

# Producing Multipurpose Metadata for Data Quality, Trending, and a Data Dictionary

John E. Bentley, First Union National Bank

## Abstract:

One of the benefits of a data warehouse is that it provides a "single version of the truth". But the "truth", unfortunately, is often difficult to understand, changes quickly, and is only as accurate as the information it is based on. Metadata is needed to assess quality of the data from which the truth is derived, and the same metadata can be incorporated into a data dictionary and used for trend analysis. This paper presents a SAS® application that generates metadata in the form of one-way frequencies (for categorical fields) and descriptive statistics (for numeric fields) from newly loaded data warehouse tables, compares current metadata to that produced in the previous load, and then reports changes and anomalies. It also produces a series of html files containing frequencies and descriptive statistics that can be linked to the data warehouse's data dictionary. SAS products used include BASE, the Macro Language, AF/FAME, SAS/CONNECT and Multi-Process Connect, and the Output Delivery System. The application was developed in for a Win98/UNIX environment but can easily be modified to work with any RDBMS or systems environment.

*Disclaimer: The views, opinions, and ideas expressed here are those of the author and not those of First Union National Bank. Readers should not infer from this paper that First Union National Bank actually uses the application that the author describes.*

## Metadata—Why?

An underlying, often-ignored truth is that a data warehouse is only as good as its metadata. A data warehouse does not generate real value until it is used to provide information that supports business decision-making, and metadata is critical for that. Metadata is needed for:

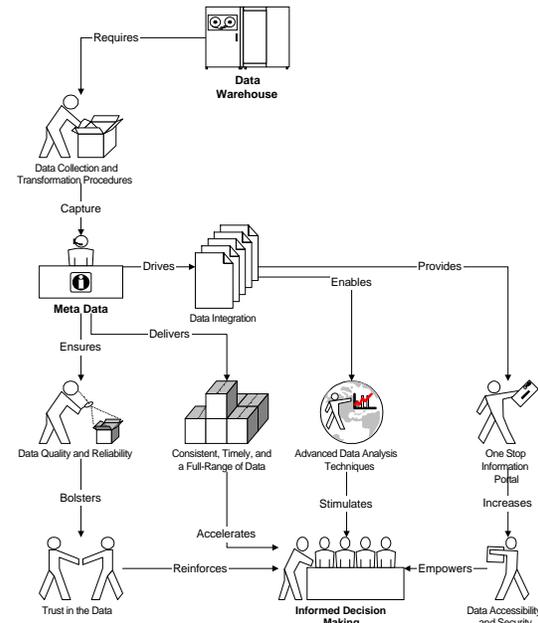
- Data Quality, because metadata tells what is supposed to be in the data warehouse and provides documentation for what is really there;
- Data Exploitation, because metadata tells business users, data analysts, and IT programmers where to find the exact data they need and helps them to understand what it means.

Put simply, good metadata makes it easier to use the data warehouse by allowing more accurate loads and faster turn-around for information requests. More accurate loads and faster request turn-around causes to higher usage and productivity, and higher usage and productivity usually equal a higher (and faster) return on investment (ROI).

Metadata provides measurable value to a data warehouse and increases the ROI. One industry-recognized expert on metadata has suggested formulas for calculating metadata ROI and estimates that good metadata can add 1% to a corporation's revenues. (Marco, 2000.) According to this expert, metadata contributes measurable value by:

- Identifying mistakes and problems with source IT systems and data
- Improving decision making accuracy
- Allowing sophisticated data quality processing
- Reducing new employee training costs
- Increasing user confidence in the data warehouse, which results in higher usage and productivity

Diagram 1: Metadata in the Decision-Making Process



Adapted from Fletcher and Pinner, "Navigating the Data Warehouse Paradox Zone."

Although metadata is widely acknowledged as critical for getting the most business value out of a data warehouse (in this paper the term "data warehouse" includes data marts), most managers actually do little more than talk about metadata. A Spring 2001 Data Warehousing Institute survey reported that although the number of organizations that have implemented a comprehensive metadata solution rose to 19 percent from 9 percent in 1988, about half the respondents had not started implementation. Still, that's an improvement from 1988 when the figure was 75 percent.

Part of the reason for this lack of action is false assumption that the ROI for a metadata management system is difficult to quantify in terms of increasing revenues or decreasing expenses while the costs in terms of people, software, and time are clear. (See Marco, 2000.) Data warehouse projects are usually on an aggressive project schedule, but collecting metadata is time consuming when documentation for the legacy systems that feed the warehouse is unorganized, sketchy, or simply unavailable.

Another reason is that the mindset and priorities of both the technical staff and the business sponsor are different. Most frequently, the business side has authority over the project—they control the budget—and so the technical group has to implement the project as directed. Because of up-front cost or time constraints, the business sponsor may minimize the metadata effort even over the advice of the technical staff. As a result, metadata is assigned a "do it later" priority, but later never quite comes.

