Paper A10-2009

Using SAS® as an Archival Repository for DB2 under z/OS (or other DBMS)

Rich Morris, Progressive Insurance, Mayfield Village, Ohio

ABSTRACT

Your DBA says that the older data has to be removed from the data base in order to make room for new data, but the Legal Department says that you still need to be able to access the older data. What do you do?

Copying the data from DB2 to a SAS set is easy enough – but how do you keep track of where it went? And how do you get the data back again?

What happens when you need to combine data from the live data base with data from the archives?

This paper discusses the reasoning and code that went into building an in-house DBMS archival/retrieval process at Progressive Insurance.

This process makes extensive use of macros, and the general methodology is currently being used to archive data from at least five DB2 data warehouses (and one SQL Server database).

INTRODUCTION

This paper traces Progressive's development of a true archival process for DB2 using SAS, beginning with a brief history and structure of the first database, and how the requirement for archival developed. The code that was developed (or borrowed) to build this application is then discussed in detail. Aspects of processing that went well are noted, as are things that did not go so well.

THE HISTORY OF MDW / M20

The data warehouse in question is Progressive Insurance's Marketing Data Warehouse (MDW), The warehouse was officially renamed to MDW 2000, and given the acronym M20 when the warehouse was restructured in the late 1990's – but the users still refer it as MDW. This paper will refer to MDW; the code that is shown will reference the DB2 creator name, or M20.

MDW was originally intended to be used solely by the product Research and Development team for research, modeling, and analysis. This data warehouse represented the first time that data from multiple systems was consolidated into a single relational data store.

As new data elements were added, new uses for the data were developed, and the user community grew to include state analysts at the product, state, and regional levels. It also became a de facto source for various regulatory and statutory reporting mechanisms.

The original database requirements only specified that data for the four calendar years prior to the current year be retained in the live database. It was not anticipated that the data would ever be needed beyond the original requirements.

The requirement that we retain the older data sprang full-grown into existence the very first time that the data warehouse was used to supply data in response to a legal action. The problem with responding to an attorney on behalf of a group is that any member of the group needs to be able to duplicate your results.

Removing rows from the database was generally triggered by the DBA reporting that one of the tables was expected to become too large within the next six to twelve months. Removing older rows from the database appears to have occurred roughly every four years.

The process of removing data from the database was ad hoc and only semi-formal because the retention of old data was not originally anticipated. The jobs used in this process were run under the user ID of the programmer who was responsible for removing the rows, and run via a Production Control Work Order.

Each programmer who executed the process tended to use their own "standards" for dataset names because data removal was so infrequent. There was nothing in the dataset name to indicate any criticality regarding contents, or requirements for retention, and dataset names were not consistent across iterations.

The creating job name in the tape management catalog indicated that this was not a production job, but a user-submitted job. Production Control and Storage Management tend to regard datasets created by user-submitted jobs to be expendable – especially if the datasets are more than a year or two old.

The tables were backed up to tape using a DB2 utility function prior to purge processing. The purge process itself was (and still is) accomplished via unload/reload of rows to be retained. There was no separate repository for "purged" rows because these rows were never going to be accessed again.

The purged rows were only available as part of a DB2 utility backup. Therefore, retrieving the purged data required restoring the entire DB2 table, to a separate DB2 subsystem. A DBA would be needed to allocate the necessary disk space. The DBA staff would then contact Storage Administration, and Storage Administration would then need to contact the Capacity Planning group, and Capacity planning would co-ordinate with hardware management. The end result is that restoring a table becomes a multi-department project.

What if retrieving archived data could be made as simple as this:

```
//S1       EXEC SAS
//MYMAC    DD DISP=SHR,DSN=PNM20CA0.M20ARC.CNTL
//KEEPIT   DD DSN=TMVS727.KEEPIT.MYPROJ,
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=(SYSDA,8),SPACE=(CYL,(20,20))
//
//SYSIN DD *
 OPTIONS SASAUTOS=(MYMAC,SASAUTOS);
 %LET STATES='45','08';   /* USED BY ARCSETS */
 %LET YEARS=1997;         /* USED BY ARCSETS */
 %ARCSETS;
 %GETDATA(POLICY);        /* USES VALUES RETURNED BY ARCSETS */
 %GETDATA(VEHICLE);       /* USES VALUES RETURNED BY ARCSETS */
```

THE STRUCTURE OF MDW / M20

MDW was designed as policy-centric, where a 'policy' represents a unique contract, over a specific period of time, with a specific insured, and governed by the laws of the state in which the policy is issued.

State-specific laws determine the types of coverages required, minimum liability requirements, which driver or vehicle characteristics can be used in rating, etc. Even the rates themselves must be approved by that state's Department of Insurance (or similar regulatory body) before those rates can be charged. One result of this is that the majority of user-submitted queries tend to request data for one state at a time.

A given policy has one or more vehicles, and each vehicle has one or more coverages (e.g., bodily injury liability, physical damage liability, collision, etc.). There may also be discounts or surcharges that apply to one vehicle or to all vehicles on a policy. Each vehicle is assigned a rated driver, and that driver may have many violations, or none.

MDW was re-partitioned from 15 partitions to 34 partitions in the late 1990's in order to allow for larger amounts of data to be kept, as well as to improve performance. The re-partitioned version was called M20 – short for MDW 2000. The new partitions were defined such that the number of policies in each partition would be approximately equal. For example, there are five partitions that are devoted to policies written in Florida, while a single partition is shared by the policies written in Arkansas, New Jersey, and Tennessee.

DEFINING THE ARCHIVE PROCESS – THE OVERVIEW

The primary objective was strictly utilitarian – once data had been archived, could that data be retrieved for querying? (Without needing an act of Congress) And would the results of those queries be consistent with queries run against the data while it was still in DB2? (The answer to both questions is yes, but not covered here.)

The original unload/reload process specified that a policy and all of its related information would be eligible to be removed from the database if it was more than four years old. This information was reflected in a temporary DB2 table containing only the policy key fields for policies that were to be excluded from being reloaded, and was accessed by the unload/reload process for each partition.

The policy key fields are the issuance state code, the policy expiration year, policy number, and renewal suffix number. These are referred to collectively as the 'policy key'; all of the tables in MDW that are subject to archival processing contain the policy key.

The archival / retention criteria for the new process were revised to indicate that a policy would be retained on the database if there had been any <u>accounting</u> activity for it in the preceding four accounting years. Accounting activity can include refund of premium, payment on claims, etc.

In order to guarantee that the archival process and the unload / reload process were absolutely synchronized, we decided to use a single table named M20.ARCHIVE_POLS for both processes. This table would contain all of the policy keys for policies to be archived and it would be used to drive both processes.

Originally, there were 15 separate tables that were to be subject to archive processing – four of these tables have subsequently been 'retired'.

DEFINING THE ARCHIVE PROCESS – FLESHING OUT THE DETAILS

Simply copying rows from a DB2 table to a SAS table is easy enough – but what would an archival system need to do in order to be truly useful?

Having an archival process implies that you also have a retrieval process – if the data cannot be retrieved, then "archiving" is functionally equivalent to "deleting".

Given that we know that we will need to retrieve the data, or selected parts thereof, what are the kinds of questions that we can anticipate?

Imagine that the process has been in place for three iterations – if this is an annual process, then the first iteration ran almost three years ago. If a user requests data from the policy table from 5 years ago, for the state of Georgia, how can you locate that data?

You could restore the job run logs and start combing through those, looking for references to the table name. After you find all of the job logs that reference the table, you could compile a list of the dataset names containing the table. You could subset the list of dataset names further if you know which partition contains data for Georgia, and the partitioning has not changed. This should not take more than, say, two weeks.

You could have committed to memory the location of all of the data that has been archived. For MDW, this would be 34 partitions X 3 years X 11 tables, or 1122 separate entries. The problem with this approach is that <u>you</u> are now an integral part of the retrieval process.

You could remove yourself from the retrieval process by recording all of the relevant information in something like a Lotus Notes Database – but someone has to be responsible for making sure that the data is recorded.

On the other hand – computers are pretty good at record keeping, aren't they? And when the archival process was running, wouldn't the program have 'known' what table, partition, and state was being processed? What about the name of the SAS set receiving the data? Shouldn't something be known about the data that is being archived in terms of why or when this particular set of data is being archived?

