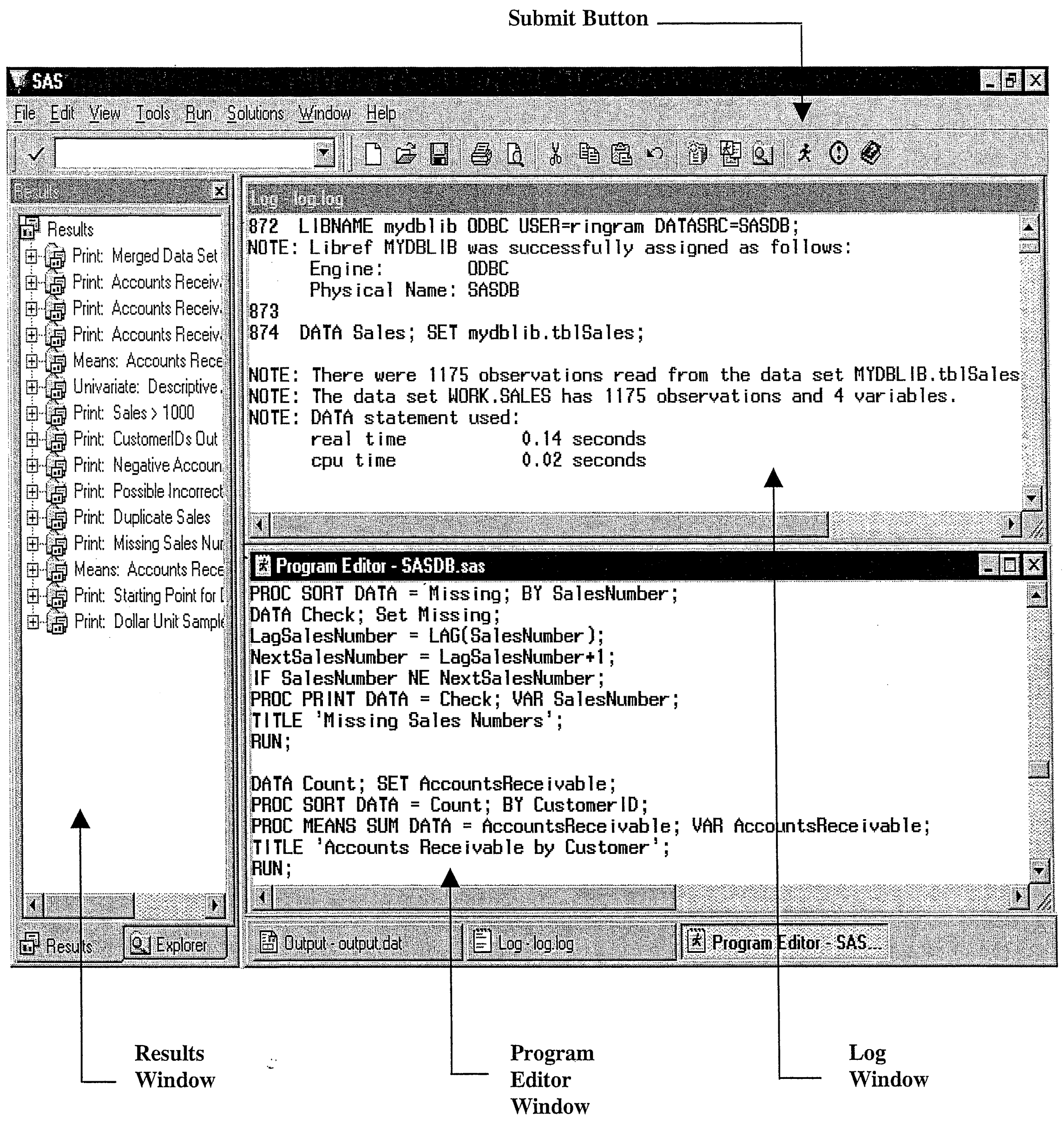
Figure 1 provides a screen view of the SAS interface from a Windows environment. The view is separated into sections, identified by tabs at the bottom. Clicking on a tab makes that view active. The primary sections that users work with are the Program Editor, Log, and Output windows. The Program Editor and Log windows are visible in Figure 1. Individual windows can be maximized on the screen and the tabs can be used to toggle among the windows.