At the same time that the business sponsors may be neglecting the business metadata, an aggressive project



The GUI uses radio button controls for specifying the database or for comparing databases, and a list box control presents the contents of each database. When comparing databases, the list box will display only the tables that the two databases have in common. A spin box control lets the user specify sampling percentage between 10 and 50 percent, though this will be overridden programmatically if the sample size is too large for efficient processing. For tables that require date selection criteria, a box to specify the "As-Of Date" will unhide itself. Finally, a spin box control is used to specify an acceptable range of differences—for example, 5 percent may be the maximum acceptable month-to-month change in the average loan application amount.

Desktop icon buttons control the processing. For the table selected, counts, percents and differences will be produced for character data and nominal and ordinal-level numeric data. Descriptive statistics will run for interval and ratio-level numeric data. If both sets of metadata are needed, then the "do everything" button can be selected. There's a separate button to run the comparison routine.

It's important that the application is as robust and dynamic as possible, and that requires using some advanced macro programming techniques. In the SCL behind the GUI, the user specifications are converted into macro variables that are used by the programs that the buttons kick off. If remote compute services are needed to extract and process the data from a RDBMS, then SAS/CONNECT connects to the remote host and %SYSLPUT passes local macro values to a macro variables there. Here's part of the INIT section of the MAIN.SCL:

#### Code Sample 1: SCL INIT Section

```
init:
<statements>

call symput('_warehouse',db.selectedItem);
call symput('_table',table.selectedItem);
call symput('_sampleProp',sample);
call symput('_loadPeriod',asOfDate);

%syslput _warehouse=&_warehouse;
%syslput _Table=&_Table;
%syslput _sampleProp=&_sampleProp;
%syslput _loadPeriod=&_loadPeriod;
%syslput _htmlLib=/prod/data/metadata/output;
%syslput _dataLib/prod/data/metadata/data;

<other statements>

rsubmit;
libname dataLib v8 "&_dataLib";
%put _user_;
endrsubmit;
return;
```

The only hard coding is the library specifications for the directories that receive the output HTML files and permanent SAS data sets. These could have been built into the GUI to capture dynamic locations, however.

#### Extracting the Sample From a Database

When extracting from a parallel database, processing is most efficiently handled by the database's own engine. This can be accomplished via either PROC SQL's explicit pass-through facility or the SAS/ACCESS LIBNAME Statement. Depending on the database being accessed and the query's complexity, one approach may perform better than the other. For example, a large extract from Informix XPS performs best when explicit SQL pass-through is used to extract from

the Informix table into a temporary memory table and then write the contents of the memory table into a SAS data set. With Oracle 8.1, it seems that the Oracle LIBNAME engine works best.

In our application, both explicit SQL pass-through and the Informix LIBNAME engine are used. An Informix libref is defined and PROC SQL is used to first count the number of records in the target table and then pull a list of the fields in the table from the syscolumns table. The record count is needed because we want to verify that the user-specified sampling percentage doesn't return too many records than needed to insure a normal distribution, such as more than 300,000. If it does, we can recalculate a more reasonable sample size.

The field name data set has only one observation with two long character string variables—lists of the field names with a space between each. One variable is the list of numeric fields and the other is a list of character fields. Querying the database table for a list of its contents frees us from depending on someone to tell us when a field is added or deleted. We need the variable list as two observations so that we can put them into a macro variables used in the SELECT statement. Later, we will recreate the variable lists as two data sets—one for character field names and one for the numeric—with only one variable holding the field names. The values in these data sets will be swapped into a %DO loop.

Assuming we want to run two extract queries, here's some of the code we might use:

#### Code Sample 2: Macro Variables Holding Field Names

```
proc sql;
create table &_table._tempCols as
select sc.colname
from rdbms.syscolumns as sc,
rdbms.systables as st
where st.tabname="&_table"
and st.tabid=sc.tabid;
quit;

data &_table._cols(keep=colName);

length colNames1 colNames2 $5000;
retain colNames1 colNames2;

if eof then do;
call symput
('colNames1',substr(trim(colNames1),3));
call symput
('colNames2',substr(trim(colNames2),3));
end;

set &_table._tempCols end=eof;

<more statements>

** Create two data sets—one with numeric fields ;
** (amount, balance, percent) the other with ;
** character fields. ;
if index(colName,'_amt') or index(colName,'_bal') or
index(colName,'pct')
then colNames1=trim(colNames1)||' '||trim(colName);
else
colNames2=trim(colNames2)||' '||trim(colName);
run;
```

Using MP Connect, it's important to remember that a remote session runs independently of the local session and other remote sessions. Shared libraries must be specified in each session, and macro variables created in a local session must be passed to the remote session. An easy, efficient way to pass macro variables is to write them out in the local session to a permanent SAS data set in a shared library and then use a DATA NULL step in the remote session to convert them back to macro variables. Here's what I mean.

