Paper 17 – 28

# Using Different Methods for Accessing Non-SAS® Data to Build and Incrementally Update That Data Warehouse

Ben Cochran, The Bedford Group, Raleigh, NC


## Abstract

Often SAS users need to access data from non-SAS sources. This is especially true when constructing a SAS data warehouse from other vendors' databases. While this task is not too difficult, sometimes unforeseen challenges can arise, especially when dealing with date values. This tutorial initially takes a look at several methods for accessing different kinds of data to do the initial load of the data warehouse. Then attention is given to various ways of doing incremental updates and how to overcome some potential problems.

This paper follows the tasks that were involved in a specific retail application and how a certain organization faced and overcame the challenges that accompany building and updating a data warehouse. To accomplish its objectives, this paper is divided into seven sections. The first section looks at the environment and issues surrounding the initial load of the warehouse. The second part looks at the inevitable task of data manipulation, specifically dealing with date values. The third part examines methods for finding out the maximum date value of transactions in the data warehouse. Next, the paper looks at finding the maximum date values in the operational data that feeds the warehouse. The fifth step looks at the method for comparing the maximum date of the warehouse transactions with the maximum date of the operational data. The sixth step looks at doing the actual updating itself. The seventh step looks at accomplishing the above by using the SAS/ACCESS® LIBNAME statement.


## Part I: Introduction and Initial Load

**Background:** A national retail organization recently built a data warehouse that allows it to analyze the purchasing patterns of its customers. The organization wanted to find out who its best customers are so that ways might be devised to reward them, with programs like a frequent buyer's club. It also wanted to find out who its worst customers are, so that it can cease mailing them expensive catalogs on a monthly basis.

This organization has almost 100 stores all across the United States. Every night, each of these stores uploads its transactional data to a regional server. Then, every few days, the sales data is loaded into Sybase tables. The plans are to build a SAS data warehouse (or data mart) from the Sybase tables (figure. 1)

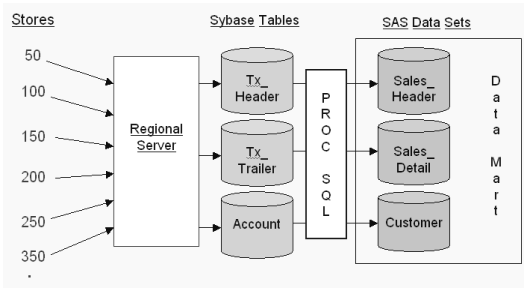


Figure 1.

For the sake of simplicity, this paper will focus only on the loading and updating of the Sales_Header SAS dataset from the Tx_Header Sybase table. Before the code can be written to perform the initial load, the column names of the Sybase tables need to be known. The column names for the Tx_Header Sybase table are shown below (Figure 2). Notice the column names. Especially notice that there are three separate fields for the date of the sale; rcpt_dte_yy, rcpt_dte_mm, and rcpt_dte_dd.

The TX_HEADER Sybase table has the following fields and values.

| account_ number | sales_ amount | rcpt_ dte_yy | rcpt_ dte_mm | rcpt_ dte_dd | rcpt_ number | store_id |
|---|---|---|---|---|---|---|
| R03012345 | 65.00 | 96 | 10 | 23 | 5001 | 50 |
| R03044567 | 77.00 | 99 | 02 | 28 | 5032 | 50 |
| R03022345 | 165.00 | 99 | 12 | 22 | 5101 | 50 |
| R03034567 | 277.00 | 98 | 02 | 11 | 5132 | 50 |
| R04012345 | 5.00 | 99 | 11 | 03 | 5201 | 50 |
| R04044567 | 97.00 | 97 | 02 | 28 | 5232 | 50 |
| R04022345 | 265.00 | 99 | 12 | 22 | 5201 | 50 |

Figure 2.

There are several ways to read Sybase data with the SAS system. The following code represents a method using PROC SQL with the Pass-through facility (Figure 3).

```
proc sql;
   connect to sybase as sas_sql
      ( server = ADAM    database = DIS01P
        user   = BTC     password = SECRET ) ;
   create table sales_header as select  *
      from connection to sas_sql
      ( select   account_number  as  acct_id ,
                 sales_amount   as   total_amount ,
                 rcpt_dte_yy    as   sales_year ,
                 rcpt_dte_mm  as   sales_month ,
                 rcpt_dte_dd    as   sales_day ,
                 rcpt_number   as   rcpt_number ,
                 store_id          as  store
          from  tx_header )  ;
          disconnect from sas_sql;
quit;
```

Figure 3.