The Program Editor is used to enter SAS programs. Programs can be keyed into the window, opened from existing files, are copied from a word processor or other source. Programs are run by clicking on the Submit Button at the top of the window.

Once a program is executed, the output appears in the Output Window. It can be saved to file or copied to the clipboard and pasted into a word processor. The Log Window lists each step in the program and any errors that occurred in the processing of that step. Accordingly, it is a valuable tool in discovering programming or data problems and should always be examined when a program is executed.

The Results Window provides a tree dia-

Figure 1
SAS Program Entry Window



```
872  LIBNAME mydblib ODBC USER=ringram DATASRC=SASDB;
NOTE: Libref MYDBLIB was successfully assigned as follows:
      Engine:        ODBC
      Physical Name: SASDB
873
874  DATA Sales; SET mydblib.tblSales;

NOTE: There were 1175 observations read from the data set MYDBLIB.tblSales
NOTE: The data set WORK.SALES has 1175 observations and 4 variables.
NOTE: DATA statement used:
      real time           0.14 seconds
      cpu time            0.02 seconds
```

```
PROC SORT DATA = Missing; BY SalesNumber;
DATA Check; Set Missing;
LagSalesNumber = LAG(SalesNumber);
NextSalesNumber = LagSalesNumber+1;
IF SalesNumber NE NextSalesNumber;
PROC PRINT DATA = Check; VAR SalesNumber;
TITLE 'Missing Sales Numbers';
RUN;

DATA Count; SET AccountsReceivable;
PROC SORT DATA = Count; BY CustomerID;
PROC MEANS SUM DATA = AccountsReceivable; VAR AccountsReceivable;
TITLE 'Accounts Receivable by Customer';
RUN;
```

gram that identifies output for each of the procedures in a program. The user can click on any of the items to examine output for that procedure, which appears in the Output Window. This window is useful if a program contains several procedures and the user needs to refer to the output in non-sequential order. The Explorer Window provides access to SAS files and programs created by the user, much like Windows Explorer.

A Data Extraction and Analysis Exercise

The exercise described in this section was developed on a Windows NT 4.0 operating system and a SQL Server 7.0 database.[3] The database contains two tables of interest for the example. One is a Sales table containing a Sales Number (the primary key), a Customer ID, a Sales Date, and a Sales Amount. The other is a Cash Receipts table containing a Cash Receipts Number (the primary key), a Sales Number (a foreign key for the Sales table), a Cash Receipts Date, and a Cash Receipts Amount. The tables contain data for a hypothetical company for the fiscal year beginning January 1, 2000.

Extracting Data

Data extraction is a simple task. The Windows version of SAS reads directly from native Windows programs such as Excel and Access and from ODBC (Open Database Connectivity) linkable databases. ODBC is beyond the scope of this paper, but connections can be made easily for most database systems by creating data source connection files using the Data Source Administrator in Control Panel. Figure 2 provides a program for extracting and merging the data in the Sales and Cash Receipts tables. Program lines are numbered for ease of reference.

Line 1 in the program creates a reference name, "mydblib," to identify the input data. The remainder of line 1 identifies the source as an ODBC connection. USER is the user name required to access the database. A password would be provided if needed. DATASRC is the data source name (DSN), Sasdb, used to identify the ODBC connection to the SQL Server database. This name was created by the user when the ODBC connection was created. As mentioned earlier, SAS is not case sensitive. Terms in all caps in the programs refer to SAS keywords to differentiate them from user supplied labels.

