Paper AD09

Generate a Metadata for Analysis

Maria-Alexandra Paladines,
Applications Systems Specialist
School District of Palm Beach County, West Palm Beach, Fl

PAPER ABSTRACT

If you are working in a big organization has been evolving in analyzing, designing and implementing a data warehouse, let SAS macros and SQL make that easier for you. This paper describes SAS macros that can be used to generate metadata for analysis purposes to help the database administrator (DBA) with the challenge of organizing and cleaning tables and fields.

The metadata repository gives the DBA the information about the library name, data set name, field name, type and value of selected elements in a SQL database. In the School District of Palm Beach County, our metadata includes more than 385,554 variables gathered from different tables.

This program generates a table containing the values of selected columns in all tables in a database in which that column occurs. For example, let us say that the column "YEAR" is used in many tables in a SQL database. When run, this program creates an output table, which gives the table name, data type, and values of "YEAR" for all tables in which the column "YEAR" appears.

INTRODUCTION

Metadata is “data about data”. Metadata helps organizations make analytic use of the data accumulated over many years. Metadata takes technical users closer to the data by giving them an understanding of their own data that they can use in different ways. Metadata is also important for the effective use of a data warehouse, especially when you are in the design phase for the data warehouse and the extract-transform-load (ETL) process.

This paper explains an approach developed for the School District to create a customize metadata solution to be used for the DBA of the data warehouse application. Although the paper states the approach using SAS, the methodology can be used in any other SQL-based system.

METADATA USERS - METADATA REPOSITORY

TECHNICAL USERS

The metadata was designed to be used by technical users (such as database administrators, designers, and programmers) who are concerned with the task of developing and maintaining the system. These users may need to determine which tables would be affected by changes to the database (for example, if a column in one or more tables is renamed or changed).

REPOSITORY

The School District uses Microsoft SQL Server to store its metadata, including the information about library names, data set names, field’s names, field types and values. The rights to create and update the metadata reside with the DBA. However, the metadata that is owned and managed by the DBA is also available to other team members.

PROGRAM STATEMENTS

My SAS solution includes the use of Global variables, Macros, Proc content statements, and the SYMPUT function. The routines create memory-simulated tables for each one of the fields in the final table, as you will see in the discussion below. The entire program list is at the end of the paper (See Appendix 1)

SECTION 1 - TABLE PREPARATION

The purpose of this section it is to load the table that I will use in the MACRO statement explained in Section 4.

1. Set library name and print options.

```
LIBNAME DB1 ODBC DSN=DPROD OWNER=dbo BCP=YES ;
OPTIONS nodate pageno=1 linesize=80 pagesize=60;
```

2. Generate a list of the data work file that contains all the information about the data in the library DB1.

```
PROC CONTENTS data=DB1._all_
                        memtype=data directory NODETAILS
                        out=list_data ;
RUN;
```

3. Assign new formats to the fields selected according to the requirement.

The following is the meaning of each field.

CMemname	Library Name concatenated with Table Name
Name	Variable Name
SMemname	Library Name concatenated with Table Name between quotes and trimming spaces
SName	Variable Name between quotes and trimming spaces
Type	Variable Type (1 for text variables or 2 for dates)
Value	Will be populated later (Section 4)
Date	Will be populated later (Section 4)

You need Smemname and SName in order to used them to create the Memory simulated arrays (see Section 2 for details). The variables 'Value' and 'Date' will be populated with data later in the process (see Section 3) .

```
DATA list  (keep= CMemname  Name  SMemname  SName  Type  Value  Date)  ;
      LENGTH CMemname $52  SName $30  SMemname $58  Value $10  Date 3;
      FORMAT Date DateTime22.3 ;
      SET list_data ;
      CMemname = Libname || "."|| Memname ;
      SMemname = """ || TRIM (CMemname)  || """;
      SName    = """ || TRIM (Name)  || """ ;
RUN;
```

4. Create a SQL table with records having selected columns ('DATADATE' 'DATA_DATE' or 'FY' in this example).

You have two options: (1) customize this procedure to select any records that you want from the list, in which case you can change the WHERE statement; or (2) skip this procedure to use all the records already selected (if you decide to do this you must include a small change in this program).

```
PROC SQL;
```

```
CREATE TABLE Table_All AS
   SELECT Distinct * FROM List
            WHERE  (UPPER (Name)  contains 'DATADATE' or
                        UPPER (Name)  contains 'DATA_DATE' or
                        UPPER (Name)  contains 'FY ');
QUIT;
```