Once this code is executed, the SAS dataset: Sales_Header is created from the Tx_ Header Sybase table. Similar SQL steps are used to read the other necessary Sybase tables, and this represents the initial load of the data warehouse (or data mart).

## Part II: Data Manipulation

There is always a need to manipulate the data. In this case, only one column for date values is needed instead of three. The following DATA step (Figure 4) is typical of the type of data manipulation that is needed.

```
data sales_header (drop = sales_month sales_day
                          sales_year);
    set  sales_header;
    sales_date = mdy ( sales_month, sales_day,
                       sales_year);
run;
```

Figure 4.

## Part III: Find the Latest Sales_Date in the Data Warehouse

For our purposes in this example, we are going to concentrate on the values of SALES_DATE to tell us whether it is time to update the SAS data warehouse from the Sybase tables. In other words, if the maximum value for the sales date is greater in the Tx_Header Sybase table than in the Sales_Header data set in the warehouse, then it is time to update. Part three illustrates a method for finding out the latest sales date in the warehouse. The DATA step is used to do this, and the exact code is illustrated in Figure 5.

```
data _null_;
   retain maxdate 0;
   set sales_header end=e;
       if sales_date > maxdate then maxdate =
          sales_date;
    if e ;
    call symput('cutoff', trim(left (put (maxdate, 8.)))) ;
run;
```

Figure 5.

After this DATA step runs, the maximum sales date is stored in a macro variable named **&CUTOFF**. The next step is to find the latest date of a sale in the Tx_Header Sybase table.

## Part IV: Find the Latest Sales Date in the Tx_Header Sybase Table

Remember that the RCPT_DTE_YY column has only 2 digits that represent the year. Before the year 2000, the following method found in Figure 6 could be used.

```
proc sql ;
   connect to sybase as sas_sql
      (server = ADAM     database = DIS01P
       user = BTC        password = SECRET ) ;
   SELECT  input ( put (last_dte, 8.), yymmdd8. )
   into :  lastsdte
   from connection to sas_sql
        (select max ((10000 * rcpt_dte_yy )  +
                     (100     * rcpt_dte_mm) +
                      rcpt_dte_dd )  as last_dte
        from   tx_header )     ;
   disconnect from sas_sql ;
quit;
```

Figure 6.

The pass-through part of this code essentially builds a date value in the yymmdd8. form. It does this by taking the 2-digit value of year and multiplying it by 10,000. This value is added to the results of multiplying the 2-digit value of month. Then finally, this is added to the 2-digit value of RCPT_DTE_DD. Then, the MAX function finds the maximum date value. Once this is found, it is pasted into a variable called LAST_DTE. This variable passes its value into an SQL macro variable named LASTSDTE.

Again, the program and logic represented here are valid only if the year value is a 2 digit number. If the year value has 4 digits, the method found in Figure 7 will have to be used.

```
proc sql;
    connect to sybase as sas_sql
        ( server = ADAM    database = DIS01P
          user   = BTC       password = SECRET ) ;
    create table s_date_check as select  *
         from connection to sas_sql
         ( select   rcpt_dte_yy    as   sales_year,
                    rcpt_dte_mm  as     sales_month ,
                    rcpt_dte_dd    as   sales_day
                         from  tx_header )   ;
              disconnect from sas_sql;
 quit;

data _null_ ;
   retain s_maxdate 0 ;
   set s_date_check end = e;
   if sales_year lt 10  then
      sales_year = sales_year + 2000;
      else sales_year=sales_year + 1900;
  sales_date = mdy (sales_month, sales_day,
                        sales_year);
   if sales_date gt s_maxdate then
     s_maxdate = sales_date;
    if e;
       call symput ('lastsdte', s_maxdate);
 run;
```

Figure 7.

We now have a macro variable that contains the latest sales date from the Sybase table Tx_Header (LASTSDATE), and we also have a macro variable from part 3 that contains the latest sales date from the SAS data warehouse.

## Part V: Compare the Two Dates

The next step is to compare the two dates to see if the data warehouse needs to be updated. Since both date values are stored in macro variables, their values can be compared by writing a short macro like the following one found in Figure 8.

```
%macro refresh;

     %if &lastsdte  >  &cutoff   %then  %do ;
           [   the PROC SQL code  ]
           [   from the next step      ]
           [   goes here                 ]
       %end ;

%mend ;

%refresh ;;
```

Figure 8.

If the condition in the REFRESH macro is true, then the Sybase tables have more recent records than those in the data warehouse. Therefore, the data warehouse needs to be refreshed with new records from the Sybase table. The program in the next section does the refreshing, and goes in the REFRESH macro between the %DO and the %END.