### Code Sample 3: Passing Macro Values to a Remote Session

```
data mac "/prod/data/metadata/data/macVars.sas7bdat";
  table=symget('_table');
  sample=symget('_sampleSize');
  lib=symget('_remlib');
  cols=symget('_colNames1');
run;

options autosignon=yes sascmd='sas';

** Remotely run query 1;
rsubmit process=query1 wait=no;

data _null_;
  set "/prod/data/codebook/data/macVars.sas7bdat";
  call symput('_table',table);
  call symput('_colNames1',cols);
  call symput('_sampleSize',sample);
  call symput('_remlib',lib);
run;;

libname remLib v8 "&_remlib";

<remote PROC SQL extract numeric fields>

endrsubmit;
```

### Code Sample 4. Extracting a Sample and Merging Data Sets

```
** Locally run query 2;
proc sql;
  connect to informix (database="rubadubdub");
  execute (set isolation to dirty read) by informix;
  execute (select &_colNames2
          from &_table
          where date=&_maxDate
          into scratch s_holding_table2) by informix;
  execute (select *
          from &_sampleSize samples of s_holding_table2
          into scratch s_sample_table2) by informix;
  create table dataLib.sample2 as
  select *
  from connection to informix
  (select * from s_sample_table2);
  disconnect from informix;
quit;

waitfor _all_query1; rget query1; signoff query1;

data dataLib.&_table._sample
  (label="Sample of &_labelTable Table loaded in &_loadPeriod"
  genmax=2);
  merge dataLib.sample1 dataLib.sample2;
run;
```

In Code Sample 2, we created two macro variables holding the variables listing the field names—colNames1 and colNames2. In Code Sample 3 we pass colNames1 to the remote host and there we run PROC SQL to extract a sample of the numeric data into a permanent SAS data set written to the REMLIB library. Notice that the SAS data set name is hard coded and that we use the same directory defined as the data libref in the MAIN.SCL INIT section.

In the local session seen in Code Sample 4, we pull a sample containing only the character fields and put it in the DATALIB library. Because the REMLIB and DATALIB libraries point to the same directory, after the remote query runs and the session ends we can refer to the data sets written to the either library using DATALIB as the libref.

Using MP Connect, the query in Code Sample 4 ran at the same time as Code Sample 3's query. In both queries, we used Informix's sampling function to avoid landing the data in a SAS data set and then sampling it. The WAITFOR command in Code Sample 4 causes local processing to not proceed until the remote query in Code Sample 3 completes before continuing to the MERGE data step.

To create the combined sample data set, we do a one-to-one merge. Because we aren't subsetting the data, the reliability of our analyses is affected by the fact that the records providing the character field data aren't necessarily the same as those containing the numeric field data.

When we merge the data sets, we use the GENMAX data set option to take advantage of SAS's ability to create "generations" of the same data set. Using generations simplifies naming conventions later when we compare the current metadata to the previous metadata.

Generation data sets are historical copies of a SAS data set. The multiple copies represent different versions of the same data set, which is archived each time it is replaced. As a group, the copies are called a generational group. They have the same root member name but a different version number. In Code Sample 4 we specify a two generations be retained—the most recent and the previous version.

### Table-Level Metadata

In addition to creating metadata about the individual fields, we also capture metadata about the source table itself. Information such as number of records in the table, date of last modification, and number of fields are important.

After the sample data sets are merged, PROC CONTENTS creates two datasets each with a single variable; one data set contains the names of the numeric variables, the other has names of the character variables. We need these data sets later when we swap variable names into the VAR statement in PROCs FREQ and MEANS.

Immediately after the PROC CONTENTS runs, we generate a "header" file that describes the contents of the table and write it to both a permanent SAS data set. That data set is added to the table's generational group of header files. So that we can run time-series analyses over a rolling twelve-month period, we specify twelve data sets in this generational group. We also use ODS to write an html file that is shown in Output Sample 1.