Lines 2 and 3 read data from the database tables and save these as "Sales" and "Receipts" respectively. The SET procedure reads an input file (tblSales) from the mydblib database into a SAS dataset, called "Sales." The data are read directly from the database and fields in the tables are read to SAS using their database labels and formats. There is no need to specifically identify

Figure 2
SAS Program to Extract and Merge Datasets

```
 1  LIBNAME mydblib ODBC USER=ringram DATASRC=Sasdb;
 2  DATA Sales; SET mydblib.tblSales;
 3  DATA Receipts; SET mydblib.tblCashReceipts;
 4  PROC SORT DATA = Sales; BY SalesNumber;
 5  PROC SORT DATA = Receipts; BY SalesNumber;
 6  DATA SalesReceipts; MERGE Sales Receipts; by SalesNumber;
 7  DSalesDate = DATEPART(SalesDate);
 8  DCashReceiptsDate = DATEPART(CashReceiptsDate);
 9  PROC PRINT DATA = SalesReceipts (FIRSTOBS = 1000 OBS = 1010);
10  VAR SalesNumber CustomerID DSalesDate SalesAmount CashReceiptsNumber
11     DCashReceiptsDate CashReceiptsAmount;
12  FORMAT DSalesDate DATE. DCashReceiptsDate DATE.;
13  TITLE 'Merged Data Set'; RUN;
```

the variables or their locations in the data records.

Lines 4 and 5 sort each of the datasets by SalesNumber. SAS requires that datasets be sorted by the same fields before files can be merged. Line 6 merges the two datasets, creating a dataset named "SalesReceipts". Lines 7 and 8 format the date fields in the dataset so that they are read as simple dates (for example, 01/01/00) rather than as datetime (including hours, minutes, and seconds) fields that are native to SQL Server. Line 9 writes (using the SAS procedure PROC PRINT) some of the records to the Output Window so the user can ensure that the data are consistent with expectations. FIRSTOBS identifies the starting point for the records to be written, record 1,000. OBS identifies the last record to be written. Lines 10-11 list the variables (VAR) to be written to output, and line 12 formats the date fields as dates (DATE.) for the output. SAS stores dates as the number of days since January 1, 1960. Formatting converts these dates to conventional values. Line 13 provides an optional title for the output and tells the program to begin execution.

Once the program is submitted, the log file describes each step that was executed and whether it executed properly. Figure 3 provides an excerpt for the extraction program, indicating the program steps executed properly.

The Output Window contains the data written by the program, as illustrated in Figure 4. The figure contains a limited set of observations and records to illustrate the type of output provided.

The extraction example presented above uses SAS program statements. SAS is also capable of using SQL statements to extract and join files. The program in Figure 2 merged the datasets and created null fields for cash receipts that did not match sales. This procedure would require an Outer Join in SQL but is easily accomplished by those adept at SQL.

Computing and Aging Accounts Receivable

As an example of a data analysis task, the program in Figure 5 computes accounts receivable as the difference between sales and cash receipts and determines the age (in days) for each receivable.

Line 1 creates a dataset labeled AccountsReceivable from the SalesReceipts dataset. Line 2 converts null values for cash receipts to zeroes. Line 3 computes accounts receivable as the difference between sales and cash receipts amounts. Line 4 formats the variable as a dollar value with a maximum of 10 digits with 2 digits right of the decimal point. Line 5 defines the date for aging receivables as December 31, 2000 and identifies it as a date (D) variable. Line 6 restricts analysis to records for which accounts receivable exist (> 0), and Line 7 computes the age of those receivables relative to the analysis date. Lines 8-13 place the records into groups based on the age

Figure 3
SAS Log Example

```
LIBNAME mydblib ODBC USER=ringram DATASRC=SASDB;
NOTE: Libref MYDBLIB was successfully assigned as follows:
      Engine:        ODBC
      Physical Name: SASDB
DATA Sales; SET mydblib.tblSales;
NOTE: There were 1175 observations read from the data set MYDBLIB.tblSales.
NOTE: The data set WORK.SALES has 1175 observations and 4 variables.
NOTE: DATA statement used:
      real time           0.14 seconds
      cpu time            0.02 seconds
```