What about the dataset name (DSN) containing the archived data? When the output archival set was closed, wouldn't

SAS have known how many records / rows / observations were written to the output dataset? Wouldn't it also know about all of the columns contained in each archival set?

In order to be able to get a particular subset of data quickly, such as data from the M20.POLICY table, partition 12, for archival year 2000, the retrieval system would need to know the following for each set of archival data:

1) name of the DB2 table that was processed
2) partition number
3) SAS set name that was created
4) Archival year (however that is defined)
5) DSN of the MVS dataset containing the SAS set

If we wanted to build additional intelligence into the process, we might also include:

6) Which state(s) were contained in which partition(s)
7) Number of observations
8) A list of column names, including the DB2 column descriptors
9) job name that created the archival set
10) date and time that the archival set was created

Now that we have some idea of what the specific objectives are, we can begin to look at putting together the necessary components.

DESIGNING THE COMPONENTS

Since we will be building a system, it's very likely that we will also end up supporting that system. It is in our best interest to a) keep the system as simple as possible, and b) do only as much work as is absolutely necessary.

MACRO GETPARM – PASSING THE OWNER AND TABLE NAME

Until now, we have been referring to DB2 tables using identifiers like "M20.POLICY". Strictly speaking, this identifier is the DB2 creator name concatenated with the DB2 table name. If we use the name of the DB2 table as the name of the SAS set, then we would only need to record one table name.

If we pass the string "M20.POLICY" to our process using SYSPARM on the EXEC statement, then the table name that is being processed will be immediately apparent to anyone who looks at the JCL (Job Control Language). We can parse SYSPARM to get the creator name and the table name, and store these in separate macro variables. Saving the DB2 creator name associated with the table will give us a reference to a unique DB2 table.

Fortunately, code that would perform a similar function already existed – so that code was used as a basis for the code in macro GETPARM.

This is macro GETPARM:

```
 /*********************************************************************/
 /* 05/26/2004 : SPLIT GETPARM INTO TWO COMPONENTS. GETPARM WIL NOW  */
 /*              *ONLY* PROCESS SYSPARM. THE PROCESSING OF DDNAME    */
 /*              PARTITON HAS BEEN MOVED TO MACRO GETPART.           */
 /*********************************************************************/
%MACRO  GETPARM;
        %GLOBAL OWNER TBNAME;
        %LET _STR1=%QSCAN(%QUOTE(&SYSPARM),1,%STR(,)); *04/23/02;
        %LET _STR2=%QSCAN(%QUOTE(&SYSPARM),2,%STR(,)); *04/23/02;
        /*******************************************************/
        /* SET TBNAME = WORD #2 OF &_STR1.                     */
        /*******************************************************/
        %*LET TBNAME=%SYSFUNC(SCAN(&SYSPARM,2));   *04/23/02;
        %LET TBNAME=%SYSFUNC(SCAN(&_STR1,2));      *04/23/02;
```

```
         /********************************************************/
         /* IF &TBNAME IS BLANK, THEN SYSPARM MUST BE ONLY 1     */
         /* WORD LONG. SET OWNER = M20, AND SET TBNAME = WORD #1.*/
         /********************************************************/
         %IF &TBNAME EQ %THEN %DO;
         *04/23/02;           %*LET TBNAME=%SYSFUNC(SCAN(&SYSPARM,1));
         *04/23/02;           %LET TBNAME=%SYSFUNC(SCAN(&_STR1,1));
                              %LET OWNER=M20;
                              %END;
                        %ELSE %DO;
         /********************************************************/
         /* TBNAME IS NOT BLANK, SO WORD #1 MUST BE THE OWNER    */
         /* NAME.                                                */
         /********************************************************/
                              %LET OWNER=%SYSFUNC(SCAN(&_STR1,1));
                              %END;
%MEND    GETPARM;
```

Note: The original version of this macro was set up to read only the table name – the DB2 creator name was assumed to be M20. There were additional database changes in 2002 that required the ability to process DB2 tables belonging to creators other than M20. This revised version is backward-compatible.

MACRO GETPART – PASSING THE PARTITION NUMBER

We want to be able to pass the partition as a symbolic parameter in the JCL (so that only one procedure is needed for each table, regardless of the number of partitions).

There was already an existing SAS-based process that was set up to process one partition of a table at a time – this process gets the partition number that is passed in via DDname PARTITON (DDname = Data Definition Name). DDname PARTITON points at a member of a parm library which contains the partition number (which corresponds to the member name). The member name is built using a symbolic parameter which is specified in the JCL.

This is macro GETPART:

```
/*******************************************************************/
 /* 05/26/2004 : GETPART - EXCISED FFROM GETPARM. THIS MACRO CONTAINS*/
 /*              THE CODE TO PICK UP PARTITION FROM DDNAME PARTITON. */
 /*******************************************************************/
%MACRO  GETPART;
        %GLOBAL PARTISHN;
        /******************************************************/
        /* GO READ DDNAME PARTITON AND PUT THE VALUE FOUND INTO */
        /* MACRO VARIABLE NAME PARTISHN.                      */
        /******************************************************/
          %LET RC=%SYSFUNC(FILEREF(PARTITON));
          %LET FID=%SYSFUNC(FOPEN(PARTITON,S));
          %PUT FID=&FID;
          %PUT RC(ALLOCATE)=&RC FID=&FID;
          %LET RC=%SYSFUNC(FREAD(&FID));
          %PUT RC(READ)=&RC;
          %*************************************************;
          %* NEXT LINE SETS MACRO VAR "PARTISHN". THE LEFT   *;
          %*  SIDE OF THE STMT IS A SYNTACTICAL REQUIREMENT. *;
          %*************************************************;
          %LET RC=%SYSFUNC(FGET(&FID,PARTISHN,2));
          %PUT RC(FGET)=&RC;
          %LET RC=%SYSFUNC(FCLOSE(&FID));
          %PUT RC(CLOSE)=&RC;
%MEND   GETPART;
```

MACRO TGT_ARC – JOINING A TARGET TABLE TO ARCHIVE_POLS

Back when we were defining the process and what we wanted it to be able to do, we stated that ARCHIVE_POLS would contain the policy keys of policies that had been selected for archive. We also noted that the policy key existed in all of the tables that are subject to archive processing.

Although this may be one of the most important requirements, it is one of the easiest ones to incorporate in terms of code.

All of the archival processing steps will require joining the target table to ARCHIVE_POLS. By consistently using correlation name ARC for ARCHIVE_POLS, and TGT as the correlation name for the target table, we can code the join criteria once and put it in a macro.

This is macro TGT_ARC:

```
%MACRO TGT_ARC;
      /*************************************************/
      /* JOIN TARGET AND ARCHIVE TABLES                */
      /*************************************************/

             TGT.PARTITION      = ARC.PARTITION
         AND TGT.RPT_BSNS_CD    = ARC.RPT_BSNS_CD
         AND TGT.ST_CD          = ARC.ST_CD
         AND TGT.POL_ID_CHAR    = ARC.POL_ID_CHAR
         AND TGT.RENW_SFX_NBR   = ARC.RENW_SFX_NBR
         AND TGT.POL_EXPR_YR    = ARC.POL_EXPR_YR
%MEND  TGT_ARC;
```

MACRO YRPARM - IDENTIFYING THE ARCHIVAL YEAR

The archival sets will be produced yearly - we would like to be able to determine which year is reflected in the data. We are going to use the term "archival year" to identify the data. This is the greatest policy expiration year that we would expect to find in the data, and should also be the most common.

"Archival year" was originally calculated by subtracting four from the current year. This worked until the accounting information was transferred to a new database named PLR (Premiums and Loss Reporting).

The new database used individual tables for various accounting periods, with an accounting period indicator embedded in the name. The 'active' tables were assembled into views which were then accessed by the users.

The intent was to separate data that was to be purged from the active data, so that the old data could be removed without impacting the users. This was to be done by:

1) Allocating a new DB2 table to hold data for the new accounting period.
2) Adding the new table to the view definition
3) Dropping the oldest table(s) from the view
4) Archiving the data from the oldest table(s)
5) Dropping the oldest table(s) from the database

During the development and implementation of this new structure, there was also a proposal to 1) increase the span of data kept in the databases, and 2) have different spans for different tables/views.This changed the archival strategy to one that would be driven by the contents of the database itself. The complexity of YRPARM increased dramatically.

