

A Legislative Bill Text Retrieval and Distribution System Using SAS, PROC SQL, and SAS/Access to DB2

John Turman and Kathe Richards
Technical Support, Application Systems Division
Texas Comptroller of Public Accounts

One of the signs of the approach of winter in even numbered years around state agencies is the flurry of activity associated with getting ready for the Legislature to convene in the spring to determine our fates. Last year, as the session loomed, the committees examining our processes and procedures for doing analysis of proposed legislation produced a wish list of supporting automation. Included prominently on this list was the ability to be able to store, retrieve, read and print proposed bills electronically.

The process of reviewing and analyzing proposed legislation in our agency typically involves many people, particularly when the legislation involves taxes, fees, funds, and state administrative policy -- and a good bit of it does. Handling the distribution of this text on-line would save us the expense and time delay involved in copying the bills and distributing them to the subject matter experts responsible for determining the administrative and fiscal impact the bill would have on the agency and state.

Initially, we looked for a classic document-based implementation. We thought maybe something like passing the bills around like word processing documents on a local area network would be the best approach. This theory ran into two snags. The subject matter experts were scattered all over the agency, in several different physical locations, and with access to an incredible mix of hardware, software and network configurations, including none. The lowest common denominator was clearly the mainframe. Also, the system being developed to support tracking and collecting analysis on bills was a mainframe CICS system using DB2.

It became clear that it would be most useful to have the text available, if not directly through the tracking system, at least closely associated with it, since people doing the analysis work needed the text while they were doing it. Unfortunately, it did not become clear until considerable time had been expended exploring the other possibilities.

By this time there was not much implementation time left. We turned once again to the tool that has bailed us out regularly in the past -- SAS. SAS, running on our mainframe, was clearly the least common denominator from the equipment mix standpoint. We had staff and expertise available. And we could be assured that the development effort would be quick.

It was decided that the text would be made available for viewing on-line through the CICS system and would be stored in DB2 tables. We get the text from one of the legislative services on a tape in the wee hours of the morning. It is read and reformatted using SAS before being loaded to the DB2 tables.

Originally, the plan was for the users to be able to get print of the bills after reviewing them on-line by executing a CICS transaction that submitted a print job to the internal reader. Because of the early indecision about how to handle the text distribution, however, this piece didn't get included in the early design of the on-line system and we discovered that some people using the system were getting their print by using CICS screen prints. This was clearly unsatisfactory. Some divisions extracted their text and downloaded to their Local Area Networks where the LAN users could import the text into word processing packages for print. Other users routed the print from the batch SAS jobs to their local VTAM printers.

Formatting and printing the bills themselves was fairly straightforward. The SAS jobs to format the print select the bill to be printed on the basis of bill number and session number stored with the lines of text. Each line of text is a separate row in the DB2 table. We used PROC SQL to select text for specific bills and a DATA _NULL_ step with PUT statements to print the text in the format everyone was used to seeing. The initial code looked something like this:

```

PROC SQL;
  CREATE VIEW BILLS AS
    SELECT * FROM DB2FILE.BILLTEXT
      WHERE SESSION = '7300' AND
            BILLTYPE = 'HB ' AND
            BILLNUM = 1 AND
            VERSION = 1;

```

```

DATA _NULL_; SET BILLS;
              FILE PRINT;
              PUT @1  LINENUM  4.
                  @10 TEXTLINE $70.;

```

To take care of the people who were doing screen prints and to make life easier for the folks who were routing print to mainframe printers, we built an interactive front-end procedure that allows the user to enter a list of bills. This proc builds a file of keys of the bills to the batch SAS job to be selected, formatted and printed. The PROC SQL statement that supported this now became more complicated. The input to the SELECT statement was now a group of parameters instead of values.

```

DATA  BILLS;
  INPUT  @1  SESSION    $4.
         @5  BILLTYPE   $3.
         @8  BILLNUM    4.
         @12 VERSION    2.;

```

```

CARDS;
7300HB  1 1
7300HB  2 1
;

```

```

PROC SQL;
  CREATE VIEW BILLS AS
    SELECT * FROM DB2FILE.BILLTEXT A BILLS B
      WHERE A.SESSION = B.SESSION AND
            A.BILLTYPE = B.BILLTYPE AND
            A.BILLNUM = B.BILLNUM AND
            A.VERSION = B.VERSION;

```

etc.