### Code Sample 5. Preparation and Writing the Header File

```
ods listing close;
ods html

body="/prod/data/codebook/output/&_table._&_loadPeriod.
_header.html";

title "Metadata for &_warehouse &_labelTable Table
loaded in &_loadPeriod";

data dataLib.&_table._header
(label="Header, &_labelTable Table, &_loadPeriod"
genmax=12);
set describe;
file print;
x=-1;

numRecs=input(put(left(trim(symget('_numRecs'))),$12.),1
2.);
length charVars numVars $3 sampleProp newSampleProp
$6 dsn $35;

charVars=left(trim(symget('_charVars')));
numVars=left(trim(symget('_numVars')));
totVars=charVars+numVars;
sampleProp=left(trim("&_sampleProportion"))||' %';
newSampleProp=substr(left(trim("&_newSampleProportion
")),1,3)||' %';
maxLoadDate=left(trim(symget('_maxLoadDate')));

<statements>

if _N_=1 then
do;
put @5 " CODEBOOK FOR TABLE: " dsn "in " whse
/@5 " Codebook generated on: " prndate +x " , " prntime
// @15 " Data Set Description: " memlabel
/ @15 " Table last modified: " maxLoadDate
// @15 " Number of Records in the " dsn "table: "
numRecs
/ @15 " Number of Variables: " totVars
<more statements>
```

### Field-Level Metadata

MP Connect is used to speed up processing of the field-level metadata. We run PROC FREQ to get one-way frequencies and PROC SUMMARY (or MEANS) to get descriptive statistics. If, for example, our character data set has thirty-two fields, then we can have two CPUs at the same time calculating frequencies for 16 fields. If the total number of unique values is the same in each set of fields (admittedly unlikely), processing could take only about half the time than if we had a single CPU calculating frequencies for all thirty-two fields,.

As with the extract processing, we must make certain macro variable values available to the remote session. We replace the macro value data set we created for the extract phase because we now also have the number of records in the table (the population size) and the sample size and proportion may have changed.

To reduce the amount of code, two macro programs are used—DO\_CHAR and DO\_NUM. They are very similar, but the major difference is that the former runs PROC FREQ and the latter runs PROC SUMMARY. Both use MP Connect to run two sessions and write output files simultaneously. To make the application manageable, both

macros INCLUDE the program with the PROC and ODS statements. Macro variables drive the code. Here's what part of the code looks like to submit PROC FREQ:

### Code Sample 6. Macro Program to Make Metadata

```
%MACRO DO_CHAR(_locRem);
options autosignon=yes sascmd="sas";

** Remote Processing ;
%if &_locRem=1 %then
%do;
rsubmit process=rem1 wait=no;
options symbolgen mprint;

<statements to recreate the macro variables>

libname qc v8 "&_remlib";

** Read the sample data set into remote memory ;
sasfile qc.&_Table._sample load;

** Process the odd-numbered variables;
%do n=1 %to %eval(&_charVars-0) %by 2;
%include char_html_pages.sas / source2;
%end;

** Clean up and reset. ;
sasfile qc.&_Table._sample close;
title;
ods html close; ods listing;
endrsubmit;
%end;

** Local Processing ;
%if &_locRem=2 %then
%do;
sasfile qc.&_informixTable._sample load;
%do n=2 %to %eval(&_charVars-0) %by 2;
< statements >
%end;
%end;
%MEND DO_CHAR;
```

We'll get to the INCLUDE statement next, but there are a few things to note here. First, like we did with the extract, we define a remote libref that is the same as the one used by the local session, and we recreate our macro variables. Next, the SASFILE command opens the sample data set and holds it in memory. Because PROCs FREQ and SUMMARY run multiple times—once for each variable—opening the data set once instead of many times improves performance. Also important is the “clean up and reset” because we need to close the sample data set and reset the ODS specifications.

Depending on the number of records in the data set, the number of variables, and the number unique values in each variable, running a series of PROC FREQ can be time consuming. To speed processing, we use MP Connect. The DO\_CHAR macro in Code Sample 6 is called twice, with the \_LOCREM parameter indicating whether the macro should execute remotely or locally. Controlled by %DO loops that read in the variable names one at a time, the remote session processes the odd numbered variables and the local session handles the even numbered variables. The results of the PROCs are output to data sets and later written to html files.

The program that is INCLUDED by DO\_CHAR is shown in Code Sample 7. Under control of a %DO loop, it creates a

macro variable holding the name of the nth variable in the character variable names data set, runs PROC FREQ on that variable, and writes the output as a new data set to that variable's generational group,

#### Code Sample 7. PROC FREQ

```
data _null_; set char(firstobs=&n obs=&n);
  call symput('_charName',name);
  call symput('_htmlFreqs','&_Table'||'_||'
    left(trim(name))||'_Freqs'||'_loadPeriod');
run;

data prep; set qc.&_Table._sample(keep=&_charName);
  numInSample=symget('_numInSample');
  numRecs=symget('_numRecs');
  weightVar=round((numRecs/numInSample)*100,1);
  call symput ('_freqDSN','&_Table'||'_||'charVar&n");

proc freq data=prep;
  tables &_charname / noprint missing out=char&n;
  weight weightVar;
run;

data qc.&_freqDSN (label="&_labelTable.&_charname,
  weighted &_newSampleProp % Sample"
  genmax=12);
  set char&n;
  run;

<more statements>
```

An important point in Code Sample 7 is that we weight the sample data to better approximate the population. In DATA PREP, we calculate a weighting variable—weightVar—at the same time we create the data set that contains provides input for PROC FREQ. WeightVar's formula uses the number of records in the table that we captured as a macro value during the extract process and the actual sample size, which we might have recalculated.