This is macro YRPARM:

```
%MACRO  YRPARM;
  %GLOBAL YEAR;
  /*****************************************************************/
  /* 06/02/04 : WHOLSALE REPLACEMENT OF YRPARM. THIS VERSION INVOKES */
  /*            A QUERY AGAINST THE DB2 SYSTEM TABLES TO DETERMINE   */
  /*            WHICH "YEAR" IS MISSING FROM THE VIEW.               */
  /*****************************************************************/
```

```
              %USERQRY(FINDYEAR);
              PROC SQL NOPRINT;
                   SELECT COUNT(*) INTO :ROW_COUNT FROM FINDYEAR;
                   SELECT SUBSTR(NAME,11,4) INTO :YEAR
                   FROM FINDYEAR;
              RUN;
              %LET YEAR=%CMPRES(&YEAR);
      %MEND   YRPARM;
```

A local utility macro named USERQRY is used to define a view named FINDYEAR. This view is built by taking the SQL in DDname FINDYEAR, and wrapping it in the boiler-plate required by the pass-through facility.

This is the SQL contained in the dataset for DDname FINDYEAR:

```
SELECT NAME
FROM SYSIBM.SYSTABLES TBLS2
WHERE
      CREATOR='PLR'
 AND  TYPE='T'
 AND  NAME LIKE 'YRL_POL_PL%AU'

 AND  NOT EXISTS (
      SELECT '' FROM
           SYSIBM.SYSTABLES   TBLS
          ,SYSIBM.SYSVIEWDEP  VDEF
      WHERE
           VDEF.BCREATOR = 'PLR'
       AND VDEF.DNAME    = 'YRL_POL_PL_AU'
       AND TBLS.NAME   LIKE 'YRL_POL_PL%AU'
       AND TBLS.CREATOR  = VDEF.BCREATOR
       AND TBLS.NAME     = VDEF.BNAME
       AND TBLS2.NAME    = TBLS.NAME
                 )
```

The value returned by YRPARM is now dependent upon the contents of DDname FINDYEAR, which can be controlled through the JCL.

MACRO GETDSN – GET THE MVS DSN FOR A DDNAME

We will need to know the specific DSN (dataset name) of an archival set in order to allocate it. There was a pre-existing utility macro that will return that DSN associated with the DDname specified on the macro call.

This is macro GETDSN (with some comments removed)

```
%MACRO GETDSN(DDNM);
   /*******************************************************/
   /* 08/02/2007. TIGHTEN-UP THE MACRO CODE TO ONE LINE.  */
   /*******************************************************/
%GLOBAL MVSDSN;

   %LET MVSDSN=%SYSFUNC(PATHNAME(&DDNM));

%MEND  GETDSN;
```

MACRO SYSCOLS – GET DB2 COLUMN INFORMATION

One of the primary drawbacks to the old process was that there was so much data in a single chunk that it was virtually unmanageable.

Breaking this into 34 partition-level subsets certainly reduces the amount of data that needs to be handled at one time.

We can further reduce the size of the archival sets by shortening numeric variables where possible, or turning on compression when we know (or suspect) that the character variables contain mostly blanks.

We can answer both questions by using information obtained by querying SYSIBM.SYSCOLUMNS. If we want information on numeric columns, we would ask for all columns having a type of 'SMALLINT', 'INTEGER', 'DECIMAL', or 'DATE'. Information about character columns will be returned if we ask for all columns having a type of 'CHAR '.

Macro SYSCOLS was written to generate a SAS set containing the information from SYSCOLUMNS about either numeric or character columns for a particular table.

Macro SYSCOLS is not shown here, because of its size. (44 lines, excl. comments)

MACRO GENLNTHS – GENERATE A LENGTH STATEMENT FOR NUMERIC VARIABLES

A table that has a large number of date columns or one/zero indicators is going to increase in size when it is copied from DB2 to SAS because numeric variables in SAS have a default length of 8 bytes.

Columns containing one/zero indicators are generally defined in DB2 as type 'smallint' and these can be stored in SAS variables of length 3, saving 5 bytes per column. (This is true for all 'smallint' columns even if they are not one/zero indicators.)

Date columns are defined in DB2 as type 'date', and can be stored in SAS variables of length 4. (Note: We used 3 bytes to store dates for this particular archive process, which allows for a range of dates within 179 years of January 1, 1960. This was considered to be sufficient. Four bytes will allow many millennia.)

DB2 columns that are type 'integer' can be stored in 5 bytes.

DB2 columns of type 'DECIMAL' are much harder to classify, because the DB2 values for length and scale both impact the precision required to store the values accurately.

We finally resorted to brute-force processing to identify the most common combinations of length and precision that existed in the database. Progressively shorter lengths were tested for numeric variables by writing out observations for the highest 200 distinct values, and then reading that data back in and counting the number of distinct values in the dataset. A test that returned less than 200 distinct values would indicate that the variable length that was tested was too short.

The 22 combinations of length and scale that were found that could be stored in less than 8 bytes were stored in member TABLE. This is then used to update an array that contains the length required for the variable in SAS. The array is indexed using the DB2 values for length and precision. Figure 1 represents this array:

	Precision			
Length	0	1	2	3
3	3	3	3	3
4	3	3	3	8
5	4	4	4	8
6	4	4	4	8
7	4	4	5	8
8	8	8	8	8
9	5	5	5	8
10	8	8	8	8
11	6	6	6	8