Figure 4
Merged Data Set

```
                                      DSales
Obs        SalesNumber   CustomerID   Date          SalesAmount
1000       2001          48           05NOV00       $422.70
1001       2002          18           06NOV00       $631.94
1002       2003          54           06NOV00       $456.28
1003       2004          32           07NOV00       $379.24
1004       2005          64           07NOV00       $248.40
```

Figure 5
SAS Program to Compute and Age Accounts Receivable

```
 1  DATA AccountsReceivable; SET SalesReceipts;
 2  IF CashReceiptsAmount = . THEN CashReceiptsAmount = 0;
 3  AccountsReceivable = SalesAmount - CashReceiptsAmount;
 4  FORMAT AccountsReceivable DOLLAR10.2;
 5  AnalysisDate = '31DEC2000'D;
 6  IF AccountsReceivable > 0 THEN AccountsReceivableAge =
 7     AnalysisDate - DSalesDate;
 8  IF AccountsReceivableAge <= 30 THEN AgeGroup = 'Current';
 9  IF AccountsReceivableAge > 30 AND AccounstReceivableAge <= 60
10     THEN AgeGroup = 'Over 30';
11  IF AccountsReceivableAge > 60 AND AccounstReceivableAge <= 90
12     THEN AgeGroup = 'Over 60';
13  IF AccountsReceivableAge > 90 THEN AgeGroup = 'Over 90';
14  PROC PRINT DATA = AccountsReceivable (FIRSTOBS = 1000 OBS = 1010);
15  VAR AccountsReceivable AnalysisDate DSalesDate AccountsReceivableAge;
16  FORMAT AnalysisDate date. DSalesDate date.;
17  TITLE 'Accounts Receivable Data';
18  RUN;
```

of the receivables. IF ... THEN statements permit the user to group data or otherwise filter data for analysis according to user defined criteria. Lines 14-18 write some of the records to output as described in Figure 2.

An example line of output from the program appears below showing the observation number, amount of the receivable, the analysis date, the sales date, and the age of the receivable in days:

```
1002    $456.28 31DEC00    06NOV00    55
```

It is always useful to review output and log files to ensure the program is performing according to expectations.

Figure 6 provides a program to compute the total amount of accounts receivable in each of the age groups computed in Figure 5. Line 1 sorts the data by age group. Line 2 computes the sum of accounts receivable in each age group. The MEANS procedure is capable of providing a variety of descriptive statistics for variables. N and SUM identify the particular statistics required in this analysis. N is the num-

Figure 7
Accounts Receivable by Age Group

```
                  AgeGroup=Current
     N                           Sum
  1052                    $34,774.28
                  AgeGroup=Over 30
     N                           Sum
    55                    $26,091.35
                  AgeGroup=Over 60
     N                           Sum
    22                    $12,928.85
                  AgeGroup=Over 90
     N                           Sum
    46                    $27,796.53
```

Figure 8
SAS Program to Compute Receivables Ratios

```
1 PROC MEANS SUM DATA = AccountsReceivable NOPRINT;
2 VAR SalesAmount AccountsReceivable;
3 OUTPUT OUT=SumSalesReceivables SUM=SumSales SumAccountsReceivable;
4 RUN:
5 DATA Ratios; SET SumSalesReceivables;
6 DaysSalesinAR = SumAccountsReceivable/(SumSales/365);
7 ARTurnover = SumSales/SumAccountsReceivable;
8 PROC PRINT DATA = Ratios;
9 TITLE 'Accounts Receivable Ratios'; RUN;
```

ber of observations, and SUM is the total. The BY command results in calculations being presented for each group in the BY statement. The VAR statement indicates the variable to be analyzed.