The output data set is written to the variable's generational group so that we can do time-series analysis later. Because these data sets are narrow and mostly short, they take very little space. Although there are a lot of data sets, using a standard data set naming convention for the generational groups makes keeping track of them very easy.

ODS is used to write the output to HTML files is similar to that used to write the Table Header file. It's still part of the code that's %INCLUDEd and so is controlled by the %DO loop seen in Code Sample 6. In the first DATA \_NULL \_ statement in Code Sample 7 we assign a value to the macro variable \_htmlFreqs; it contains the output filename that is used in the ODS HTML BODY= statement seen in Code Sample 8. This value is reassigned each time the %DO loop executes so that a separate HTML file/page is written for each variable. A sample HTML page of frequency counts is at Output Sample 2.

#### Code Sample 8. Writing the Variable Values, Counts, and Percentages to an HTML File

```
ods html
  body="/prod/data/codebook/output/&_htmlFreqs..html";

title "Distribution of &_warehouse &_labelTable.&_charName
loaded in &_loadPeriod";

data _null_; set char&n end=eof;
  file print;

  dsn=upcase(left("&_Table"));
  prndate=put(today(),mmdyy10.);
  prntime=put(time(),hhmm6.);
  x=-1;

  if _n_=1 then do;
    set charDes;
    put / ' Table Name = ' dsn
      / ' Variable Name = ' name
      / ' Label = ' label $busname.
      / 'Character Field'
      / ' Maximum Length = ' maxlen
      // 'As of ' prndate +x ", " prntime
      / 'Number of Unique Values = ' nobs
      // ;
    if nobs > 25 then do;
      put / "MORE THAN 25 UNIQUE VALUES.
FREQUENCY COUNTS NOT PRINTED";
    end;
  end;

  if nobs <= 25 then do;
    move=maxlen+10;
    if _n_=1 then
      put @1 'Value' @move 'Count' @move+15'Percent'/;
    put @1 &_charname @move count @move+15 percent;
  end;

  if eof then
    put // "Based on a Weighted Sample of
&_newSampleProportion %."
      / "See the Header page for data set descriptors.";

  format nobs comma6. count comma12. percent 6.2
    &_charname $xfmt.;
run;
```

One major difference between this ODS code and that for the Table Header that here we use conditional execution (if \_n\_=1) to read and print the contents of the field description data set that we created earlier with PROC CONTENTS. So that the output isn't excessive, we've hard coded that if there are more than twenty-five unique values then a message will print instead of the counts and percentages. This could have made this into a user-specified parameter in the GUI. Whatever we choose, though, the data remain available in the variable's generational data set.

Not shown is the code for generating descriptive statistics metadata for the numeric variables. It is very similar to that seen in Code Samples 6 through 8 except that PROC SUMMARY is used. A macro that contains a %DO loop cycles through the numeric variable list, analyzes each one, and then writes separate a generational data set and an HTML file of the output.

Output for the numeric fields is shown at Output Sample 3 and includes

- minimum and maximum values
- mean
- median
- standard deviation
- number of records with a non-missing value
- number of records with a missing value
- the sum of the non-missing values

## Identifying Changes in the Metadata

Standard naming convention and generational groups organize the permanent data sets and make it easy to use either PROC COMPARE or a DATA step to find changes in the metadata. These changes include the addition or removal of a "valid value" in a character variable or a change to the average or median value of a numeric variable.

The GUI provides the ability to compare the current and previous month's data sets and identify differences between them. The user chooses the appropriate Database radio button and a list of tables available for comparison appears. After selecting a table, the "Compare Metadata" button runs the comparison program.

As seen below in Code Sample 9, a DATA step with a MERGE statement checks for the addition or removal of unique values. A DATA step was used here because of the output we wanted and its ease of formatting.