## Part VI: Update the Data Mart With The Newest Sybase Records

Once we determine that the warehouse needs updating by comparing the two dates above, we need a way to pass the maximum date values from the SAS data warehouse into the program that reads the Sybase data. This is done by using the following three %LET statements seen in Figure 9.

```
%let mm = %sysfunc (month ( &cutoff ), z2.) ;
%let  dd =  %sysfunc (day  ( &cutoff ), z2. ) ;
%let  yy =  %sysfunc (year ( &cutoff ), z4. ) ;
```

Figure 9.

If, for example, the value of **&cutoff** is 15340 (December 31, 2001), then the value of **:**

- **&mm** is equal to **12**
- **&dd** is equal to **31**
- **&yy** is equal to **2001**

Remember, the date values in the Sybase table are in three independent columns: one for sales month, one for sales day, and one for sales year. Also remember that the year values in the Sybase table have only two digits. We need a way to convert them to four digits so that we can compare them to the four digit values of **&yy**. The next task is to create a four digit value for year. Using the following CASE clause that will be placed in the PROC SQL program does this.

```
case
  when rcpt_dte_yy  lt  10 then 2000 + rcpt_dte_yy
  else  1900  + rcpt_dte_yy
end as yyyy
```

Figure 10.

The above CASE clause creates a new variable called YYYY that contains the four digit value

for year. However, because YYYY is not in the Sybase table, it cannot be used in a WHERE clause. We can still use the subsetting logic in our SQL step by using the HAVING clause. Note the use of the HAVING clause, CASE clause, and the %LET statements in the following SQL code (Figure 11).

```
%let mm = %sysfunc( month( &cutoff ), z2. ) ;
%let dd = %sysfunc( day( &cutoff ), z2. ) ;
%let yy = %sysfunc( year( &cutoff ), z4.) ;

proc sql ;
    connect to sybase as sas_sql
      (server = ADAM   database = DIS01p
       user   = BTC    password = secret ) ;
    create table work.sales_header as
        select mdy( month, day, year ) as
                sales_date,  acct_id,  total_amount ,
                store ,  rcpt_nbr

    from connection to sas_sql
      ( select  account_number  as acct_id,
                sales_amount   as  total_amount ,
                store_id          as  store,
                rcpt_dte_mm    as  month ,
                rcpt_dte_dd    as day,
                rcpt_dte_yy  as  year,   rcpt_nbr
  case
    when rcpt_dte_yy  lt 10 then 2000 + rcpt_dte_yy
      else 1900 + rcpt_dte_yy
  end as yyyy
  from  tx_header

  having ( yyyy > &yy )  or
          ( yyyy = &yy  and  rcpt_dte_mm > &mm ) or
          ( yyyy = &yy  and  rcpt_dte_mm = &mm
                  and rcpt_dte_dd > &dd ))
          order by rcpt_nbr;
          disconnect from sas_sql ;
quit ;
```

Figure 11.

After reading the new Sybase rows into the WORK.SALES_HEADER data set, we can then **append** them to the permanent SAS data set.

## Part VII: Using the SAS/ACCESS LIBNAME Statement

Beginning experimentally in Version 7 of the SAS System and continuing in production mode with Version 8, the SAS/ACCESS product provids its own enhancement to the LIBNAME statement that makes the work covered in Parts I through VI of this paper much easier. Now, a user of SAS can take advantage of these features and use them instead of the Pass-through facility of PROC SQL.

First, let's take a look at the SAS/ACCESS LIBNAME statement. The general syntax is:

```
libname  libref  engine – name
                 engine – connection options
                 engine – LIBNAME options ;
```

Figure 12.

Specifically, for the retail example that this paper has been following, to access Sybase data, the LIBNAME statement should look like this:

```
libname  s_tables   sybase
                 user = BTC123
                 password=XXXXXX
                 database = lwdw01p
                 server = lwdw01p
                 preserve_tab_names = yes
                 schema = DBA ;
```

Figure 13.

Once this LIBNAME statement is issued, SAS treats all the Sybase tables in the 'lwdw01p' database as thought they are SAS data sets in the same SAS library. For instance, we can issue the LIBRARY command and the following LIBRARY window appears:

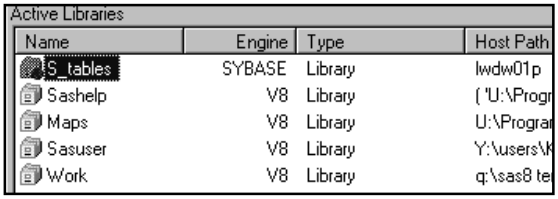


Figure 14.