Output from the program is shown in Figure 7. Uncollectible accounts could be estimated by multiplying by an expected collection rate for each group.

Computing Ratios

Computing ratios is a simple matter of totaling variables and calculating ratios, as illustrated in Figure 8. Lines 1-2 calculate the total amount of sales and accounts receivable for the company. The NOPRINT command prevents the data from being printed to output by the MEANS procedure. Output is stored instead in the SumSalesReceivables dataset in line 3. Line 5 reads the output file. Lines 6-7 calculate ratios, and lines 8-9 write the ratios to output.

Output appears as follows:

```
Sum              Sum           Days       AR
Sales          Accounts       Salesin   Turnover
              Receivable        AR
$594,975.09   $101,591.01     62.32      5.857
```

The PRINT statement in Figure 8 does not restrict the variables to be written; so all variables in the dataset are written to output.

Grouping and Outputting Data

The BY statement is a common means of grouping observations. For example, Figure 9 provides a program for computing the amount of receivables by customer. Grouping requires data to be sorted by the grouping variable, line 1. Lines 2-3 calculate the total amount of receivables for each customer. If the NOPRINT option is not used, data for all customers will be written to output. Using the NOPRINT option and lines 4-6 to output the data permits the user to restrict the number of observations written.

Output shows the Customer ID, number of receivables items for each customer (_FREQ_) and total amount of receivables:

```
                                     Sum
                                   Accounts
Obs    CustomerID     _FREQ_      Receivable
1          1             8          $718.15
2          2            12            $0.00
3          3             7        $1,614.76
4          4            15        $1,617.74
5          5            12          $560.17
```

The output from the Means procedure can be written to a permanent file as shown in Figure 10. The FILE statement defines where the file will be written, and the PUT statement identifies the variables to be written to the file and their format. Customer ID is written in columns 1-10, and accounts receivable is written as a floating point number with two decimal places. The +1 format includes a space between the variables. The file can be read by a word processor or a spreadsheet. Also, it can be read by SAS for additional processing. The statements to read the file would be identical to those for writing, except that INFILE would be substituted for FILE and INPUT would be substituted for PUT.

Describing and Filtering Data

SAS provides several methods for obtaining descriptive statistics. One of the simplest is the Univariate procedure. Figure 11 contains a program to execute this procedure. Running procedures in SAS is a simple matter of calling the procedure and identifying the variable to be analyzed. Only one variable can be analyzed at a time with PROC UNIVARIATE.

Figure 9
SAS Program to Compute Total Accounts Receivable by Customer

```
1 PROC SORT DATA = AccountsReceivable; BY CustomerID;
2 PROC MEANS SUM DATA = AccountsReceivable NOPRINT; BY CustomerID;
3 VAR AccountsReceivable;
4 OUTPUT OUT=SumAR SUM=SumAccountsReceivable; RUN;
5 PROC PRINT DATA=SumAR (OBS = 10);
6 TITLE 'Accounts Receivable by Customer'; RUN;
```

Figure 10
SAS Program to Write Output to a Permanent File

```
DATA ARbyCustomer; SET SumAR;
FILE 'c:\My Documents\AcctsRec.dat';
PUT CustomerID 1-10 +1 SumAccountsReceivable 8.2;
```

Figure 11
SAS Program to Compute Descriptive Data

```
PROC UNIVARIATE DATA = AccountsReceivable; VAR SalesAmount;
TITLE 'Descriptive Analysis of Sales Amount'; RUN;
```