Figure 1: SAS variable lengths for DB2 columns by Length and Precision

This is Macro GENLNTHS.

```
/*******************************************************************/
/* GENLNTHS : BUILDS A MACRO NAMED &MACNAME (DEFAULT IS SETLNTHS).  */
/*            GENLNTHS REQUIRES THE FOLOWING:                       */
/*            1) SAS SET COLUMNS EXISTS (BUILT BY SYSCOLS OR IT'S   */
/*               EQUIVILENT).                                       */
```

```
 /*            2) INFILE TABLE EXISTS, POINTED TO BY DDNAME TABLE    */
 /*               (MANUALLY BUILT, AND ALLOCATED IN THE JCL).        */
 /*******************************************************************/
%MACRO GENLNTHS(MACNAME);

     /*********************************************************/
     /* ASSIGN DEFAULT NAME IF NON SPECIFIED ON CALL.          */
     /*********************************************************/
       %IF &MACNAME EQ %THEN %LET MACNAME=SETLNTHS;

     /*********************************************************/
     /* NOW BUILD THE %MACRO _; AND %MEND _; STATEMENTS        */
     /* THESE WILL BE REFERENCED VIA "PUT" STATEMENTS LATER.   */
     /* THE NRSTR FUNCTION IS USED TO WRAP TICK MARKS AROUND   */
     /* THE %MACRO AND %MEND STATEMENTS SO THAT SAS KNOWS THAT */
     /* WE ARE REFERRING TO A LITERAL IN THE PUT STATEMENT.    */
     /* WITHOUT THE NRSTR FUNCTION, THE SAS PARASER WOULD GET  */
     /* TO THE "PUT" STATEMENT, SEE THE RESOLVED VALUE "%MACRO",*/
     /* AND THINK THAT WE WERE STARTING TO COMPILE A NEW MACRO. */
     /* WHICH, OF COURSE, IS NOT WHAT WE ARE TRYING TO DO.     */
     /*********************************************************/
       %LET _MACRO=%NRSTR(%%MACRO) &MACNAME %STR(;);
       %LET _MEND =%NRSTR(%%MEND) &MACNAME %STR(;);

OPTIONS MPRINT;
     /*********************************************************/
     /* ALLOCATE A TEMPORARY DATASET ON UNIT=VIO.              */
     /*********************************************************/
FILENAME TEMP '&&TEMP' UNIT=VIO SPACE=(TRK,(1,1));

     /*********************************************************/
     /* THIS MODULE BUILDS A MACRO NAMED (BY DEFAULT) SETLNTHS. */
     /* SETLNTHS (OR WHATEVER NAME IS REQUESTED) WILL BUILD A   */
     /* LENGTH STATEMENT THAT IS SPECIFIC TO THE DB2 TABLE THAT */
     /* WAS PROCESSED BY SYSCOLS.                               */
     /*********************************************************/

DATA ONE;
     /*********************************************************/
     /* WE NEED TO BUILD A MATRIX THAT CONTAINS THE DB2 VALUE   */
     /* FOR LENGTH, THE DB2 VALUE FOR SCALE, AND THE SAS VALUE  */
     /* FOR LENGTH. WE WILL THEN USE THE DB2 VALUES FOR LENGTH  */
     /* AND SCALE TO FIND THE APPROPRIATE LENGTH OF THE VARIABLE*/
     /* IN SAS. (WHY? BECAUSE SAS STORES ALL NUMERIC VARIABLES  */
     /* AS FLOATING-POINT NUMBERS. SEE THE SAS LANGUAGE         */
     /* REFERENCE : CONCEPTS, UNDER THE TOPIC "SAS VARIABLES:   */
     /* NUMERIC PRECISION" FOR A FULL DISCUSSION.)              */
     /*********************************************************/
     /* FIRST STEP IN THIS PROCESS IS TO POPULATE THE ARRAY WITH*/
     /* SAS' DEFAULT VALUES - I.E., 8.                          */
     /* *                                                       */
     /* NOTE THAT THIS IS A 9 X 4 MATRIX. THE ROW SUBSRIPTS ARE */
     /* 3 THROUGH 11, AND THE COLUMN SUBSCRIPTS ARE 0 THROUGH 3.*/
     /* *                                                       */
     /* THE ROW SUBSCRIPT WILL BE THE DB2 VARIABLE 'LENGTH'.    */
     /* *                                                       */
     /* THE COLUMN SUBSCRIPT WILL BE THE DB2 VARIABLE 'SCALE'.  */
     /* *                                                       */
     /* THE ARRAY IS NOT FULLY POPULATED BECAUSE I CANNOT FIGURE*/
     /* OUT AN ALGORITHM TO USE ON THE DECRIPTION OF DB2'S      */
     /* DECIMAL COLUMNS THAT GIVES ME A GOOD (CORRECT) ANSWER.  */
     /*********************************************************/
     ARRAY DB2DEC (3:11,0:3) L1-L36;
     RETAIN L1-L36 0;
     IF _N_=1 THEN DO;  /* _N_=1 */
     DO I = 3 TO 11;
        DO J = 0 TO 3;
     DB2DEC(I,J)=8;
   END;
END;
              END; /* _N_=1 */
RETAIN TFLAG 0;
IF TFLAG=0  THEN DO;
/************************************************************/
/* NOW READ THE TABLE OF 'PROVEN' VALUES, AND OVER-WRITE    */
/* SELECTED CELLS IN THE ARRAY. (THE 'PROVEN VALUES' WERE   */
/* GENERATED ONLY FOR THE MOST COMMON DESCRIPTIONS. I.E.,   */
/* LENGTH 11, SCALE 2 SHOWS UP ALL OVER THE PLACE. LENGTH   */
/* 11, SCALE 9 IS ABSENT FROM CRA.)                         */
/************************************************************/

                 DO UNTIL(LAST);
                    INFILE TABLE END=LAST;
                    INPUT LENGTH SCALE SASLNTH;
                    DB2DEC(LENGTH,SCALE)=SASLNTH;
                    IF LAST THEN OUTPUT;
                    DROP I J LENGTH SCALE SASLNTH;
                 END; /* UNTIL (LAST) */

                 TFLAG=1;

                 END; /* TFLAG */
/************************************************************/
/* NEXT, READ THE VALUES RETURNED BY SYSCOLS. SYSCOLS       */
/* BASICALLY DUMPS SYSIBM.SYSCOLUMNS FOR A PARTICULAR       */
/* TABLE, KEEPING ONLY ROWS FOR NUMERIC COLUMNS (IN THIS    */
/* USAGE).                                                  */
/************************************************************/
SET COLUMNS  END=ENDOFCOL ;
FILE TEMP;
RETAIN FLAG2 1;

/************************************************************/
/* ONCE AND ONLY ONCE WRITE THE MACRO STATEMENT.            */
/************************************************************/
IF FLAG2 THEN DO;
              PUT @1 "&_MACRO         ";
              PUT @1 "       LENGTH   ";
              FLAG2=0;
              END;

/************************************************************/
/* THE VALUES FOR COLTYPES DATE, INTEGER, AND SMALLINT ARE */
/* HARDCODED. "INTEGER" AND "SMALLINT" ARE "GIMMES", SINCE */
/* STORING AN INTEGER AS FLOATING-POINT EFFECTIVELY        */
/* REQUIRES ONE ADDITIONAL BYTE.                            */
/************************************************************/
/* DATE IS THE ONLY 'IFFY' VALUE. LENGTH 3 WILL ACCOMODATE */
/* DISCRETE DATE VALUES BETWEEN JULY 27, 1780, AND JUNE 6, */
/* 2139 - THIS SHOULD BE SUFFICIENT FOR INSURANCE-RELATED  */
/* DATA THAT IS IN USE AS OF THIS WRITING (04/16/2002).    */
/************************************************************/
IF      COLTYPE='DATE'     THEN NEWLNTH=3;
ELSE IF COLTYPE='SMALLINT' THEN NEWLNTH=3;
ELSE IF COLTYPE='INTEGER'  THEN NEWLNTH=5;
ELSE IF COLTYPE='DECIMAL'  THEN DO;

/************************************************************/
/* IF THE LENGTH AND SCALE ARE WITHIN THE BOUNDS OF THE    */
/* ARRAY, LOOK UP THE ARRAY ENTRY FOR THE DECIMAL COLUMN.  */
/* IF NOT WITHIN THE BOUNDS, SET NEWLENTH TO 8.            */
/************************************************************/
     IF     3 <= LENGTH <=11
       AND  0 <= SCALE  <= 3 THEN NEWLNTH=DB2DEC(LENGTH,SCALE);
```

```
                                             ELSE NEWLNTH=8;
                                                END;

     PUT  @8 NAME $25.  @35 NEWLNTH 3.;

     IF ENDOFCOL THEN DO;
     /***********************************************************/
     /* ALL DONE. WRITE THE MEND STATEMENT. (ONLY DO THIS ONCE.)*/
     /***********************************************************/
                      PUT @1 "      ;";
                      PUT @1 "&_MEND ";
                      END;
RUN;
OPTIONS SOURCE SOURCE2;
     /*************************************************************/
     /* NOW THAT WE'VE GENERATED A FILE CONTAINING THE DEFINITION */
     /* MACRO SETLNTHS, WE NEED TO TURN BACK AROUND AND INCLUDE IT*/
     /* SO THAT SAS KNOWS WHAT WE ARE TALKING ABOUT WHEN WE REFER */
     /* TO IT LATER.                                              */
     /*************************************************************/
%INCLUDE TEMP;

RUN;
%MEND  GENLNTHS;
```

MACRO SETCOMP – CONDITIONALLY TURN ON COMPRESSION

A table that has a large number of character variables may be a good candidate for SAS' dataset compression, if most of the data is blank.

After examining a large amount of data, we decided to turn on SAS' compression option for the archival set if the total length of all of the character variables was greater than 50, and the average length was at least 5. This was effectively a SWAG following a small amount of testing.

This is macro SETCOMP, with the comments removed.

```
%MACRO SETCOMP;
     %GLOBAL COMPRESS;
     PROC SQL NOPRINT;
               SELECT
                       SUM(LENGTH)
                     , AVG(LENGTH)
                INTO  :SUM,
                      :AVERAGE
               FROM COLUMNS
                              ;
     RUN;
     %PUT AVERAGE=&AVERAGE SUM=&SUM;
     %IF       %SYSEVALF(&AVERAGE GE 5)
           AND %SYSEVALF(&SUM  GE 50)    %THEN %DO;
                  %LET COMPRESS=YES;
                                            %END;
                                      %ELSE %DO;
                  %LET COMPRESS=NO;
                                            %END;
%MEND SETCOMP;
```

MACRO KEEPLOG – STORE INFORMATION ABOUT THE ARCHIVAL

The final component that we need is the one that will record what the archival process read, what it wrote, and where the output was written.