5. We need to be careful with any possible duplicate records in the table, so all distinct records are saved in Table_Uniq. This is the table that will be used later in the macro process.

```
PROC SQL;
   CREATE TABLE Table_Uniq AS
      SELECT Distinct CMemname, Name, SMemname, SName,
                                          Type, Value, Date
            FROM Table_ALL ORDER BY CMemname ASC, Name ASC ;
QUIT;
```

SECTION 2 – MEMORY SIMULATED ARRAYS

6. Create a final table with no records, using a dummy value in the WHERE statement.

```
PROC SQL ;
     Create TABLE Final_Table  AS
            SELECT CMemname, Name, Type, Value, Date
            FROM Table_Uniq WHERE Value = 'X' ;
QUIT;
```

7. Create the memory-simulated arrays from the unique table.

The Arrays names are: Table, Type, Field, Stable, and Sfield. Each one has a sequential number assigned after the name so we can easily identify each element of the simulated arrays.

```
DATA Tmp_Work ;
            SET Table_Uniq ;
            CALL symput  ('TABLE' || left (_n_) , CMemname )  ;
            CALL symput  ('TYPE' || left (_n_)  , Type ) ;
            CALL symput  ('FIELD' || left (_n_) , Name)  ;
            CALL symput  ('STABLE' || left (_n_) , SMemname )  ;
            CALL symput  ('SFIELD' || left (_n_) , SName)  ;
RUN;
```

The number of elements depends of the number of rows in the unique table.
Each number identifies only attributes of the same row. For example:

Example of the Tmp_Work Table

CMemname	NAME	SMemname	SName	TYPE	Value	Date
DB1 .TX_ABSENCE_YEARLY	FY	'DB1 .TX_ABSENCE_YEARLY'	'FY'	2		
DB1 .VCENTER_FY1996	DATA_DATE	'DB1 .VCENTER_FY1996'	'DATA_DATE'	1		
DB1 .VDISCIPLINE_CODES_FY2002	FY	'DB1 .VDISCIPLINE_CODES_FY2002'	'FY'	2		
DB1 .VFAILNOTE	FY	'DB1 .VFAILNOTE'	'FY'	2		
DB1 .VFCAT_001	FY	'DB1 .VFCAT_001'	'FY'	2		
DB1 .VFCAT_APS1	Data_Date	'DB1 .VFCAT_APS1'	'Data_Date'	2		
DB1.VFCAT_BY_CLASS_FY2002	FY	'DB1 .VFCAT_BY_CLASS_FY2002'	'FY'	2		
DB1 .VFCAT_SSS_COHORT_CLASS_FY03	pst_sss_fy	'DB1 .VFCAT_SSS_COHORT_CLASS_FY03'	'pst_sss_fy'	2		
DB1 .VFLW_001	Data_Date	'DB1 .VFLW_001'	'Data_Date'	1		
DB1 .VIEW1	FY	'DB1 .VIEW1'	'FY'	2		
DB1 .VK3ROSTER_001_OLD	DATA_DATE	'DB1 .VK3ROSTER_001_OLD'	'DATA_DATE'	1		
DB1 .VK3ROSTER_EOY1999_001	DATA_DATE	'DB1 .VK3ROSTER_EOY1999_001'	'DATA_DATE'	1		
DB1 .VMATH_COURSES	DATA_DATE	'DB1 .VMATH_COURSES'	'DATA_DATE'	1		
DB1 .VMATH_FACULTY_FY02	FY	'DB1 .VMATH_FACULTY_FY02'	'FY'	2		
DB1 .VPSAT_with_STUDID	PSATFY	'DB1 .VPSAT_with_STUDID'	'PSATFY'	2		
DB1 .VQ_KR1_STUDENT_RRR	FY	'DB1 .VQ_KR1_STUDENT_RRR'	'FY'	2		
DB1 .VSAT_001	Data_Date	'DB1 .VSAT_001'	'Data_Date'	1		
DB1 .VSAT_WITH_STUDID	SATFY	'DB1 .VSAT_WITH_STUDID'	'SATFY'	2		
DB1 .VSIFMINI_003	FY	'DB1 .VSIFMINI_003'	'FY'	2		

← Row 1

TABLE1 = DB1 .TX_ABSENCE_YEARLY
TYPE1 = 2
FIELD1 = FY
STABLE1 = 'DB1 .TX_ABSENCE_YEARLY'
SFIELD1 = 'FY'

→ **First Row of Data**

TABLE2 = DB1.VCENTER_FY1996
TYPE2 = 2
FIELD2 = DATA_DATE
STABLE2 = 'DB1. VCENTER_FY1996'
SFIELD2 = 'DATA_DATE'