Figure 12
Descriptive Analysis of Sales Amount

```
                         The UNIVARIATE Procedure
Moments
  N                   1175                Sum Weights                1175
  Mean                506.361775          Sum Observations     594975.085
  Std Deviation       300.130909          Variance             90078.5627
  Skewness            0.5096936           Kurtosis             2.03348034

Basic Statistical Measures
  Location                           Variability
  Mean               506.3618        Std Deviation         300.13091
  Median             492.2133        Variance              90079
  Mode               887.8604        Range                 2644

  Quantile             Estimate
  100% Max             2644.5888
  99%                   993.7520
  95%                   955.1337
  90%                   903.8239
  75% Q3                757.3536
  50% Median            492.2133
  25% Q1                255.2973
  10%                   112.2561
  5%                     56.2436
  1%                     12.1221
  0% Min                  0.2463

Extreme Observations
         Lowest                    Highest
  Value             Obs        Value          Obs
  0.2463            558      997.655          925
  2.4406              4      998.240           36
  2.6460            452      998.429          910
  5.2624            141     2382.013         1174
  6.3377            231     2644.589         1175
```

Figure 13
Identifying the Largest Sales

```
1 DATA Extreme; SET AccountsReceivable;
2 IF SalesAmount > 1000;
3 PROC PRINT; VAR SalesNumber SalesAmount;
4 TITLE 'Sales > 1000'; RUN;
```

Figure 12 contains key results of the Univariate procedure. The statistics describe the distribution of the variable in terms of central tendency, distribution about the mean, percentiles, and lowest and highest values.

If the user wants additional information about specific observations, such as the sales numbers for the largest sales amounts, a simple IF statement can filter those observations, as in Figure 13. SAS permits IF statements only in the DATA step. Thus, a DATA statement is needed to identify the dataset that will receive the filtered data, as in line 1.

Output of the program follows:

Obs	SalesNumber	SalesAmount
1	1587	$2,382.01
2	1960	$2,644.59

Figure 14
SAS Program to Compute Out of Bounds Values

```
DATA OutofBounds; Set AccountsReceivable;
IF CustomerID < 0 OR CustomerID > 100;
PROC PRINT DATA = OutofBounds;
VAR CustomerID SalesNumber SalesAmount CashReceiptsAmount;
TITLE 'CustomerIDs Out of Bounds'; RUN;
```

Figure 15
SAS Program to Compute Negative Accounts Receivable

```
DATA Negative; Set AccountsReceivable;
IF AccountsReceivable < 0;
PROC PRINT DATA = Negative;
VAR SalesNumber SalesAmount CashReceiptsAmount AccountsReceivable;
TITLE 'Negative Accounts Receivable'; RUN;
```

Figure 16
SAS Program to Identify Potential Data Errors

```
1 DATA Incorrect; Set AccountsReceivable;
2 IF AccountsReceivable < .01 THEN DELETE;
3 IF AccountsReceivable NE SalesAmount;
4 PROC PRINT DATA = Incorrect;
5 VAR SalesNumber SalesAmount CashReceiptsAmount AccountsReceivable;
6 TITLE 'Possible Incorrect Receipts'; RUN;
```

A similar procedure can be used to identify values that are outside of expected bounds. For example, Figure 14 provides a program to identify Customer IDs that are smaller or larger than those expected in the database.

The program identifies two observations with unexpected Customer IDs. The amounts of both observations are large and neither has been collected.

Obs	Customer ID	Sales Number	Sales Amount	Cash Receipts Amount
1	102	1587	$2,382.01	$0.00
2	102	1960	$2,644.59	$0.00

A similar program can be used to identify potential data entry errors, as in Figure 15. This program identifies accounts receivable with negative values.

Results of the program indicate a likely transposition error in recording the cash receipt:

Obs	Sales Number	Sales Amount	Cash Receipts Amount	Accounts Receivable
1	1418	$466.37	$646.37	$-180.00

Other errors of this type can be identified by comparing the accounts receivable and sales amounts. If the user expects customers to pay individual sales invoices in a single payment, amounts shown as partial payments may result from data entry errors as in Figure 16. Line 2 deletes accounts receivable with zero or negative values from the file before a comparison is made between receivables and sales amounts in line 3.