Build the DSN of the production archive log if this is a production job; otherwise build the DSN of the presumed test version.

Dynamically allocate the archive log using the DSN that was just built. Dynamic allocation allows us to get update access only when we need it – after the data for a table has been processed. This lets us run as many copies of the JCL simultaneously as the operating system will support.

M20.PARTITION maps the state code and last two digits of the policy number to a partition. It is not necessary to capture state-level information if we capture the data from M20.PARTITION. This data will only need to be once for each archival year that is processed.

The column information for any table during a single archival year will be consistent across partitions. This data also only needs to be captured once per archival year.

The last thing that we need to find out is how many observations were just written by inquiring against DICTIONARY.TABLES.

Now we can build a single observation that contains the following information:

1) name of the DB2 table (which is identical to the SAS set name),
2) the name of the DB2 owner
3) the partition number
4) the archival year
5) the DSN containing the archival set
6) the job name that created the data

After this observation is appended to the archive index, we can free the archive log so that other archival processes can record their information.

This is macro KEEPLOG:

```
%MACRO KEEPLOG;
%*******************************************************************;
%* 06/30/2004 : UPDATED TO CAPTURE THE NAME OF THE OWNER OF THE DB2 *;
%*              TABLE IN ARCHCOLS AS WELL AS ARCHIDX. (SEE BELOW).  *;
%*              ALSO CORRECT COMMENTS REGARDING ARCHIVAL YEAR. THE  *;
%*              NEW COMMENTS REFLECT THE USER OF MODULE FINDYEAR,   *;
%*              WHICH EFFECTIVELY RETURNS THE CURRENT YEAR MINUS 5  *;
%*              (WELL, SORT OF).                                    *;
%*******************************************************************;
%* 05/17/2004 : UPDATED TO CAPTURE THE NAME OF THE OWNER OF THE DB2 *;
%*              TABLE. THIS CHANGE SHOULD ALLOW US TO USE THE SAME  *;
%*              MODULE FOR PLRT, SLDW, OR M20 (OR ANY DB2 TABLE,    *;
%*              FOR THAT MATTER).                                   *;
%*              "OWNER" WAS ALSO USED AS A TEMPORARY MACRO VARIABLE *;
%*              NAME IN BUILDING THE DSN OF THE LOGGING FILE - THE  *;
%*              DSN IS NOW BUILT USING THE AUTOAMTIC MACRO VARIABLE *;
%*              "SYSUID".                                           *;
%*******************************************************************;
%* 04/17/2002 : KEEPLOG FOR M20 DATA ARCHIVAL PROCESS.              *;
%*                                                                  *;
%*              THIS MACRO UNCONDITIONALLY CREATES AN ENTRY IN      *;
%*              ARCHLOG.ARCHIDX. ARCHLOG.ARCHIDX CONTAINS THE DSN,  *;
%*              MEMNAME, JOBNAME, PARTITION ID, AND THE 'YEAR' THAT *;
%*              THE ARCHIVAL OCCURRED.                              *;
%*                                                                  *;
%*              POLICIES WITH EXPIRATION YEARS 1995 AND EARLIER     *;
%*              THAT WERE STILL IN M20 AND MET THE ARCHIVAL CRITERIA*;
%*              WERE ARCHIVED IN 2002.                              *;
%*                                                                  *;
%*              POLICIES WITH EXPIRATION YEAR 1996 WERE ALSO        *;
%*              ARCHIVED IN 2002.                                   *;
%*                                                                  *;
%*              DITTO FOR POLICIES WITH EXPIRATION YEAR 1997.       *;
%*                                                                 *;
%*               THE ARCHIVAL THAT WAS RUN IN 2003 ORIGINALLY      *;
%*               ARCHIVED DATA FROM 1998, BUT LABELLED AS 2003. THE *;
%*               DATA HAS BEEN CORRECTED, AND MODULE FINDYEAR (NEW) *;
%*               SHOULD NOW RETURN THE 'CORRECT' YEAR EVERY TIME.   *;
%*******************************************************************;
%* MODULE OVERVIEW :                                               *;
%*        1. FIND OUT WHO CALLED THIS MODULE, AND ALLOCATE THE     *;
%*           APPROPRIATE DATASET FOR LOGGING.                      *;
%*                                                                 *;
%*        2. GET THE SAS SET NAME AND PROCESS YEAR FOR THIS RUN,   *;
%*           (SAS SET NAME = DB2 TABLE NAME) AND FIND OUT IF WE    *;
%*           HAVE RECORDED WHAT THE DATASET LOOKS LIKE FROM THE SAS *;
%*           POINT OF VIEW. IF WE HAVE ALREADY RECORDED IT, DO     *;
%*           NOTHING. IF WE HAVE NOT RECORDED IT YET, DO SO NOW.   *;
%*                                                                 *;
%*        3. USE DICTIONARY.TABLES TO GET NUMBER OF OBS INTO MACRO *;
%*           VARIABLE NUMOBS, AND REFERENCE THAT IN THE CREATION   *;
%*           THE ARCHIVE INDEX RECORD (ONE ROW PER PARTITION PER   *;
%*           TABLE PER PROCESS-YEAR.).                             *;
%*                                                                 *;
%*        4. DYNAMICALLY DE-ALLOCATE THE LOGGING DATASET.          *;
%*                                                                 *;
%*               THE SECOND SET IS A ONE-OBS DATASET CONTAINING THE *;
%*               TABLENAME (FROM &TBNMAME), AND MVSDSN (FROM %GETDSN)*;
%*               AS WELL AS PARTITION AND YEAR.                    *;
%*                                                                 *;
%*               THE FIRST SET SHOULD TELL YOU EVERYTHING THAT YOU  *;
%*               NEED TO KNOW ABOUT ABOUT THE CONTENTS OF THE       *;
%*               ARCHIVED DATA IN TERMS OF WHAT VARIABLES ARE IN THE *;
%*               SET, WHAT THEIR FORMATS ARE, ETC.,.               *;
%*                                                                 *;
%*               THE SECOND SET IS INTENDED TO GIVE YOU ENOUGH      *;
%*               INFORMATION TO BE ABLE TO DETERMINE WHICH DATASETS *;
%*               YOU NEED TO INQUIRE ABOUT.                        *;
%*                                                                 *;
%*               BETWEEN THE TWO, YOU SHOULD BE ABLE TO IDENTIFY    *;
%*               DATASETS THAT NEED TO BE RECALLED USING HSM, AS    *;
%*               OPPOSED TO RESTORING THE DATA AND -THEN- FIGURING  *;
%*               OUT WHETHER OR NOT IT IS THE DATA THAT YOU NEED.   *;
%*******************************************************************;
%*               NOTE THAT A RELATIVE GDG MUST BE SPECIFIED AS EITHER*;
%*               '(0)' OR (-N), WHERE N CAN ALSO BE ZERO. IF YOU    *;
%*               SPECIFY '(+0)', SAS THINKS THAT YOU ARE TRYING TO  *;
%*               ALLOCATE A NEW GDG. IT THEN FAILS THE LIBNAME      *;
%*               STATEMENT AND SETS CC=0008.                 RAM   *;
%*******************************************************************;
   %GLOBAL PARTISHN YEAR;
%*******************************************************************;
%* THE FOLLOWING CODE BLOCK CRIBBED FROM THE CAPP VERSION OF KEEPLOG*;
%*******************************************************************;
%* THIS  CODE CHECKS THE JOBNAME AND SUBMITTING USERID TO DETERMINE *;
%* THE JOB STATUS. IF THE JOBNAME STARTS WITH A "P" AND WAS        *;
%* SUBMITTED BY CA7, ALLOCATE THE PRODUCTION COPY OF THE ARHCIVAL  *;
%* LOG. OTHERIWSE, ALLOCATE A DATASET UNDER THE ID OF THE USER THAT *;
%* SUBMITTED THE JOB.                                              *;
%*******************************************************************;
%IF     %SUBSTR(&SYSJOBID,1,1) = P
          AND ( &SYSUID         = PRODCA7
             OR &SYSUID         = WO7ONL )
                                  %THEN %DO;
               %LET ARCHLOG='PNM20CA0.ARCHLOG.RUNSTATS';
                                  %END;
                            %ELSE %DO;
                              %*LET OWNER=%SUBSTR(&SYSJOBID,1,7);
               %LET ARCHLOG="&SYSUID..ARCHLOG.RUNSTATS";
                                  %END;
```

```
%*****************************************************************;
%* DYNAMICALLY ALLOCATE THE ARCHIVAL LOG. IF UNAVAILABLE, KEEP      *;
%* TRYING PERIODICALLY FOR AT LEAST 5 MINUTES.                      *;
%*****************************************************************;
OPTIONS FILEMSGS; RUN;
   LIBNAME ARCHLOG &ARCHLOG DISP=OLD WAIT=5;

%*****************************************************************;
%* WRAP THE TABLE NAME IN QUOTES SO WE CAN USE IT IN PROC SQL.      *;
%*****************************************************************;
%LET MEMNAME=%UNQUOTE(%STR(%'&TBNAME%'));

%*****************************************************************;
%* IF THE OWNER IS M20, FIND OUT WHETHER OR NOT THE TABLE NAMED     *;
%* 'PARTIION' HAS BEEN PROCESSED YET.                               *;
%*****************************************************************;
%PUT OWNER=&OWNER IN KEEPLOG;
%IF &OWNER=M20 %THEN %DO;
    PROC SQL NOPRINT;
         SELECT COUNT(*) INTO :CVAR
         FROM ARCHLOG.PARTITION
         WHERE YEAR=&YEAR
         ;
    RUN;  /* FORCE RUN BOUNDARY SO THAT WE CAN CHECK &CVAR */
    %IF &CVAR = 0 %THEN %DO;

                        %DB2SSID;
                        LIBNAME M20 DB2 SSID=&DB2SSID AUTHID=M20;

                        PROC SQL;
                        CREATE VIEW MYVU AS
                        SELECT *
                        FROM CONNECTION TO DB2
                                (
                           SELECT * FROM M20.PARTITION
                                );
                        DATA PARTVU;
                        LENGTH
                             PARTITION
                             STRT_NBR
                             END_NBR
                             JOB_NBR
                             YEAR
                             4;
                             SET MYVU;
                             LENGTH PART $ 2
                                    OWNER $ 8;
                             PART=PUT(PARTITION,Z2.);
                             OWNER="&OWNER";
                             YEAR=&YEAR;
                        PROC SORT DATA=PARTVU OUT=PARTITION;
                             BY YEAR ST_CD;
                        PROC APPEND
                             BASE=ARCHLOG.PARTITION
                             DATA=PARTITION
                             ;

                       %END;
                      %END;
%*****************************************************************;
%* NOW INQUIRE AGAINST THE ARCHIVE INDEX TO FIND OUT WHETHER OR NOT *;
%* WE HAVE ALREADY CAPTURED THE INFORMATION ABOUT THIS TABLE FOR    *;
%* THIS ARCHIVAL YEAR.                                              *;
%*****************************************************************;
PROC SQL NOPRINT;
     SELECT
```

```
              COUNT(*) INTO :CVAR
        FROM
              ARCHLOG.ARCHCOLS
        WHERE
              MEMNAME = &MEMNAME
           AND YEAR    = &YEAR
        ;
        RUN; /* FORCE PROC BOUNDARY SO THAT WE CAN CHECK &CVAR */

   %IF %EVAL(&CVAR) = 0 %THEN %DO;   /* ARCHCOLS DOES NOT YET EXIST */

        /**************************************************/
        /* CREATE TABLE COLINFO TO BE APPENDED TO          */
        /* ARCHLOG.ARCHCOLS .                              */
        /**************************************************/
   PROC SQL;
        CREATE TABLE COLINFO AS (
        SELECT
              *
              ,&YEAR        AS YEAR
              ,"&OWNER"     AS OWNER LENGTH 8
        FROM
              DICTIONARY.COLUMNS
        WHERE
              LIBNAME = 'ARCHIVE'
           AND MEMNAME = &MEMNAME
                                 )
        ;
   %*******************************************************;
   %* APPEND COLINFO TO ARCHLOG.ARCHCOLS.                  *;
   %*******************************************************;
   PROC APPEND DATA=COLINFO
              BASE=ARCHLOG.ARCHCOLS;

                    %END;  /* ARCHCOLS DOES NOT YET EXIST */

        /**************************************************/
        /* GET THE NUMBER OF OBS FROM DICTIONARY.TABLES.   */
        /* (DICTIONARY.TABLES IS CREATED AUTOMAGICALLY BY  */
        /* SAS).                                           */
        /**************************************************/
   PROC SQL NOPRINT;
        SELECT NOBS INTO :NUMOBS
        FROM
              DICTIONARY.TABLES
        WHERE
              LIBNAME = 'ARCHIVE'
           AND MEMNAME = &MEMNAME
        ;


   DATA ARCHIDX;
        LENGTH
              PART    $  2
              YEAR       3
              DSN     $ 44
              OWNER   $  8
              MEMNAME $ 18
              JOBNAME $  8
              ;

        YEAR=&YEAR;
        NOBS=&NUMOBS;

        PART="&PARTISHN";
        OWNER="&OWNER";
        DSN="&MVSDSN";
```

```
     MEMNAME="&TBNAME";
     JOBNAME="&SYSJOBID";
     LABEL JOBNAME='CREATED BY*JOB NAME';

PROC APPEND DATA=ARCHIDX
            BASE=ARCHLOG.ARCHIDX;
        RUN;
%*******************************************************************;
%* DYNAMICALLY DE-ALLOCATE THE ARCHIVAL LOG. IF WE DID NOT DO THIS, *;
%* WE WOULD HAVE TO SINGLE-THREAD ALL OF THE STEPS IN THE ARCHIVAL  *;
%* PROCESS.                                                         *;
%*******************************************************************;
   LIBNAME ARCHLOG CLEAR;
   RUN;
OPTIONS NOFILEMSGS; RUN;
%MEND  KEEPLOG;
```