### Code Sample 9. Comparing Character Variables

```
%MACRO CHAR_CHANGES;
%do n=1 %to %eval(&_charVars-0);
  data _null_; set char(firstobs=&n obs=&n);
  call symput('_varName',name);
  call symput ('_dsn',
    "&_table"||'_'||"charVar&n"||'_Freqs');
run;

<more statements>

data badData;
merge dataLib.&_dsn
  (gennum=-1 in=old drop=percent
  rename=(count=oldCount))
dataLib.&_dsn (in=new drop=percent
  rename=(count=newCount))
by &_varName;
retain nDropped nAdded nChanged 0;

if newCount > . then do;
  diff=newCount-oldCount;
  pctDiff=round((diff/oldCount)*100,1);
end;

length status $12;
if old=1 and new=0 then do;
  status='dropped'; nDropped=nDropped+1;
  output;
end;
else if old=0 and new=1 then do;
  status='added'; nAdded=nAdded+1;
  output;
end;
else if old=1 and new=1 and
  abs(PctDiff) %eval(&_maxDiff-0) then do;
  status='> big change'; nChanged=nChanged+1;
  output;
end;
run;
```

As with other routines, a macro is used so that we can cycle through the character name data set. Because the variable name is the same in each of the comparison data sets, we can one at a time assign the name to a macro variable and use it as the BY value when we merge the data sets.

Only values with discrepancies are output. Newly added and removed values are output, of course. For each existing value the percentage difference between the current and previous counts is calculated; if the percentage difference is greater than that allowed, a value of "big difference" is assigned to the status and the observation is output.

Code Sample 9 continues in Code Sample 10 by printing the results of the comparison to an HTML file. The number of records in the 'badData' data set is checked and a %IF statement controls what is printed to the log. If the number or records is not zero, then at least one discrepancy was identified and PROC PRINT executes. If the number or records is zero, then no problems were found and a DATA \_NULL\_ executes. Output Samples 4 and 5 show the reports.

### Code Sample 10. Writing Comparison Results as HTML

```
data _null_;
  dsid=open("badData");
  call symput('_numObs',attrn(dsid,"nobs"));
  busName=put(%trim(%upcase("&_varName")),
    $busName.);
  call symput('_busName',busName);
run;

%if %eval(&_numObs-0) > 0 %then %do;
data _null_; set badData(firstobs=&_numObs
  obs=&_numObs);
  call symput('_nDropped',nDropped);
  call symput('_nAdded',nAdded);
  call symput('_nChanged',nChanged);
run;

proc sort data=badData;
  by descending status &_varName;
run;

proc print data=badData label;
  var status &_varName oldCount newCount diff;
  label &_varName='Value';
  title "Major Changes to %upcase(&_varName)";
  title2 "%left(&_busName)";
  title3 "Load Period = &_loadPeriod";
  title4;
  title5 "  Number of Values Dropped = &_nDropped";
  title6 "  Number of Values Added = &_nAdded";
  title7 "Number of Counts Changed > &_maxDiff %
    =&_nChanged";
  footnote "Data set name= &_dsn";
run;
%end;

%if %eval(&_numObs-0) = 0 %then %do;
data _null_;
  file print;
  put "No Major Changes to %upcase(&_varName)"
    / "%left(&_busName)"
    / "Load Period = &_loadPeriod"
    // "  Number of Values Dropped = 0"
    / "  Number of Values Added = 0"
    / "Number of Counts Changed > &_maxDiff % = 0"
    // "Data set name= &_dsn";
run;
%end;

%end;
%mend CHAR_CHANGES;
```

For the numeric variables, we use PROC COMPARE inside a %DO loop to find differences. PROC Compare is faster here because there is only one record in each data set and it will calculate both the difference and percentage difference that we need. In the same %DO loop we print the results of the comparison.

### Code Sample 11. Comparing Numeric Variables

```
%macro NUM_CHANGES;
%do n=1 %to %eval(&_numVars-0);
  ** Create the name of the variable ;
  data _null_; set num(firstobs=&n obs=&n);
  call symput('_varName',name);
  call symput ('_dsn',
    "&_table"||'|_'||"numVar&n"||'_stats');
  run;

  proc compare
  base=dataLib.&_dsn (gennum=-1)
  compare=dataLib.&_dsn
  noprint outstats=diffs;
  var nMissing minimum maximum mean median
  standardDeviation sum;
  run;

  data final;
  set diffs(where=( _type_='MEAN')
  retain problems 0;
  if abs(_dif_) < 1 then _dif_=0;
  if abs(_dif_) > 0 then problems=problems+1;
  if eof then
  do;
    call symput('_problems',problems);
  end;
  run;

  %if %eval(&_problems-0) > 0 %then
  %do;
  data final;
  set checkEm(where=( _type_='MEAN')
  rename=( _dif_=diff _pctdif=pctDiff));
  label statistic='Statistic'
  diff='Difference'
  pctDiff='Percentage Difference';
  run;

  proc print data=final label noobs;
  var statistic sigmaValu rmadValu diff pctDiff;
  format sigmaValu rmadValu diff pctDiff
  comma20.3;
  title "Possible Problem with &_varName,
  &_loadPeriod";
  run;
  %end;

  %if %eval(&_problems-0) = 0 %then
  %do;
  data _null_;
  file print;
  put " Data Load Validation,
  %trim(%upcase(&_dsn)) Table", &_loadPeriod"
  // "No Problems with
  %trim(%upcase(&_varName))"
  / "The number of records, sum of the values,
  average value, and"
  / "minimum and maximum values in the Risk
  Data Mart matched those"
  / "in the previous load
  run;
  %end;
  title; title2; footnote;
  %end;

%mend NUM_COMPARE;
```

The CHAR\_CHANGES and NUM\_CHANGES macros are called as shown below. Because the processes run quickly, MP Connect is not needed. It is important to remember that the same ODS output file remains open until it is either redefined with a new BODY= statement or closed.