A likely data entry error is indicated by the output:

Obs	Sales Number	Sales Amount	Cash Receipts Amount	Accounts Receivable
1	2028	$755.75	$575.75	$180.00

Duplicate and Missing Observations

Identifying duplicate and missing records is relatively simple. Figure 17 provides a program to search for duplicate sales amounts that might indicate a sale was recorded more than once. Data are first sorted by sales amount, the field of primary interest. The LAG function in lines 3 and 4 identifies the value of the variable from the previous iteration of the DATA step. Consequently, if sales are sorted by amount and the amount of the previous record is equal to that of the current record, a potential duplication has occurred. Identifying the current and previous sales numbers and customer IDs can help in the evaluation.

The program identifies a sale with a likely duplication. The sales numbers are contiguous and the customer IDs are the same, in addition to the sales amounts being the same.

Obs	Sales Num -ber	Lag Sales Num -ber	Cust- omer ID	Lag Cust- omer ID	Sales Amount	Lag Sales Amount
1	1416	1415	93	93	$887.86	$887.86

A similar program can identify missing sales numbers. In Figure 18, the LAG function again is used to identify the value of the previous observation. In this case, the value of the previous observation is incremented by one in line 3 and then compared with the value of the current observation in line 4. Because the data are sorted by sales number, the current sales number should equal the previous number plus one.

The program identifies four missing sales numbers:

Obs	SalesNumber
1	1000
2	1159
3	1594
4	2092

Sampling

SAS can be used to select samples. As an example, assume an accountant wanted to select customers for a receivables confirmation using dollar unit sampling. The sample would be selected by sorting the data by customer, computing the total of accounts receivable, determining a random starting point, and selecting a stratified sample.

Figure 19 provides a program to sort the data and compute the total amount of accounts receivable.

The amount is computed to be $101,591.01. Depending on the power of the test desired, the accountant would select the number of records to be sampled. If the accountant wants 50 observations in the sample, the sample each sampling unit will be $1,992 greater than the previous unit ($101,591.01/(n+1)). A random starting point is then identified within the first sampling unit. Because only a portion of the first and last sampling units is used, the procedure requires n+1, rather than n sampling units.

Figure 20 contains a program to determine a random starting point. The UNIFORM function returns a random value that is equally distributed between 0 and 1. Multiplying by 1,992 results in a random value in the desired range.

The random value identifies the dollar unit that is the first observation. The customer associated with this dollar unit then becomes the first item in the sample. The next item will be the customer associated with the receivable dollar that is $1,992 greater than the first observation, and so forth. Figure 21 contains a program for identifying the sample, assuming the starting point determined in Figure 20 is $672.

Line 1 reads data from the output file created in Figure 8. This file contains the total amount of receivables for each customer. Line 2 computes a cumulative variable, TotalAR, that increases by the amount of the customer receiv-

Figure 17
SAS Program to Identify Duplicate Sales

```
PROC SORT DATA = AccountsReceivable; BY SalesAmount;
DATA Duplicate; Set AccountsReceivable;
LagSalesNumber = LAG(SalesNumber); LagCustomerID = LAG(CustomerID);
LagSalesAmount = LAG(SalesAmount); IF LagSalesAmount = SalesAmount;
PROC PRINT DATA = Duplicate;
VAR SalesNumber LagSalesNumber CustomerID LagCustomerID
   SalesAmount LagSalesAmount;
TITLE 'Duplicate Sales'; RUN;
```

Figure 18
SAS Program to Identify Missing Sales Numbers

```
PROC SORT DATA = AccountsReceivable; BY SalesNumber;
DATA Missing; Set AccountsReceivable;
LagSalesNumber = LAG(SalesNumber); NextSalesNumber = LagSalesNumber+1;
IF SalesNumber NE NextSalesNumber;
PROC PRINT DATA = Check; VAR SalesNumber;
TITLE 'Missing Sales Numbers'; RUN;
```

Figure 19
SAS Program to Compute Total Accounts Receivable

```
DATA Select; SET AccountsReceivable;
PROC SORT DATA = Select; BY CustomerID;
PROC MEANS SUM DATA = Select; VAR AccountsReceivable;
TITLE 'Total Accounts Receivable'; RUN;
```