→ **Second Row of Data**

SECTION 3 – NUMBER OF RECORDS IN MEMORY MACRO

8. Obtain the number of rows in the table and convert this into a global variable called 'g_TotalRec'. This variable will be used in the macro statement in section 4.

```
PROC SQL;
        Create TABLE Num_Rec AS SELECT Count (CMemname)  AS TotalRec FROM Table_Uniq ;
QUIT;

DATA TotalRec ; SET Num_Rec ;
                CALLsymput ('g_TotalRec', TotalRec)  ;
 RUN;
```

SECTION 4 – USE OF MACRO 'CREATE'

9. The Macro 'Create' inserts records in 'Final_Table' with the value of the field specified. The variable 'g_TotalRec' tells the macro how many elements are in the array. The macro 'Create' will run from the first element to the last looking for all distinct values populated in the table for one particular variable. The 'i' variable indicates what element of the array the macro is working with.

The macro look like:

%MACRO *Create*;

```
        %do i=1 %to &g_TotalRec ;
                PROC SQL ;          /* Create a temp work table with distinct values */
                Data temp2 ;  /* Create temp2 work table with values */
                PROC SQL ;          /* Insert final records to Final Table */
        %end ;
        quit;
%MEND Create;
```

10. A SQL procedure creates a temporary worktable with all distinct values from for the fields stored in the array element.

```
        PROC SQL ;
                Create TABLE temp AS
                SELECT Distinct &&FIELD&i AS Value
                        FROM &&TABLE&i  ;
        QUIT;
```

The procedure uses the form of macro:

&&FIELD&i, when 'i' is equal 1 the macro is pointing to content of the content of FIELD number 1 element, and &&TABLE&i, when 'i' is equal 1 the macro is pointing to the content of the content of TABLE number 1 element.

In or example FIELD1 = FY so the procedure load all distinct values stored in the variable FY in the table DB1 .TX_ABSENCE_YEARLY

11. The data statement formats the value read from 'temp' table depending on the type. In the same way, the macro form &&Type&i is used to point to the corresponding value type associated with the variable.

```
        DATA temp2 ; LENGTH SValue $10 SDate 3;
                        FORMAT SDate DateTime22.3;
          SET temp ; if &&Type&i = 2 then SValue = Value ;
                                        else SDate  = Value ;
        RUN;
```

12. The last SQL procedure inserts into the Final_Table all the final data: Library and table name, Field name, Value and Date.

```
        PROC SQL ;
                Insert into Final_Table  ( CMemname, Name, Value, Date)
                        SELECT &&STABLE&i, &&SFIELD&i,   temp2.SValue, temp2.Sdate
                 FROM temp2 ;
        QUIT ;
```

SECTION 5 – SAVE THE METADATA TABLE (MACRO 'TABLE-LOAD')

12. We use the macro %Table-Load to save the table in SQL format. This is a general macro; it can be use in any SAS program that needs to drop or create tables, or delete and insert rows from and in the table. The parameters are:

p_CreateTableFlag is 'Y' if you want save records in a new table or 'N' if you want to insert records in the existing table;

p_DeleteCriteria stores a condition for deleting records from the table before adding the new records;

p_InputFileName is the input table name;

p_OutputTableName is the final table;

p_Libname contains the library name were the output file will reside.

The program calls the macro in the following way

```
%Table-Load (p_CreateTableFlag=&g_CreateTableFlag,
        p_DeleteCriteria=&g_Value,
                p_InputFileName=Final_Table,
              p_OutputTableName=TMETADATA,
              p_LibName=DB1) ;
```

MACRO 'CREATE'

The complete Macro 'Create' has the following structure:

```
 %MACRO Create;
        %do i=1 %to &g_TotalRec ;
                PROC SQL ;
                        Create TABLE temp AS
                SELECT Distinct &&FIELD&i AS Value
                                        FROM &&TABLE&i  ;
                QUIT;

                Data temp2 ;
        LENGTH SValue $10 SDate 3; FORMAT SDate DateTime22.3;
                        SET temp ; if &&Type&i = 2 then SValue = Value ;
           else SDate  = Value ;
                RUN;

                PROC SQL;
                        insert into Final_Table  ( CMemname, Name, Value, Date)
                                SELECT &&STABLE&i, &&SFIELD&i,
          temp2.SValue, temp2.Sdate
          FROM temp2 ;
                QUIT ;
        %end ;
        quit;
%MEND Create;
```

CONCLUSIONS

This SAS program is still used by the DBA in different ways for different projects. The approach used in this paper can be used in many other projects. The methodology could be followed by other programmers independently of the database the company is using.