### Code Sample 12. Calling the Comparison Macros

```
ods listing close;
ods html body="&_htmlLib./&_charDiffs..html" ;

%char_changes;

ods html body="&_htmlLib./&_numDiffs..html" ;

%num_changes;

ods html close;
ods listing;
```

### Future Directions and Planned Modifications

Code that will do trend analysis has yet to be implemented but is straightforward. As seen in Code Sample 7 we retain twelve generations of the frequency counts and descriptive statistics. The numeric data lends itself naturally to line and bar charts produced by SAS/Graph and quickly show seasonal changes and changes that result from quarter- or year-end. For the character fields, though, bar charts with more than a few values each month quickly become cluttered and grouping may have to be defined, but PROC FORMAT can easily handle that without changing the data. If the GUI were modified to capture more user specifications for merging variables into new data sets, then three-dimensional charts could be produced.

It was mentioned early that automating the process could be accomplished using AutoSys or perhaps SAS's messaging facility. Although this sort of automation is well understood and has been implemented for other processing, the volume of data to be processed here—thousands of variables—will require detailed planning and testing for automation to be done without degrading ETL performance.

The output HTML files are now manually moved to the Web server. The data QA process requires that the monthly loads be reconciled before the data is released to users, so the application does not now land the files directly to the Web server. The file names are such that once moved to the Web server, they replace the existing files after those files are copied to a historical directory. Eventually, a SAS program or UNIX script will handle this.

The code samples presented in this paper are not necessarily the most efficient possible and either have or will be improved. For example, Code Sample 4 lands the sample as a SAS data set and Code Sample 7 uses PROC FREQ to do produce counts. It would be much more efficient to combine the extract and analysis in the SQL to that we can take advantage of the database's parallel processing capabilities. The results sets then would contain only the one-way frequencies and the descriptive statistics. One question to be solved, though, is how to use a weighted sample in the query—the number of records in many tables and performance considerations precludes using all of them in the analysis.

The generational approach to managing the data sets results in a lot of data sets. A better approach might be to have one data set per field name and include a variable that give the load date. An idea not yet fully formed is to use the data

sets as input for a multi-dimensional database or an exploratory data mart available to data analysts.

## Summary

The value of a data warehouse doesn't come from having a lot of data in one place. The database by itself has only potential value no matter how big it is. Return on investment is generated when the data is converted to information and then used to make decisions that solve problems. Without comprehensive metadata, though, much of the data in a warehouse may never become information; at best, converting it to information will take much longer and even then decision-makers may have doubts about its accuracy.

Metadata does not necessarily have to be costly to compile and make available to users. The application presented here uses common SAS Software products—BASE and CONNECT—and well-understood techniques such as macro programming, conditional execution, and the Output Delivery System, to provide a robust solution for creating and disseminating important metadata.

This metadata application described in this paper produces metadata of value to the IT staff as well as business users. This application

- Documents the data used to populate the warehouse;
- Generates quality-control metrics;
- Provides early warning of significant data changes from load-to-load;
- Allows data warehouse users to more fully understand the contents of the database.

## References and Resources

Ekerson, Wayne W. (2000) "Ignore Meta Data Strategy at Your Peril." Application Development Trends, March.

Fletcher, Tom and Jeff Pinner (2000) "Navigating the Data Warehousing Paradox Zone." dmDirect, March 3. [www.dmreview.com](http://www.dmreview.com)

Garner, Cheryl (2000) "New V7 Client/Server Capabilities to Solve and Secure Your Distributed Processing Needs". SUGI25 Proceedings.

Garner, Cheryl (2000) "How to Use Version 7 Features to Optimize the Distributed Capabilities of SAS Software". SUGI25 Proceedings.

Marco, David (2000) Meta Data ROI 3-part series. DM Review, September, October, November.

Marco, David (2000) Building and Managing the Meta Data Repository: A Full Life-Cycle Guide. New York: Wiley.

SAS Institute, SAS Online Documentation CD, Version 8, February 2000.

Silva, Greg (2000). "Codebook: Taking Another Look At Your Data." SUGI25 Proceedings.

Stearns, Naoko S. and John R. Gerlach (1998). "A Comprehensive Codebook Generator." SUGI23 Proceedings.