Figure 20
SAS Program to Identify Starting Value for Dollar Unit Sample

```
DATA Select; Select = UNIFORM(0)*1992; PROC PRINT DATA = Select;
TITLE 'Starting Point for Dollar Unit Sample'; RUN;
```

Figure 21
SAS Program to Select Dollar Unit Sample

```
DATA TotalAR; SET SumAR;
TotalAR + SumAccountsReceivable;
Select = 672;
SampleItem = 'no '; IF TotalAR >= Select THEN SampleItem = 'yes';
IF TotalAR >= Select THEN TotalAR = TotalAR - 1992;
IF SampleItem = 'yes'; PROC PRINT DATA = Read;
VAR CustomerID SumAccountsReceivable;
TITLE 'Dollar Unit Sample'; RUN;
```

able each time a new customer record is read in line 1. Line 3 defines the starting point. Line 4 selects a record when the cumulative value of receivables is equal to or greater than the starting point. Line 5 decreases the cumulative value by the dollar value that is to be skipped before the next observation is selected. Thus, the cumulative value must increase by $1,992 before another record is selected for the sample. Lines 6-8 print the selected observations.

The program results in 50 observations of the following form:

```
                               Sum
                             Accounts
Obs        CustomerID       Receivable
1              1               $718.15
2              4             $1,617.74
3              6             $1,439.44
4              7               $751.79
5              8             $2,061.57
```

Customers with large receivables are likely to be included in the sample more than once, because each dollar of receivables has an equal chance of being selected. Sorting the sample by customer and selecting the sample at equal increments minimizes the number of duplicate observations in the sample.

Summary and Suggestions for Future Research

Data extraction and analysis can be complicated tasks. Tools like SAS can simplify these tasks. Though SAS is very powerful software and contains a large number of built-in procedures, accountants often find that a relatively small set of procedures and commands meets most of their needs. Once developed, SAS programs can be saved and recalled whenever they are needed. Minor modifications usually are sufficient to adapt existing programs for new purposes or data environments. SAS provides accountants with tremendous power for examining and understanding computerized data.

This paper describes advantages of using SAS for data extraction and analysis, and it provides a brief overview of the SAS program. Primarily, the paper provides an example exercise that explores many of the basic features of SAS as a data extraction and analysis tool.

SAS is an extremely powerful and flexible tool. There are many extensions to the tasks described in this paper that can be explored in more detail. Some of these include using SAS for analytical review, for detailed analysis of accounting data, and for use of accounting data in management decisions.

Research also could explore the relative effect of SAS and other tools on decision making. Examples of research questions that might be considered include: Are students, accountants, and managers who are capable of working with SAS and similar tools, more capable of understanding and using business data than other individuals? Are particular tools more useful for developing these skills than others, SAS versus Excel, for example. Other data analysis and extraction tools are available commercially, such as ACL®[4], Audit Command Language. Comparisons among these tools are another possibility for future research.

Endnotes

1. SAS is a registered trademark of the SAS Institute, Inc., Cary, NC.
2. A more detailed introduction to basic SAS commands is provided by Delwiche and Slaughter, 1998. An introduction to SAS statistical procedures is provided by Cody and Smith, 1997. The SAS Institute publishes a large variety of user manuals and provides workshops for user training.
3. The data are available from the author as an Excel file that can be imported into most standard relational database management systems.
4. ACL is a registered trademark of ACL Services Ltd., Vancouver, BC, Canada.

References

1. Cody, R. and J. Smith, *Applied Statistics and the SAS Programming Language*, 4th ed., Prentice-Hall, Upper Saddle River, NJ, 1997.
2. Delwiche, L. and S. Slaughter, *The Little SAS Book: A Primer*, 2nd ed., SAS Institute, Cary, NC, 1998.

Notes