CONTACT INFORMATION

Author Name Maria-Alexandra Paladines
Application Computer Specialist

Educational Data Warehouse Team Member
Developer Team Leader

Company School District of Palm Beach County
Research, Evaluation and Accountability Department
Address 3370 Forest Hill Blvd. Suite B-228
City West Palm Beach Fl 33409
Work Phone: (561) 357-7602
Fax:
(561) 963-3842
E-mail: apaladines@mail.palmbeach.k12.fl.us

Appendix 1

```
/*************************************************************************/
/*                                                                       */
/*       Author: Maria-Alexandra Paladines                               */
/*        Title: Applications Systems Specialist                         */
/*   Department: Research, Evaluation & Accountability                   */
/*         Date: September 3, 2002                                       */
/*   Prog. Name: TMETAD01 Generates Table for matadata analysis.         */
/*  Input Table: DB1. (All Tables)                                       */
/*  Output File: TMETADATA                                               */
/*************************************************************************/


OPTIONS SOURCE2;                                          /* Prints %INCLUDE
files to log. */
%INCLUDE 'S:\Prod\system\utilities\Startup Options 1.sas';
%INCLUDE 'S:\Prod\system\sasmacros\Table Load Macro.sas' ;

/******************/
/* Debugging options */
/******************/
%MacroDebugOff;
%ClearLog;
%ClearOutput;
%DeleteWorkFiles;

/* User Specified Parameters */
%LET g_CreateTableFlag = Y;      /* << i.e.: Y or N (do not use quotes) */
%LET g_Value = ' ';                        /* delete criteria */

************;
* Set input sources;
************;
LIBNAME DB1  ODBC DSN=DPROD OWNER=dbo BCP=YES ;  /* 'SAS-data-library'    */

OPTIONS nodate pageno=1 linesize=80 pagesize=60;

proc contents data=DB1._all_ memtype=data directory NODETAILS out=list_data ;
run;
```

```
DATA list (keep= CMemname Name SMemname SName Value ) ;
         LENGTH CMemname $42 SName $20 SMemname $48  Value $25;
         SET list_data ;
         CMemname = Libname || "." || Memname ;
         SMemname = "'" || TRIM(CMemname) || "'";
         SName    = "'" || TRIM(Name) || "'" ;
RUN;

PROC SQL;
     CREATE TABLE Table_All AS   /* You can change this part to select tables
and fields */
         SELECT Distinct * FROM List
                 WHERE CMemname NOT contains '_DT' and
                          (UPPER(Name) contains 'DATADATE' or UPPER(Name)
contains 'DATA_DATE');
QUIT;
PROC SQL;
     CREATE TABLE Table_Uniq AS
         SELECT Distinct CMemname, Name, SMemname, SName, Value
                 FROM Table_ALL ORDER BY CMemname ASC, Name ASC ;
QUIT;

proc SQL ;              /* Create final Table with no records I am using a
dummy value */
         Create TABLE Final_Table  AS
                SELECT CMemname, Name, Value
                FROM Table_Uniq WHERE Value = 'X' ;
QUIT;

DATA Tmp_Work ;
        SET Table_Uniq ;
        CALL symput('TABLE'||left(_n_), CMemname ) ;
        CALL symput('FIELD'||left(_n_), Name) ;
        CALL symput('STABLE'||left(_n_), SMemname ) ;
        CALL symput('SFIELD'||left(_n_), SName) ;
RUN;

/* Macro to create a table with the content of each Table in the field
specified. */
 %MACRO Create;
        %do i=1 %to &g_TotalRec ;
                proc SQL ;              /* Open the table and load the
records meet the criteria */
                Create TABLE temp AS
                SELECT Distinct &&FIELD&i AS Value
                                FROM &&TABLE&i  ;
        QUIT;

                Data temp2 ; LENGTH SValue $25;
                        SET temp ; SValue  = Value ;
                RUN;
```

```
                proc SQL ;              /* Insert records in the Final Table
*/
                        insert into Final_Table ( CMemname, Name, Value )
                                select &&STABLE&i, &&SFIELD&i, temp2.SValue
from temp2 ;
                QUIT ;
        %end ;
        quit;
%MEND Create;
proc SQL ;              /* Create final Table with no records */
         Create TABLE Num_Rec AS SELECT Count(CMemname) AS TotalRec FROM
Table_Uniq ;
QUIT;
DATA TotalRec ; SET Num_Rec ; CALL symput('g_TotalRec', TotalRec) ; RUN;
%Create;
********;
%TableLoad(p_CreateTableFlag=&g_CreateTableFlag,       /* Call TableLoad
Macro */
           p_DeleteCriteria=&g_Value,
           p_InputFileName=Final_Table,
           p_OutputTableName=TMETADATA_DATADATE,
           p_LibName=DB1,
           p_DSNName=DPROD);
```