Several similar jobs were built to take care of special needs of different divisions. After working out all the minor problems associated with print routing and weird configurations, this technique worked out pretty well. In fact, there were virtually no problems associated with it until late in the session when one of the divisions began reporting missing print.

When we looked at the output files from the jobs with problems we noticed that these little print jobs that ran in sub-seconds early in the session were now taking minutes of CPU time. The jobs with missing print were running out of time and ABENDING.

Clearly the amount of data in the table holding the bill text was growing. With text from every bill proposed during the legislative session being added to the table, it was to be expected that we would have quite a bit of data by the end of the session. What was unexpected was the retrieval time in the PROC SQL statement.

We determined pretty quickly that because SAS has no way of telling DB2 that the selection criteria being passed to DB2 are keys, DB2 does a table space scan to satisfy the request. When there are several bills to be selected, the cumulative effect of all the table space scans soon starts to amount to quite a bit of processing.

PROC SQL operates within SAS. The only contact it has with DB2 is when it uses an access and view descriptor to get to a DB2 table. Even if our group of keys was in another DB2 table, PROC SQL would request the complete set of data from each table required to make the selection from both tables, pull them into the SAS work space, and perform the actual selection within SAS. The only way to take advantage of the power of DB2 to make the match would have been to use the SQL Pass-Thru feature of PROC SQL. This wasn't helpful in our case, however, since the proc that builds the key file was not building it as a DB2 file. (Making it a "temporary" DB2 table was considered as a possibility, but the administrative problems associated with this, such as getting the table defined, setting up the security, etc., proved to be daunting.)

When we presented the match values in the SELECT statement as literals -- numbers and text strings -- it was clear that DB2 would use the data values to do a keyed match on the table. The problem with this is that we had a variable number of keys. We thought first about building a macro that generates the code for each SELECT statement and execute it. This turned out to be much more complicated than it looks. We could fairly easily build and execute one SELECT statement, but doing a sequence of them proved to be problematical. The situation grew to be very complex and the effort was abandoned. (There is a sample of how to do this in the SAS Sample Library but I found it to be far from straightforward.)

Material from a previous SUGI Conference included the suggestion that a side effect of the MODIFY statement could possibly work for us here:

```
DATA DB2FILE.BILLTEXT BVTX;  
    MODIFY DB2FILE.BILLTEXT BILLS;  
    BY SESSION BILLTYPE BILLNUM VERSION;  
  
IF _IORC_ = 0 THEN OUTPUT BVTX;  
_ERROR_ = 0;
```

We were warned that this approach only works if KEY does not repeat in A. It sends repeated where clauses to the DBMS and does not scale well for lots of values of KEY. We managed to make this work, but we could not be assured that there would not be repeated keys. This also required that we have a full key to work with, so we would have had to add logic to be sure we were trapping all the qualifying rows. Too messy.

The ultimate and most elegant solution to this problem was to build a temporary file of SELECT statements, one for each requested bill, using DATA _NULL_ and PUT statements. Then we executed them using %INCLUDE. (Because we wanted a separate print file for each bill, we actually built a complete PROC SQL step for each and followed it with the print formatting step.)

```
FILENAME  TEMPSQL '&&TMPSQL';
```

```
OPTIONS $DB2DEBUG;  /* Allows you to see what SQL calls SAS is  
                      generating */
```

<< DATA step that generates the SAS dataset of keys called BILLS
goes here >>

```
DATA _NULL_;  
    FILE TEMPSQL;  SET BILLS;  
PUT "PROC SQL; ";  
PUT "CREATE VIEW BVTX AS ";  
PUT "SELECT * FROM DB2FILE.BILLTEXT "  
    "WHERE SESSION = " SESSION  
    " ' AND BILLTYPE = ' " BILL  
    " ' AND BILLNUM = " BILLNUM  
    " AND VERSION = " VERSION  
    ",";";
```

```
PUT "PROC PRINT DATA=BVTX; ";
```

```
RUN;
```

```
%INCLUDE TEMPSQL;
```

From the standpoint of CPU utilization and ease of maintenance, this turned out to be quite satisfactory. The SASLOG can get to be quite large here since each execution is faithfully reported, but that proved to be a minor concern, since the users don't see that. We were able to cut down the run time for an average selection of bills from close to 5 minutes of CPU time to a couple of seconds (and make our computer performance guys really happy.)