ASSEMBLING THE COMPONENTS

We now have all of the parts that we need to build an archival step.

The JCL that we need to archive the first table will look like this:

```
//************************************************************
//* M21CA010 : ARCHIVE OF ROWS FROM M20.POLICY               *
//************************************************************
//M21CA010 EXEC SAS,OPTIONS='SYSPARM=''''M20.POLICY'''''
//ARCHIVE  DD DSN=&ARCDSN.,
//            UNIT=(SYSPERM,20),SPACE=(CYL,(200,100)),
//            MGMTCLAS=MCMDW,
//            DISP=(NEW,CATLG,DELETE)
//PARTITON DD DSN=SNMVSR00.PARMLIB(CPPC01&PART.),DISP=SHR
//MACLIB   DD DISP=SHR,DSN=PNM20CA0.M20ARC.CNTL
//TABLE    DD DISP=SHR,DSN=PNM20CA0.M20ARC.CNTL(TABLE)
//FINDYEAR DD DISP=SHR,DSN=PNM20CA0.M20ARC.CNTL2(M22CA032)
//RUNDATE  DD DISP=SHR,DSN=PNM20CA0.M20ARC.CNTL(ARCDATE)
//SYSIN    DD DISP=SHR,DSN=PNM20CA0.M20ARC.CNTL(M21CA000)
```

Note: This example assumes that ARCDSN was set earlier in the job stream. The JCL required to archive a second DB2 table to the same dataset used above would require changing the step name (because you should), the table name in SYSPARM, and the DISP parameter for ARCHIVE.

It is important to note that DDname ARCHIVE is allocated on disk so that we can archive multiple DB2 tables to a single MVS dataset. Also note that DDname archive has its HSM management class – this management class is migrated to DFHSM level 1 after two days on non-use. This management class was set up after SYSPERM ran low on space because of the number of archival sets that were on disk.