Stevens, Vernee (2000). "SAS Metadata Architecture and Current Industry Metadata Trends." SUGI25 Proceedings.

## Acknowledgements

The Author acknowledges the contributions of Naoko S. Stearns and John R. Gerlach for their SUGI23 paper "A Comprehensive Codebook Generator" and Greg Silva for his SUGI25 paper "Codebook: Taking Another Look At Your Data." These two papers provided some early conceptual and design guidance. Developers who are thinking about any sort of codebook-generation or metadata-related application should read those papers.

## Author Contact Information

John E. Bentley 704-383-2686  
First Union National Bank [John.Bentley2@FirstUnion.Com](mailto:John.Bentley2@FirstUnion.Com)  
201 S. College Street  
Mailcode NC-1025  
Charlotte NC 28288



## About the Author

John Bentley has used SAS Software for fourteen years in the healthcare, insurance, and banking industries. For the past four years he has been with the Enterprise Information Group of First Union National Bank with responsibilities of supporting users of First Union's data warehouse and data marts and managing the development of SAS client-server applications to extract, manipulate, and present information from them. John regularly presents at national, regional, and local SAS User Group Conferences.

**Output Sample 1. The Data Set Metadata**

METADATA FOR TABLE: CS\_ACCT\_BASE in CDW  
Codebook generated on: 07/13/2001, 16:32  
Data Set Description: Sample of data Table Loaded May01  
Table last modified: May 2, 2001

Number of Records in the CS\_ACCT table: 1,860,562  
Number of Variables: 60  
Character Variables: 41  
Numeric Variables: 19

Requested Sample Proportion: 10 %  
Sample Proportion Used: 10 %  
Number of Records in Sample: 186,056

**Output Sample 2. Character Variable Metadata**

Table Name = CS\_ACCT\_BASE  
Variable Name = PPD\_INS\_REI ST\_CDE  
Label = Prepaid Insurance Reinstatement Code  
Character Field  
Maximum Length = 1  
Table last loaded on April 3, 2001  
Number of Unique Values = 2

Value	Count	Percent
I	54,525	6.47
P	320,841	38.08
R	30	0.00
Missing	467,050	55.43

Printed on 07/06/2001, 17:08  
Based on a Weighted Sample of 10 %.  
See the Header page for data set descriptors.

**Output Sample 3. Numeric Variable Metadata**

```

Table Name = CS_ACCT_BASE
Variable Name = IPD_AGG_UERD_AMT
Label = Interest Paid Aggregate Unearned Amount
Numeric Field
Table last loaded on April 3, 2001
As of 07/06/2001, 17:18

Number with a non-missing value = 842,446
Number with a missing value = 0
Sum of the values = 5,843,786.80
Minimum value = 0.00
Maximum value = 21,400.97
Median = 0.00
Average = 6.74
Standard Deviation = 205.45

Printed on 07/06/2001, 17:18
Based on a Weighted Sample of 10 %.
IMPORTANT: SAS truncates weight variables to the integer portion.
See the Header page for data set descriptors.

```

**Output Sample 4a . Comparison Reports for Character Variables—Possible Problem Found**

**CS\_STATIC Table, May 2001**

**Major Changes to CC\_REL**

**Number of Values Compared = 267**

**Number of Values dropped = 6**

**Number of Values added = 3**

**Number of Counts Changed > 5 % = 10**

status	Value	April Records	May Records	Percent Diff.
only April	ã0	49	.	
<snip>				
only May	A	.	864	
<snip>				
Big Difference	ERL	189,324	161,087	-14.91
<snip>				

**Output Sample 4b . Comparison Report for Character Variables—No Problems Found**

Data Load Validation, CS\_STATIC Table, May 2001

Major Changes to SIMPLE\_REG\_CDE

Number of Values Dropped = 0

Number of Values Added = 0

Number of Counts Changed > 5% = 0

**Output Sample 5a. Comparison Reports for Numeric Variables—Possible Problem Found**

***Possible Problem with CS\_STATIC.BLCY\_PYMT\_INTVL, May 2001***

<b>Statistic</b>	<b>April Value</b>	<b>May Value</b>	<b>Difference</b>	<b>Percentage Difference</b>
numRecs	1,643,524	1,538,937	--104,587	-6.362
minimum	1.000	1.000	0.000	0.000
maximum	4.000	4.000	0.000	0.000
mean	1.022	1.021	-0.001	-0.057
sum	2,389,253	2,371,058	-18,195	-0.760

***Difference is May minus April***

**Output Sample 5b. Comparison Reports for Numeric Variables—No Problem Found**

Data Load Validation, CS\_STATIC Table, May 2001

No Problems with ADV\_PFMT\_AMT

The number of records, sum of the values, average value, and minimum and maximum values match those in the April load.