If we double click on the S_tables icon in the LIBRARY window, we will see the following:

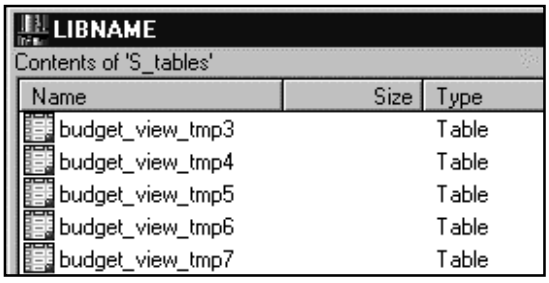


Figure 15.

While these icons look like they are representing SAS data sets, they are in fact Sybase tables. The magic of the SAS/ACCESS libname statement is that it can present these tables to the SAS System in a way that they are treated like SAS data sets. For example, if we double-click on the **budget_view_tmp3** icon above, we will see:

VIEWTABLE: S_tables.budget_view_tmp3

| | sbu_grp_id | div_dept_grp_id | acct_type_cde | maj_class_ |
|---|---|---|---|---|
| 1 | 1 | 04 | 4 | 101 |
| 2 | 1 | 04 | 4 | 101 |
| 3 | 1 | 04 | 4 | 101 |
| 4 | 1 | 04 | 4 | 101 |
| 5 | 1 | 04 | 4 | 102 |
| 6 | 1 | 04 | 4 | 101 |
| 7 | 1 | 04 | 4 | 102 |
| 8 | 1 | 04 | 4 | 101 |
| 9 | 1 | 04 | 4 | 101 |
| 10 | 1 | 04 | 4 | 101 |
| 11 | 1 | 04 | 4 | 101 |
| 12 | 1 | 04 | 4 | 101 |
| 13 | 1 | 04 | 4 | 101 |

Figure 16.

Because a VIEWTABLE window opens, we may think we are looking at the rows and columns of a SAS data set. But, again, the data are physically stored in a Sybase table. So now, when we want to refer to this Sybase table in our SAS program, we can refer to it using the same syntax as we would a SAS data set. In other words if we want to see the first 10 rows of this Sybase table, we can submit the following SAS code:

```
proc print data =
        s_tables.budget_view_tmp3(obs=10) ;
run ;
```

Figure 17.

Knowing how this type of LIBNAME statement works, we can do an initial load of the data warehouse AND do data manipulation using the following DATA step:

```
data dw.sales_header ;
    set s_tables.Tx_Header ;
    sales_date = mdy ( rcpt_dte_mm ,
                       rcpt_dte_dd ,
                       rcpt_dte_yy ) ;
run ;
```

Figure 18.

At this point, we can a DROP the three independent variables that are used in the MDY function to make up the date.

The next step is to find the latest sales date in the data warehouse. We can use the same DATA step as before:

```
data _null_;
   retain maxdate 0;
   set sales_header end=e;
      if sales_date > maxdate then maxdate =
         sales_date;
    if e ;
    call symput('cutoff ', trim(left (put (maxdate, 8.))))
;
```

Figure 19.

To find the latest sales date in the Sybase table we can use the following DATA step.

```
data _null_;
   retain s_maxdate 0;
   set s_tables.tx_header ( keep = rcpt_dte_mm
                rcpt_dte_dd  rcpt_dte_yy )  end=e;
   sales_date = mdy ( rcpt_dte_mm , rcpt_dte_dd ,
                      rcpt_dte_yy ) ;
   if sales_date > s_maxdate  then
                s_maxdate = sales_date ;
    if e ;
    call symput ( ' lastsdte ',
                 trim( left ( put ( s_maxdate, 8. )))) ;
run;
```

Figure 20.

We now have the maximum sales date for the SAS data warehouse, and the maximum sales date from the Sybase tables. We can use the same REFRESH macro to compare dates.

We can use the following DATA step to refresh the data warehouse.

```
data  new;
   set s_tables.tx_header ;
   sales_date = mdy ( rcpt_dte_mm , rcpt_dte_dd ,
                      rcpt_dte_yy ) ;
   if sales_date > &cutoff ;
run;
```

Figure 21.

And finally, we use the following code to append the new data set to the existing data set in the warehouse.

```
proc append base = dw.sales_header  base = new;
run;
```

Figure 21.

## Conclusion

There are a number of ways to build and update a Data Warehouse using SAS. This paper initially illustrates using the Pass-Through facility of PROC SQL. If one has licensed the SAS/ACCESS product then a more direct and less complex solution can be used. This paper illustrates the power and flexibility of using the SAS system.

The author can be reached at:

Ben Cochran
The Bedford Group
3216 Bedford Avenue
Raleigh, NC 27607
(919) 831-1191
bedfordgroup@nc.rr.com
www.bedford-group.com