This is the SAS code in member M21CA000:

```
*********************************************************************;
* M21CA000 : THIS IS THE MASTER MODULE FOR THE                      *;
*            M20 DATA-ARCHIVAL-TO-SAS PROCESSING.                   *;
*            THIS MODULE IS COMMON TO ALL TABLES                    *;
*            AND ALL PARTITIONS.                                    *;
*********************************************************************;
* 05/27/2004 : ADD INVOCATION OF GETPART FOLLOWING INVOCATION OF    *;
*              GETPARM. (GETPARM WAS SPLIT INTO TWO SEPARATE MODULES)*;
*********************************************************************;
OPTIONS SASAUTOS=(MACLIB,SASAUTOS);
***********************************************;
* GET THE MVS DATASET NAME THAT WILL HOLD THE *;
*     ARCHIVE FILE.                           *;
```

```
**************************************************;
%GETDSN(ARCHIVE);
**************************************************;
* GET THE VALUES PASSED IN VIA SYSPARM= ON THE   *;
*     EXEC STATEMENT.                            *;
**************************************************;
%GETPARM;
**************************************************;
* GET THE PARTITION THAT IS BEING PROCESSED.     *;
*     (THIS IS PASSED VIA DDNAME PARTITON).      *;
**************************************************;
%GETPART;                      /* 05/27/2004 RAM */
**************************************************;
* FIND OUT WHAT "YEAR" WE ARE PROCESSING IN/FOR. *;
**************************************************;
%YRPARM;
**************************************************;
* RUN SYSCOLS TO FIND OUT ABOUT THE NUMERIC COLS *;
*     IN DB2.                                    *;
**************************************************;
%SYSCOLS(TBNAME=&TBNAME);
**************************************************;
* DECREASE LENGTHS FOR CORRESPONDING SAS VARS    *;
* BASED ON DB2 COLUMN CHARACTERISTICS.           *;
**************************************************;
%GENLNTHS(LNUM);
**************************************************;
* RUN SYSCOLS AGAIN TO FIND OUT ABOUT THE        *;
*     CHARACTER VARIABLES.                       *;
**************************************************;
%SYSCOLS(TBNAME=&TBNAME,TYPE=CHAR);
**************************************************;
* ANALYZE DATA ABOUT THE CHARACTER VARIABLES AND *;
*     FIGURE OUT WHETHER TO TURN COMPRESSION ON  *;
*     OR OFF.                                    *;
**************************************************;
%SETCOMP;
RUN;

**************************************************;
* NOW DEFINE THE VIEW THAT WE WILL USE TO GET    *;
*     THE DATA FROM DB2.                         *;
**************************************************;
PROC SQL;
CONNECT TO DB2 (SSID=DB0P);
CREATE VIEW MYVU AS
SELECT *
FROM CONNECTION TO DB2
        (
     SELECT
            TGT.*
     FROM
            &OWNER..&TBNAME        TGT
           ,&OWNER..ARCHIVE_POLS   ARC

     WHERE
                %YEARCHK
             TGT.PARTITION=&PARTISHN
            AND %TGT_ARC
        );

**************************************************;
* PROCESS THE VIEW, LABEL THE DATASET, AND       *;
*     COMPRESS IF COMPRESSION IS APPROPRIATE.    *;
**************************************************;
DATA
     ARCHIVE.&TBNAME (
                    COMPRESS=&COMPRESS
                    LABEL="&TBNAME &SYSDATE"
                    )
     ;
     ************************************************;
     * MACRO LNUM BUILT BY EXECUTING MACRO GENLENTHS. *;
     ************************************************;
     %LNUM;
     SET MYVU;
************************************************;
* LOG RELEVANT DATA TO THE LOG DATASET. MACRO    *;
*     KEEPLOG USES DYNAMIC ALLOCATION, SO DO     *;
*     NOT BOTHER LOOKING IN THE JCL FOR THE DSN. *;
************************************************;
%KEEPLOG;
```

This process of archiving data from a DB2 table to a SAS set can be repeated for additional tables by simply changing the table name on SYSPARM.

At this point, we can build job streams for each partition to process the tables that need to have data archived. (Or better yet, build one job stream, and programmatically build the other 33 job streams – but that's another topic.)

AFTER THE FACT: WHERE'S THE DATA?

All of the code shown up to this point was generated for the purpose of answering the question "Where's the data for state X for year Y"?

KEEPLOG updated the archive index (ARCHIDX) with the information needed to answer this question. We can query ARCHIDX and get the list of candidate data set names that contain all of the archival data for the requested state and year. This list of candidate DSNs and the number of DSNs will be passed via global macro variables to the process that performs the retrieval.

This is macro ARCSETS (with some comments removed):

```
%MACRO ARCSETS;
%DEFAULT(YEARS,);  /* GUARANTEE THAT &YEARS EXISTS  */
%DEFAULT(STATES,); /* GUARANTEE THAT &STATES EXISTS */
%DEFAULT(PARTS,);  /* GUARANTEE THAT &PARTS EXISTS  */
%**********************************************************;
%* MACRO ARCSETS :                                        *;
%*  06/24/2009 : DROP DISTINCT CLAUSE. SWITCH TO MAX(YEAR) SO   *;
%*     THAT WE CAN PROCESS THE NEWER DATA FIRST. THIS WAS DONE  *;
%*     SO THAT NEWER COLUMNS DO NOT GET LOST.                   *;
PROC SQL NOPRINT;
     %**********************************************************;
     %* ALLOCATE THE PRODUCTION ARCHIVE LOG.                   *;
     %**********************************************************;
     LIBNAME ARCHLOG 'PNM20CA0.ARCHLOG.RUNSTATS' DISP=SHR;RUN;
     %**********************************************************;
     %* BUILD A TABLE THAT CONTAINS DISTINCT DATASET NAMES.     *;
     %**********************************************************;
     CREATE TABLE DETAIL AS (    /* TBLNAME = DETAIL */
     SELECT
            IDX.DSN
           ,MAX(IDX.YEAR)    AS MAX_YEAR
     FROM
          ARCHLOG.ARCHIDX   IDX
         ,ARCHLOG.PARTITION PRT
     WHERE

           IDX.PART=PRT.PART
       AND IDX.YEAR=PRT.YEAR

        %IF &YEARS NE
```

```
                  %THEN %DO;       /* ONE OR MORE YEARS SPECIFIED */
         AND IDX.YEAR IN (&YEARS)
                        %END;      /* ONE OR MORE YEARS SPECIFIED */

          %IF &STATES NE
                  %THEN %DO;       /* ONE OR MORE STATES SPECIFIED */
         AND PRT.ST_CD IN (&STATES)
                        %END;      /* ONE OR MORE STATES SPECIFIED */
          %IF &PARTS  NE
                  %THEN %DO;       /* ONE OR MORE PARTITIONS ....  */
         AND PRT.PARTITION IN (&PARTS)
                        %END;      /* ONE OR MORE PARTITIONS ....  */
         GROUP BY
                  IDX.DSN
                              )
       ORDER BY
             DSN      DESC

                              ;   /* TBLNAME = DETAIL */

       %*********************************************************;
       %* NOW USE AUTOMATIC MACRO VARIABLE &SQLOBS TO BUILD %GLOBAL;
       %* STATEMENTS FOR DSN1 THROUGH DSN&SQLOBS.                *;
       %*********************************************************;
       %DO _IDX=1 %TO &SQLOBS;
           %GLOBAL DSN&_IDX;
       %END;
       %*********************************************************;
       %* GLOBAL AND CREATE DSNCOUNT CONTAINING THE NUMBER OF DSNS*;
       %* THAT WE HAVE SO THAT OTHER MACROS CAN FIND OUT HOW MANY *;
       %* THERE ARE.                                             *;
       %*********************************************************;
       %GLOBAL DSNCOUNT;
       %LET DSNCOUNT=&SQLOBS;

       %*********************************************************;
       %* USE SQL TO POPULATE DSN1-DSN&SQLOBS USING THE VALUES OF  ;
       %* DSN IN THE TABLE.                                      *;
       %*********************************************************;
       SELECT DSN INTO :DSN1 - :DSN&SQLOBS
       FROM   DETAIL
       ;
       RUN;
       %*********************************************************;
       %* WRITE THE YEARS, STATES, AND THE DSNS THAT CONTAIN SAID  ;
       %* DATA TO THE SAS LOG.                                     ;
       %*********************************************************;
       %IF &YEARS EQ  %THEN %LET MSG_YEARS=ALL;
                      %ELSE %LET MSG_YEARS=&YEARS;
       %IF &STATES EQ %THEN %LET MSG_STATES=ALL;
                      %ELSE %LET MSG_STATES=&STATES;
       %PUT ****************************************;
       %PUT DATASETS SELECTED CONTAINING DATA FOR STATES: &STATES /*
            */ AND YEARS: &YEARS. ;
       %DO _IDX=1 %TO &SQLOBS;
           %PUT DSN&_IDX=&&DSN&_IDX;
       %END;
       %PUT ****************************************;
  %MEND  ARCSETS;
```

Note: This archival process was originally written for one set of tables, all containing specific columns in common, and managed at the policy level. With the introduction of PLR and an accounting-centric focus, that changed. "Archival year" for PLR is the accounting year. In addition, PLR tables with monthly data are archived sooner than PLR tables with yearly data. Having multiple archival years present within a single dataset caused those datasets to be processed twice.

