

Faster Flat File Testing

John E. Bentley, Wachovia Bank, Charlotte NC

ABSTRACT

When doing data discovery for a new database project, we often have a flat file whose contents we know something about, such as the valid values for specific fields, but for one reason or another we want to look at only a sample of the data. When the file is small, using SAS® to pull the sample is easy and fast to do. But when the file has a record length of thousands of bytes, contains millions of records each with dozens of fields, and has over a dozen values for our sampling by-group, we can expect lengthy processing times if we use brute force to do the job. This tip will show one easy-to-code approach that dramatically reduces processing time.

THE PROBLEM

We received a flat file containing about 12 million records each with 142 variables and having a record length of 3198. The data was from fourteen different source systems and, based on past experience, the data analyst had reason to suspect that data provided by some source systems would be more complete than others. An easy way to examine the data, she thought, would be to pull a small sample records from each data source included in the file. The problem was that records from each source systems weren't grouped together.

We didn't want to create a SAS data set of the entire file and then sample that, and we didn't want to make multiple passes thru the file to first pull a sample and then create a lookup table to use. Because we know the source system codes, though, we quickly developed a solution.

THE SOLUTION

Here's what we did to simply pull the first twenty-five records for each source system code. The code is on the next page.

- ◆ Specify the sample size as a macro variable.
- ◆ In a DATA step use the RETAIN statement to initialize a counter for each source system code at zero.
- ◆ Use the INFILE and INPUT statements to read the data.
- ◆ After the INPUT statement reads a record, use a series of IF statements to check the value of the source system code and the number of records already selected for the sample.
 - If the sample size is less than the number wanted, then output the record and increment the counter.
 - If the sample for that source system code has been filled, read the next record.
- ◆ Use the STOP statement when all source system samples are filled so that the DATA step stops processing the remaining records.

OTHER APPLICATIONS FOR THIS SOLUTION

It is easy to modify this approach to pull different sized samples from the groups that you know exists in your data or to pull a random number generated sample. It all revolves around knowing the valid values that define the groups you want to sample and then using group-specific macro variables holding the sample size or sampling proportion.

If your relational database has limited sampling capabilities like Informix does, you can also apply this approach to a database table by using either the appropriate SAS libname engine or a in stored procedure.

AUTHOR BIOGRAPHY AND CONTACT INFORMATION

Since 1987 John Bentley has used SAS in the healthcare, insurance, and banking industries. He is currently an Assistant Vice President with Wachovia Bank's Corporate Data Management Group where he supports the Bank's data warehouse and data marts. John is a SAS Certified Advanced Programmer, has been a Section Chair at both SUGI and SESUG, and is the 2003-2004 President of the Charlotte Area Wachovia In-House SAS Users Group. His current professional interests are automating SAS programs and parallel computing. John has a Masters degree in Political Science with a Concentration in Southeast Asian Politics and is currently enrolled in a Masters of Information Systems program.

John E. Bentley
Corporate Data Management and Governance
Wachovia Bank
201 S. College Street, NC-1025
Charlotte NC 28210

CODE

```
** Assign the sample size to a macro variable. ;
%let _sampSize=25;

** Data step does all the work. Libname samples assigned earlier;
data samples.wireSample(drop= aqSamp bnSamp ixSamp j7Cnt);

    ** Specify counters holding number of records in each source system specific ;
    ** sample. Number of counters has been reduced to reduce the length of the ;
    ** code. ;
    retain aqSamp bnSamp j7Samp 0 ;

    infile "&_flat_file_location_and_filename" missover lrecl=3198;
        @1 eftSysCde      $char2.
        @3 bbkInfoSentIDC $char1.
        @4 ibkInfoSentIDC $char1.
        @5 obkInfoSentIDC $char1.
        @6 calrName       $char3.
        @9 processDte      $char10.
        @19 transactDte    $char10.
        /* lots and lots of other fields. */
        @3140 crpayRefNbr   $char24.
        @3164 bbkRefNbr     $char24.
        @3188 eftMsg        $char10.
    ;

    ** Put the sample size value into a numeric variable to stop warning of char ;
    ** to num conversion and to make the code more readable. ;
    size=input(put("&_sampSize",$2.),3.);

    ** Check the value of each source system code and the number of records in ;
    ** that sample. If its less than the max then output the record, increment ;
    ** the counter and read the next record. ;
    if eftSysCde='AQ' and aqSamp < size then do;
        output;
        aqSamp=aqSamp+1;
        return;
    end;
    if efsSysCde='BN' and bnSamp < size then do;
        output;
        bnSamp=bnSamp+1;
        return;
    end;
    if efsSysCde='J7' and j7Samp < size then do;
        output;
        j7Samp=j7Samp+1;
        return;
    end;

    ** If all samples have been filled, then stop. ;
    if aqSamp = size and bnSamp = size and j7Samp = size the stop;
run;

** Sort the sample data set ;
proc sort data=samples.wireSample;
    by eftSysCde;
run;
```

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.