AFTER THE FACT: RETRIEVING THE DATA

Macro ARCSETS generates a list of candidate DSNs and stores them in global macro variables DSN1 thorough DSNn. Macro variable DSNCOUNT contains the number of DSNs.

Now we need to loop through these DSN,s and retrieve the data. It is at this point in the process that we are going to specify the table name, and optionally, a subsetting query. The table name is used in a second query against ARCHIDX which verifies that the DSN being tested contains data for the table. If a DDname is passed on the GETDATA call, the contents of that DD are copied to the view definition.

Append processing is used to process the archival sets for the requested table, accumulating the retrieved data to DDname KEEPIT.

This is macro GETDATA:

```
%*****************************************************************;
%* PNM20CA0.M20ARC.CNTL(GETDATA). THIS MEMBER DEFINES MACRO      *;
%* GETDATA.                                                      *;
%*****************************************************************;
%* 09/09/2008: MAJOR RESTUCTURE ON HOW THE VIEW GETS BUILT.      *;
%*             REMOVE REFERENCES TO %USERQRY, AND REPLACE WITH   *;
%*             %RDFILE.                                          *;
%* 08/13/2004: MODIFICATIONS TO PROCESS TABLES FROM ANY OWNER.. *;
%*****************************************************************;
%* THIS MACRO :                                                  *;
%*      1. REQUIRES THAT MACRO ARCSET HAS EXECUTED AND           *;
%*         IDENTIIFED THE MVS DATASETS THAT ARE REQUIRED.        *;
%*      2. REQUIRES THAT A TBNAME IS PASSED VIA "TBNAME".        *;
%*      3. ACCEPTS A DDNAME PASSED VIA "QUERYDD". THIS DD CAN    *;
%*         OR SHOULD CONTIAN SQL TAHT LIMITS THE COLUMNS THAT    *;
%*         ARE SELECTED. IF SPECIFIED, THE DDNAME MUST EXIST.    *;
%*****************************************************************;
%MACRO GETDATA(TBNAME,QUERYDD);

       %*****************************************************;
       %* CHECK THE 2ND PART OF THE TABLE NAME.             *;
       %*****************************************************;
       %LOCAL FILEREF;
       %LET FILEREF=%SCAN(&TBNAME,2);
       %*****************************************************;
       %* IF 2ND PART DNE (DOES NOT EXIST), SET DEFAULT OWNER TO*;
       %* M20.                                              *;
       %*****************************************************;
       %IF &FILEREF= %THEN %DO;
                     %LET OWNER=M20;
                     %LET TBNAME=%SCAN(&TBNAME,1);
                     %END;
              %ELSE %DO;
                     %LET OWNER=%SCAN(&TBNAME,1);
                     %LET TBNAME=%SCAN(&TBNAME,2);
                     %END;

        %DO  DSNIDX=1 %TO &DSNCOUNT; /* DSN1 - DSN&DSNCOUNT */


        PROC SQL  NOPRINT;
             SELECT COUNT(*) INTO :GOOD_DSN
             FROM ARCHLOG.ARCHIDX
             WHERE DSN="&&DSN&DSNIDX"
               AND MEMNAME="&TBNAME"
               AND OWNER  ="&OWNER"
               ;
             %IF &GOOD_DSN NE 0 %THEN %DO;    /* DSN CONTAINS TBNAME*/

             LIBNAME &OWNER
```

```
                    "&&DSN&DSNIDX"
                    DISP=SHR;RUN;
                    PROC SQL;
                    CREATE VIEW &TBNAME AS (

                                %IF &QUERYDD NE %THEN %DO;
                                                      /********************/
                                                      /* USER HAS A QUERY */
                                                      /********************/
                                                      %RDFILE(&QUERYDD)
                                                      %END;
                                                 %ELSE %DO;
                                                      /********************/
                                                      /* GEN THE DEFAULT  */
                                                      /********************/
                                                      SELECT *
                                                      FROM &OWNER..&TBNAME
                                                      %END;

                                           );
                    PROC APPEND DATA=&TBNAME BASE=KEEPIT.&TBNAME FORCE;
                    RUN;
                    LIBNAME &OWNER CLEAR;
                    RUN;
                                             %END;   /* DSN CONTAINS TBNAME*/
               %END;                        /* DSN1 - DSN&DSNCOUNT */
       %MEND  GETDATA;
```

ASSEMBLING THE RETRIEVAL

We now have the tools that we need to perform a basic data retrieval. If we supply the codes for the state or states, ARCSETS will find all of the datasets that contain data for that list of states. If we specify one or more archive years, the retrieval will be restricted to those archival years.

The JCL and SAS code shown below will retrieve the archived data for archival year 1997, state codes ‘08’ and ‘45’ (Washington, DC, and Virginia).

```
//S1      EXEC SAS
//MYMAC   DD DISP=SHR,DSN=PNM20CA0.M20ARC.CNTL
//KEEPIT  DD DSN=TMVS727.KEEPIT.MYPROJ,
//           DISP=(NEW,CATLG,DELETE),
//           UNIT=(SYSDA,8),SPACE=(CYL,(20,20))
//
//SYSIN DD *
 OPTIONS SASAUTOS=(MYMAC,SASAUTOS);
 %LET STATES='45','08';   /* USED BY ARCSETS */
 %LET YEARS=1997;         /* USED BY ARCSETS */
 %ARCSETS;
 %GETDATA(POLICY);        /* USES VALUES RETURNED BY ARCSETS */
 %GETDATA(VEHICLE);       /* USES VALUES RETURNED BY ARCSETS */
```

I believe that this satisfies the objective of simplifying the retrieval process – this is the same code that was presented in the section titled “The History of MDW / M20”.

HOW WELL HAVE THESE PROCESSES WORKED?

The basic methodology has worked so well that the process has been copied and modified for four other production DB2 databases. We (Marketing Support) are currently using the archives to respond to data requests two or three times a month.

The original topic of archiving was brought up shortly after it was discovered that the DB2 utility backups for the most recent “archival” had been deleted. There was also a short window (two months?) for development, testing, validation, and implementation before the process needed to be executed.

We originally focused on just being able to archive the data in a manner that would let us find it again at a later date.

The first request for data from the archives was quite a shock – it was received the Tuesday immediately following the purge. We were able to respond to the user by Wednesday morning (the archival files were still on disk at that time).

The second request was received less than 2 months after the initial purge. and was followed in short order by requests three and four. It was then that I began documenting the manual version of the process for using the archive index.

It was while I was documenting the process that I realized that there were no decision points in the process that required intervention. This realization spurred the development of ARCSETS and GETDATA. This has simplified the retrieval process to the point that some of our users are performing retrievals without any assistance from our group.

I am aware of only one instance of the archival data being used to directly recover DB2 data that was erroneously removed from a table – but that instance did prove that data could be loaded from the archives back into DB2.

We did run into some operational issues that were caused by too many archive sets being resident on disk simultaneously. The Storage Management group set up a separate management class for the archival sets, and this problem has not recurred.

The majority of problems have been caused by human error, such as being told that a table has been dropped from DB2 when it has not been dropped.

ACKNOWLEDGMENTS

I would like to thank the support personnel at SAS.com for their help when I have managed to get in over my head with the macro facility or the DB2 interface.

Contact Information

Your comments and questions are valued and encouraged. Contact the author at:

Name: Rich Morris
Enterprise: Progressive Insurance
Address: 300 N. Commons Blvd   Box W94
City, State ZIP: Mayfield Village, Oh, 44143
Work Phone: (440)395-7925
Fax:
E-mail: Rich_Morris@Progressive.com
Web: