

## Designing an Application with Business Intelligence

Brian Deitch, Federal Deposit Insurance Corporation, Washington DC.

### ABSTRACT

Business intelligence (BI) broadly refers to Information Technology (IT) that is tailored to help staff make better business decisions. At the Federal Deposit Insurance Corporation (FDIC), SAS/AF® and Base SAS® Release 8.2 are combined in a single application to allow users to query and report information that would otherwise require considerable IT skills to produce. This paper will illustrate how the application integrates these products to provide staff with BI capabilities.

### INTRODUCTION

The FDIC is mandated by Congress to insure checking and savings accounts of member banks. Like any organization, business processes are affected by IT constraints: software must function in both mainframe and server environments; data are stored in relational database tables or flat files; data sources are wide and deep, offering users hundreds of variables to choose from when querying data; and not all data is structured uniformly.

Staff, therefore, need an application to offer a consistent framework for reading, processing, and displaying information. The application has to allow users to append or join tables, create time-series variables and fields derived as of lagged time-periods to facilitate on-the-fly calculation of growth rates. The application must insulate users from the hardware/software issues that often complicate the process of capturing data, and balance user-friendliness and sophistication to satisfy users with IT skills ranging from novice to expert.

### APPLICATION DESIGN

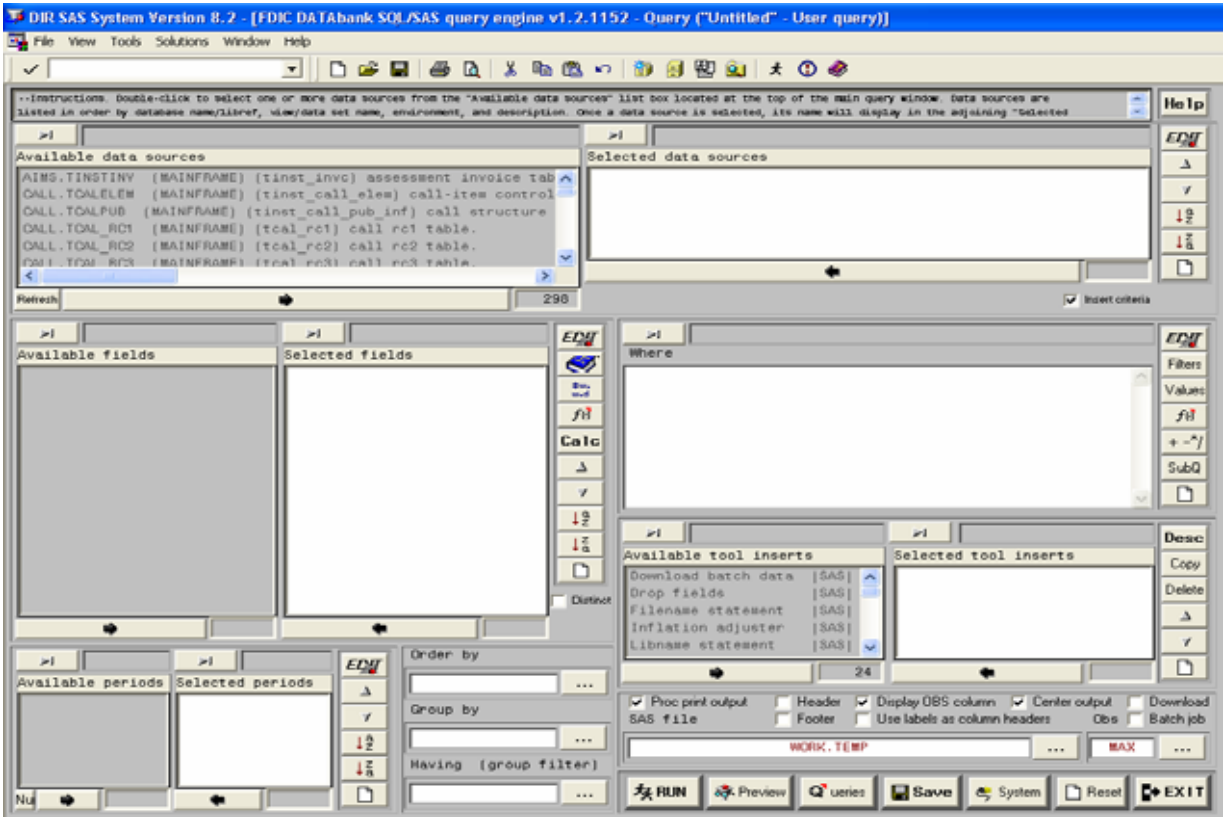
The application designed to meet these requirements, called Databank, is modeled on Structured Query Language (SQL) concepts. SQL is widely understood by IT professionals and is easy for non-programmers to learn. Its basic components – table selections, field columns, and a *Where* clause – are well suited for graphical-user-interfaces, where each component may be conceptualized with a screen object. Since the SAS Component Object Model (SCOM) offered a powerful platform for object-oriented development, the interface was created with SAS Component Language (SCL).

The application stores query selections in SCL lists, where each list contains the items used to populate a screen object. Users may save their selections for later recall or execute them as a batch query. To execute the selections as a query, the list items are assembled into SQL procedures and data steps, and executed as a SAS program (see APPENDIX A for the lists used to create a query). Once the query has finished running, users may view the output from SAS, Microsoft® Excel, or Access.

### INTERFACE DESIGN

All main components of the application are placed inside one window to maximize ease of use (see Figure 1). With one window the process of building a query is simplified since less work is required to navigate multiple screens. By reducing the amount of work required to view, select, and edit information, the application is easier to learn, or re-learn after periods of infrequent use. Object integration and communication are also maximized, allowing drag-and-drop and other event handling.

Once a data source is selected, other screen objects are populated with selectable items (as shown in Figure 2). To navigate the interface, users press the [Tab] key on the computer keyboard to move the pointer from object to object, or engage the mouse to point to objects for direct focus.



**Figure 1. Most query selections are assigned from one window, allowing object integration to become maximized. Several components within this window will be examined more closely below.**

Most screen objects are represented by composite classes for maximum functionality. Selectable items, for example, are displayed in dual list box objects, where items are listed inside an *Available* list box and when selected, move into an adjoining *Selected* list box. The advantage of a dual list box is that information about *Available* and *Selected* items are observable together, which helps users to assess the scope of their query at a glance (see Figure 3).

Text boxes located above each *Available* and *Selected* list box allow users to quickly search for and select items – as users enter text inside the lookup box, the list box below it is searched; if a match is found, then the list box scrolls to the item and highlights the item inside the list (see Figure 4).

To select the item once it is highlighted, users only need to press the [Enter] key on their computer keyboard, which moves the highlighted item to the *Selected* list box (see Figure 5). With this design, users do not have to lift their hands from the keyboard to select items, which makes selections relatively easy (the SCL methods that provide these lookup and selection capabilities are shown in APPENDIX B and C).

In addition to the above, users may choose to select items using traditional methods: drag-and-drop; double-clicking; or by highlighting one or more items with a single mouse-click and then by pressing a button to move the highlighted items into the *Selected* list box.

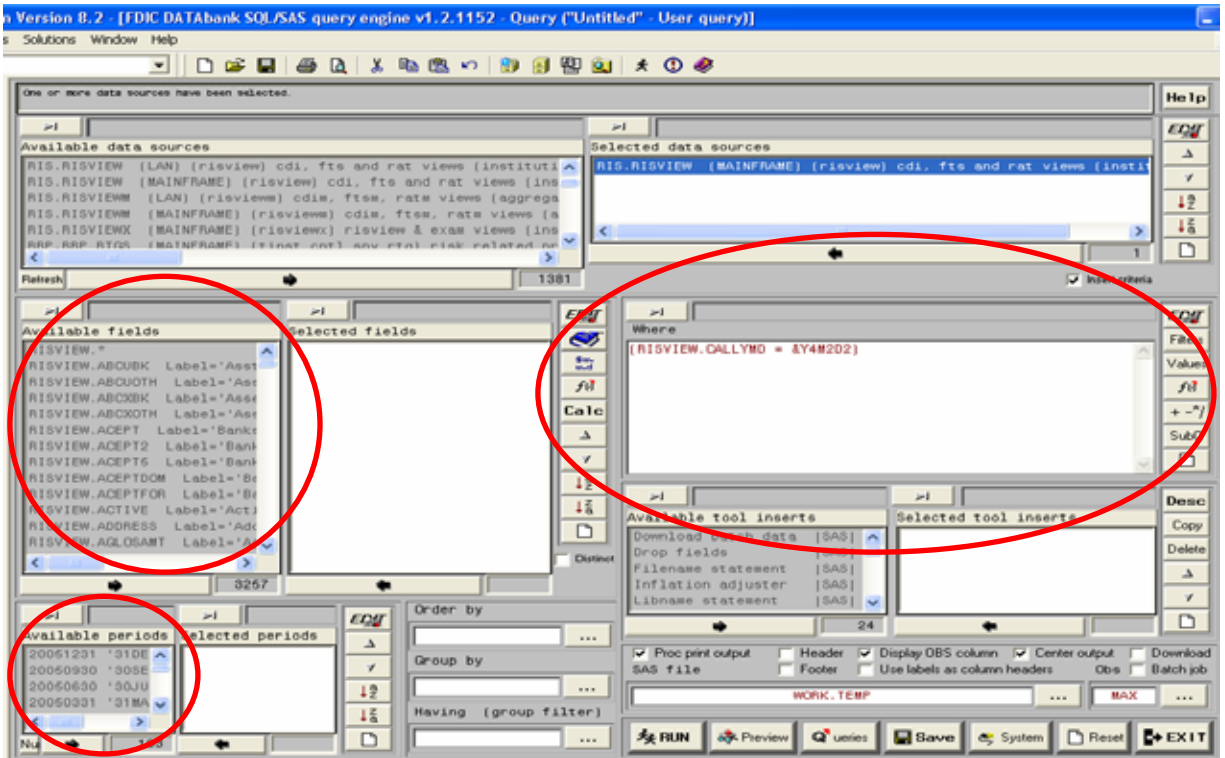


Figure 2. Once a data source selection is completed, the *Available fields* and *Available periods* list boxes, and a *Where* clause box (all circled) become populated automatically.

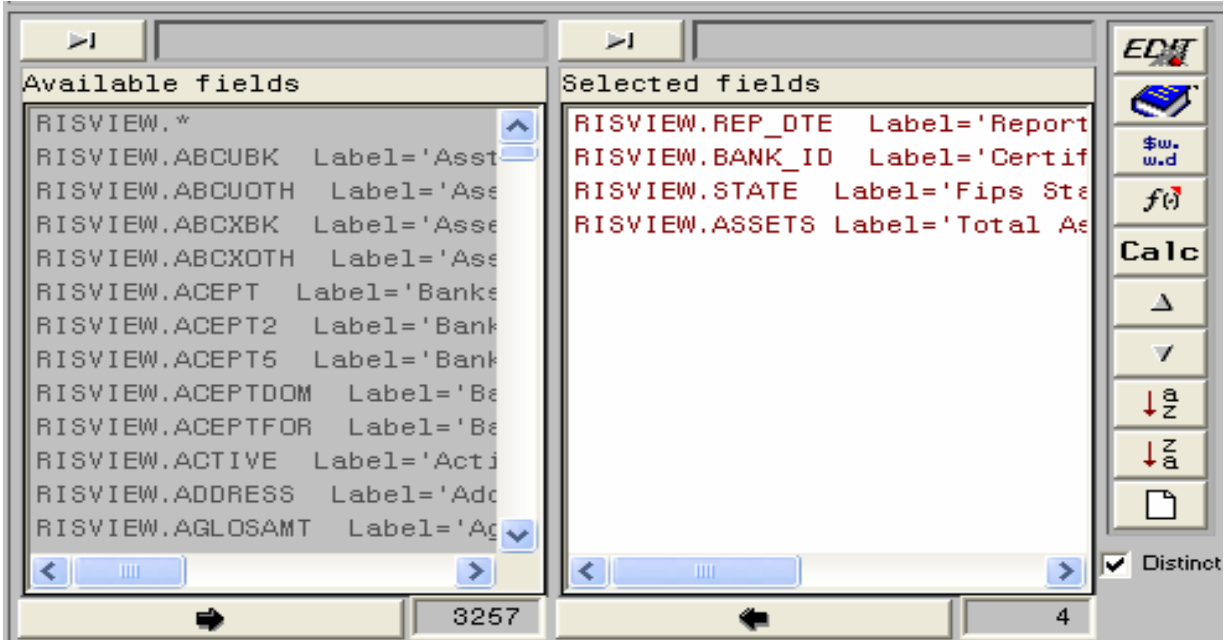


Figure 3. The figure shows a composite dual list box class containing the *Available fields* and *Selected fields* list boxes. The data source selected here contains over three thousand variables. Text boxes located above each list box allow items to be quickly located and selected, even when the item list is as long as this one. The *Selected fields* list box on the right displays four items selected from the *Available fields* list box on the left.

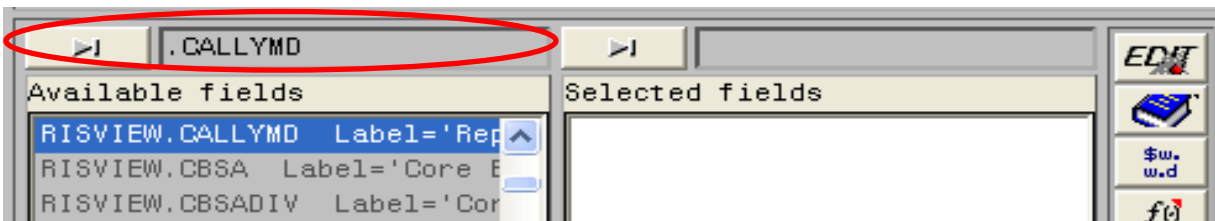


Figure 4. When text is entered inside a lookup text box (circled above), the list box scrolls to the item and highlights it; this functionality is provided by extending the `_OnValueChanged` method of the text box's `TextEntry` class with a method override.

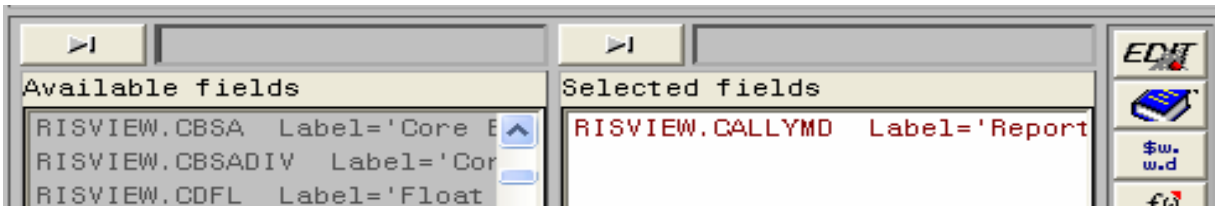


Figure 5. Once highlighted, the item is selected after the [Enter] key is pressed; this functionality is provided by extending the `_OnReturnKey` method of the text box's `TextEntry` class with a method override.

## DECOUPLING DATA WITH METADATA

In SAS, data sources must be allocated before they can be read; this means users must know beforehand where data are located. Generally, users must also have the IT skills necessary to allocate data once they have found it. To insulate users from this concern, the application relies on technical metadata retrieved from a SAS dataset to supply the code necessary to access any data source. When the code is executed at run-time, all requested tables are allocated for the user transparently. As a result, users may spend more time querying data and less time looking for it (sample metadata for a single data source is shown in APPENDIX D).

## DE-CONSTRUCTING DATA FOR CONSISTENCY

Most data at the FDIC are organized by time-period – annual, quarterly, or monthly. Depending on the type of data, records belonging to a single entity, such as a bank, may be stored in separate flat files – one file per time-period – or spread horizontally across multiple database tables, where one table might contain records for all periods of time. To standardize the way data sources are read, the application uses the metadata file described in the previous section to identify the periodicity of records; this then allows all records to be read with the same logical constructs regardless of how many time-periods are stored in the data source. Gradually, each section of data is reassembled by time-period into the final output dataset. The process is completely transparent to users and provides a consistent framework for reading files that are not uniformly structured.

## USING MACROS TO FACILITATE DATA FILTERING

The application uses the Macro Language to facilitate data filtering. As with SQL, users request or filter rows by entering criteria inside a *Where* clause box; the traditional problem encountered when querying time-period based data is that additional criteria must be added to capture records from each specific period of time requested by the query. To alleviate this, the application relies on the aforementioned metadata file to populate the *Where* clause box with a default filtering statement; the default statement is designed to capture records from any number of periods (see Figure 6).

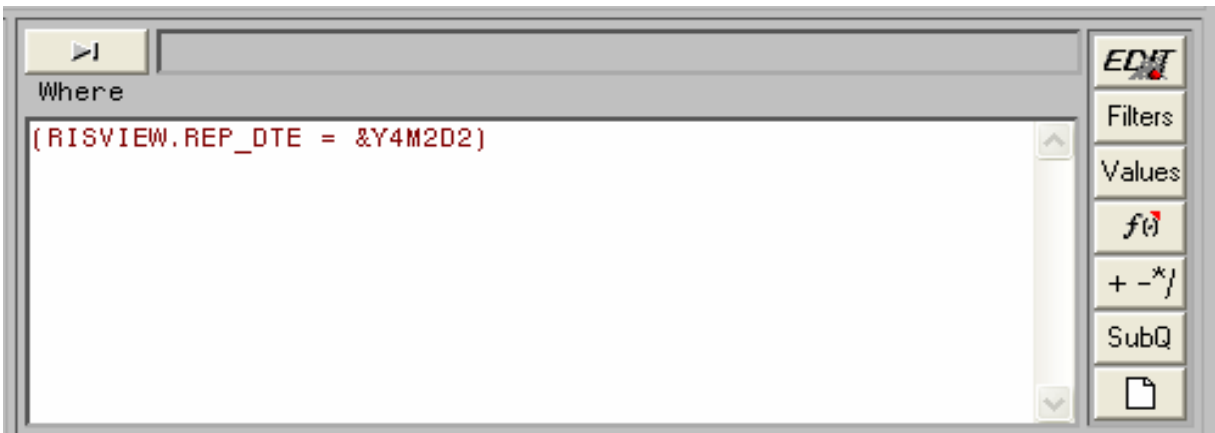


Figure 6. The figure shows the *Where* clause box located in the main query window. When a data source is selected, the application populates the *Where* clause box with default criteria sufficient for capturing records from any number of time-periods.

The process works this way: the *Where* clause references a macro variable called &Y4M2D2, which is declared by the application at run-time (shown in Figure 6, the name of the variable, &Y4M2D2 indicates an eight-digit time-period value, which is present in all data sources – where Y4 represents the four-digit year, M2 the two-digit month, and D2 the two-digit day). When the query is run, the application will place the *Where* box criteria inside an SQL procedure and encapsulate the SQL inside a SAS macro. The macro and SQL will execute once for each time-period listed in the *Selected periods* list box (see Figure 7).

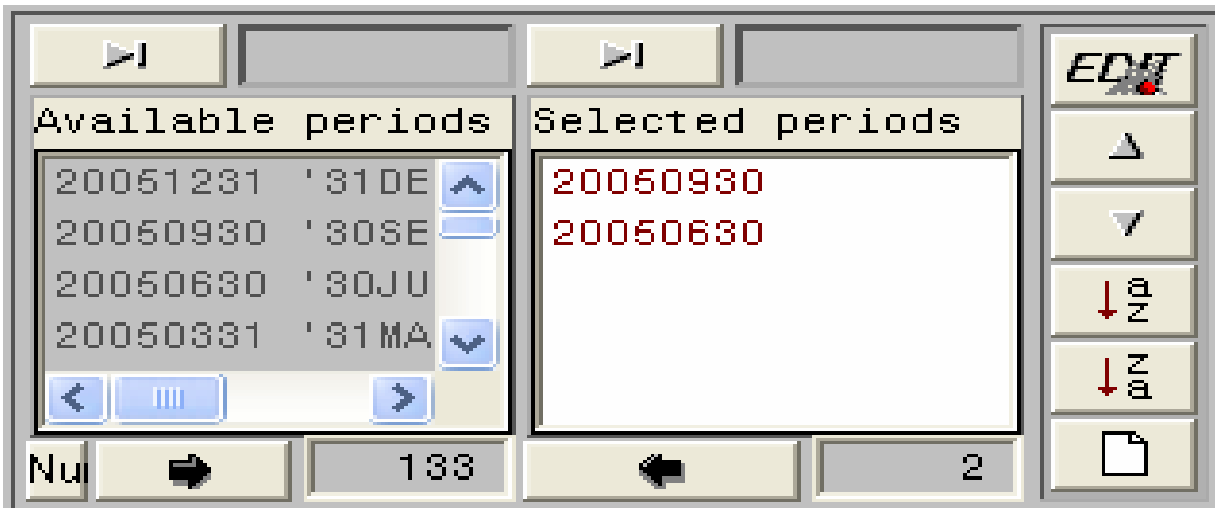


Figure 7. This figure shows the *Selected periods* list box (located inside the main query window) with two time-periods selected. At run-time, each time-period will be passed to an SQL procedure macro to capture the data for that period.

With each execution of the macro, one of the selected time-period values will be passed to the macro as a parameter, which in turn will cause the macro variable &Y4M2D2 to resolve to the selected time-period value. As the SQL executes, rows that meet the resolved criteria will be captured and appended to the final output dataset. Of course, users may place additional filter statements inside the *Where* box; however, the default *Where* clause is sufficient to capture all time-period based data without the need for additional filtering, even if all 133 time-periods are selected from the *Available periods* list box (see Figure 8).

```

* _____;
* Macro definition: "MAIN";
* _____;
%Macro MAIN(PERIOD=);
%SETDATES(DATEPARM=&PERIOD);
%Let COUNT = %Eval(&COUNT + 1);
Proc SQL;
Create Table _MAIN As
Select
RISVIEW.REP_DTE Label='Report Date' Format=YYMMDD10.0 As REP_DTE
,RISVIEW.BANK_ID Label='Certificate Number' Format=Z6.0 As BANK_ID
,RISVIEW.STATE Label='Fips State Alpha Code' Format=$2.0 As STATE
,RISVIEW.ASSETS Label='Total Assets' Format=COMMA12.0 As ASSETS
,&COUNT As __COUNT
From RIS&Y2M2..RISVIEW As RISVIEW
Where (RISVIEW.REP_DTE = &Y4M2D2)
;
Quit;
%RISOFF(&Y4M2D2);
* _____;
* Concatenate dataset _MAIN to dataset TEMP;
* _____;
Data TEMP;
%If (&COUNT LE 1) %Then %Do;
    Set _MAIN;
%End;
%Else %Do;
    Set TEMP _MAIN;
%End;
Run;
%Mend MAIN;
* _____;
* Execute macro "MAIN";
* _____;
%MAIN(PERIOD=20050930);
%MAIN(PERIOD=20050630);

```

Figure 8. This figure shows a section of a SAS program the application creates at run-time to execute a query. The program contains a SAS macro called %MAIN, which contains a PROC SQL. The macro executes once for each selected time-period, as shown by the two %MAIN(PERIOD=...) statements located at the bottom. Another macro, %SETDATES (not shown), parses the period value into the macro variable &Y4M2D2. Since &Y4M2D2 is referenced by the default *Where* clause, the resolved macro value completes the default criteria statement; data as of the requested time-period is then captured and appended to the final output dataset.

## USING MACROS TO CREATE TIME-SERIES VARIABLES

The application declares other macro variables at run-time besides &Y4M2D2. These variables (declared in a macro called %SETDATES) resolve to six or four-digit representations of each selected period value. Users may attach the macro variable names to the names of selected fields to create new time-series variables.

To illustrate, consider a query with the following selections listed inside the *Selected fields* list box.

```

RISVIEW.REP_DTE Label='Report Date' Format=YYMMDD10.0 As REP_DTE
RISVIEW.BANK_ID Label='Certificate Number' Format=Z6.0 As BANK_ID
RISVIEW.STATE Label='Fips State Alpha Code' Format=$2.0 As STATE
RISVIEW.ASSETS Label='Total Assets' Format=COMMA12.0 As ASSETS

```

These statements will become part of the SQL procedure macro that the application will execute at run-time. If executed for two time-periods, 20050930 (September 30th, 2005) and 20050630 (June 30<sup>th</sup>, 2005), the selections will produce the unadjusted output shown below; note how the records are appended and grouped by time-period, where each institution is listed once within a quarterly period of time (the output is also abridged for clarity).

| <u>REP_DATE</u> | <u>BANK_ID</u> | <u>STATE</u> | <u>ASSETS</u> |
|-----------------|----------------|--------------|---------------|
| 20050930        | 260022         | CA           | 4,550,100     |
| 20050930        | 477064         | NY           | 3,130,000     |
| 20050630        | 260022         | CA           | 4,446,000     |
| 20050630        | 477064         | NY           | 2,990,500     |
| 20050630        | 532200         | SC           | 33,000        |

Suppose the user now wishes to create a separate Total Assets time-series variable for each selected period so that trend analysis can be performed with the data. To do this, the user simply attaches one of the time-period macro variables to the end of the Total Assets field name, ASSETS, as shown in Figures 9 and 10 (in this case the declared macro variable &Y2M2 is attached to the field name – where Y2 represents the two-digit year and M2 the two-digit month).

After the name of the field is changed, the Total Assets selection in the *Selected fields* list box will display as:

```

RISVIEW.ASSETS Label='Total Assets' Format=COMMA12.0 As ASSET_&Y2M2

```

...and when executed, two new time-series variables – ASSET\_0509 and ASSET\_0906 – will be created from the single Total Assets field entry, as shown in the output below.

| <u>REP_DATE</u> | <u>BANK_ID</u> | <u>STATE</u> | <u>ASSETS_0509</u> | <u>ASSETS_0506</u> |
|-----------------|----------------|--------------|--------------------|--------------------|
| 20050930        | 260022         | CA           | 4,550,100          | .                  |
| 20050930        | 477064         | NY           | 3,130,000          | .                  |
| 20050630        | 260022         | CA           | .                  | 4,446,000          |
| 20050630        | 477064         | NY           | .                  | 2,990,500          |
| 20050630        | 532200         | SC           | .                  | 33,000             |

The total number of time-series variables created will depend on the number of periods listed in the *Selected periods* list box. As more periods are selected, more time-series variables will be created, although the number of field selections will not change.

**Edit specifications for the selected field (change any that apply)**

Original field name/function

As

Label

SAS informat  ... SAS format  ...

Length  ...

Ok  Cancel

Figure 9. Users may press a button located beside the *Selected fields* list box to display a window where they may change the name, label, length, and format of any field.

**Edit specifications for the selected field (change any that apply)**

Original field name/function

As

Label

SAS informat  ... SAS format  ...

Length  ...

Ok  Cancel

Figure 10. The name of the Total Assets field (shown circled) is changed from ASSETS to ASSETS\_&Y2M2. At run-time, this will cause a new Total Assets time-series variable to be created for each period value listed in the *Selected periods* list box. Only one field entry is needed to create n-number of Total Assets variables.

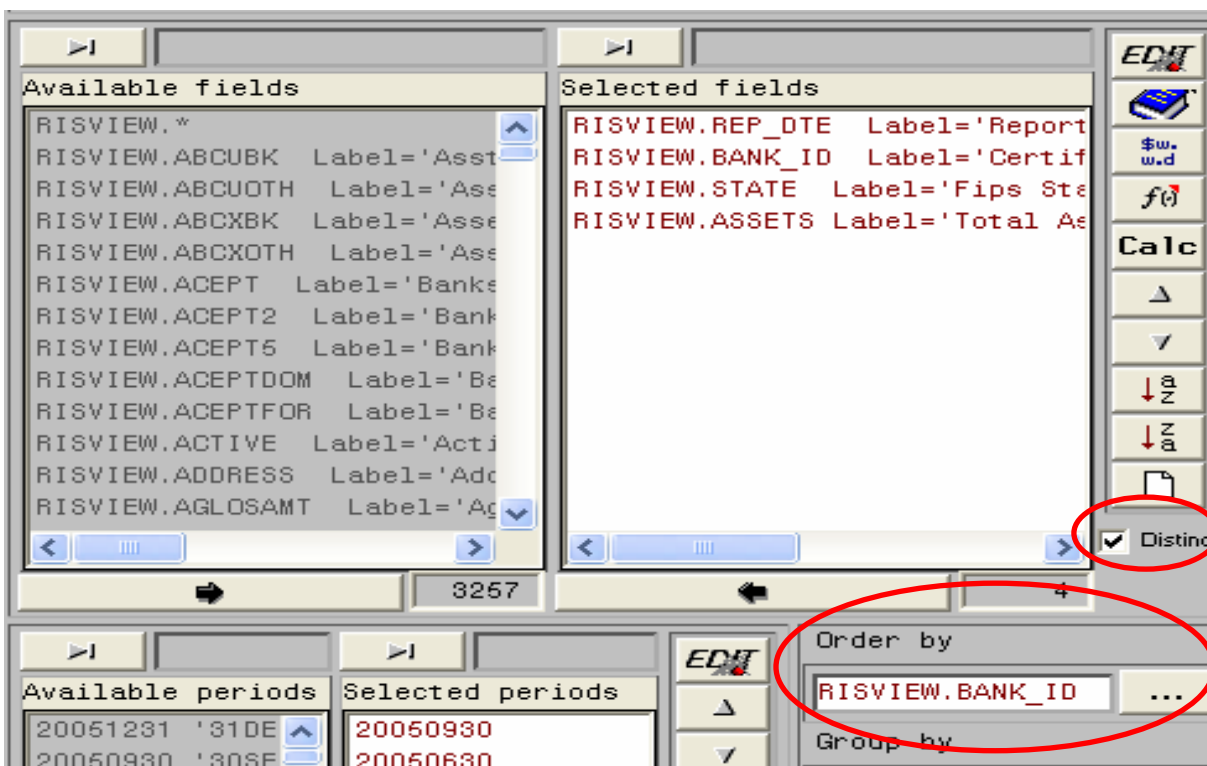
## CREATING OUTPUT WITH BUSINESS INTELLIGENCE

Since the application creates output by appending data from one time-period on top of another, the output shown in the previous section appears disjointed. The appearance could be improved,

however, if each institution displayed only once and if the time-series values for each bank moved into the same row.

If we were to write a SAS program to accomplish this, each group of time-period data would need to be merged instead of appended; and merged specifically by a key variable such as BANK\_ID. This is exactly what the application will do if the key variable BANK\_ID is placed inside an *Order by* box and if a *Distinct* option is checked (both options are located inside the main query window and are shown circled in Figure 11 below). The application will then produce the following output, which is much easier to read and better suited for trend analysis.

| <u>REP_DATE</u> | <u>BANK_ID</u> | <u>STATE</u> | <u>ASSETS_0509</u> | <u>ASSETS_0506</u> |
|-----------------|----------------|--------------|--------------------|--------------------|
| 20050930        | 260022         | CA           | 4,550,100          | 4,446,000          |
| 20050930        | 477064         | NY           | 3,130,000          | 2,990,500          |
| 20050630        | 532200         | SC           | .                  | 33,000             |



**Figure 11.** When the *Distinct* checkbox and *Order by* box (both circled) are used, time-series data will merge by the *Order by* variable, creating output where each bank displays only once and where time-series values for that bank display side-by-side in the same row. If the *Distinct* checkbox is not checked when an *Order by* variable is assigned, then output will be appended as before, but will only be sorted by the *Order by* variable.

## LAGGING DATA FOR GROWTH RATE CALCULATIONS

FDIC staff frequently uses growth rates to measure the financial health of member banks. To meet this requirement, the application's *Define calculated fields* window allows users to calculate growth rates, deltas, or almost any kind of calculated value that may be needed. Each new variable may be derived within the main SQL procedure or calculated separately in an independent SQL step, where each step may have its own time-period selection and *Where* clause. In case of the latter, the application will merge the calculated data so that information from each SQL step displays side-by-side in the query output (see Figure 12).

Time-period/step of definition  
☐ Selected periods {C} ☒ Lag period {L} ☐ Query {Q} ☐ Mixed {M} 1 Of 1

Name: LAG\_ASSETS Label: Lag period calculated field

SAS informat: SAS format: Length:

Definition: ASSETS

Options (for "Lag period {L}" or "Query {Q}")

Lag period: 20050630 By: BANK\_ID Group by:

Join type: Left Having (group filter): Where: [RISVIEW.REP\_DTE = &Y4M2D2]

**Figure 12.** Users may press a button located beside the *Selected fields* list box to display a window where they may define calculated fields. In the *Define calculated fields* window, new variables may be derived in the main SQL procedure or in separate procedures that contain their own time-period selection and *Where* clause.

One application of this feature is to create variables calculated as of a time-period that lags data captured from the main query by some interval. For example, consider the query that captures data as of the time-periods 20050930 (September 30<sup>th</sup>, 2005) and 20050630 (June 30<sup>th</sup>, 2005). Suppose also that the calculated field LAG\_ASSETS is defined to display Total Assets as of the relative lagged period represented by the value &Y4M2D2-1 (where -1 indicates a time-period lagged one quarter prior to any time-period in the *Selected periods* list box – or in this case, since the *Selected periods* list box contains two time-periods, 20050930 and 20050630 – and assuming the year is divided into March, June, September, and December quarters – then the run-time lagged period values will resolve to 20050630 (June 30<sup>th</sup>, 2005) and 20050331 (March 31<sup>st</sup>, 2005). See Figure 13.

Assuming the unadjusted (non-time-series) selections introduced in the **USING MACROS TO CREATE TIME-SERIES VARIABLES** section, this query will produce the output shown below. Note how the values of the field LAG\_ASSETS precede the ASSET values relative to REP\_DATE by one quarter (indicated more clearly by the data in the first and third rows). No matter what time-period is selected from the *Available periods* list box, the values in the LAG\_ASSETS field will lag this data by the interval assigned in the LAG\_ASSETS calculation.

| REP_DATE | BANK_ID | STATE | ASSETS    | LAG_ASSETS |
|----------|---------|-------|-----------|------------|
| 20050930 | 260022  | CA    | 4,550,100 | 4,446,000  |
| 20050630 | 260022  | CA    | 4,446,000 | 4,202,700  |
| 20050930 | 477064  | NY    | 3,130,000 | 2,990,500  |
| 20050630 | 477064  | NY    | 2,990,500 | 2,633,000  |
| 20050630 | 532200  | SC    | 33,000    | 58,540     |

Time-period/step of definition  
☐ Selected periods {C} ☒ Lag period {L} ☐ Query {Q} ☐ Mixed {M} 1 Of 1

Name: LAG\_ASSETS Label: Lag period calculated field

SAS informat: SAS format: Length:

Definition: ASSETS

Options (for "Lag period {L}" or "Query {Q}")

Lag period: &Y4M2D2-1 By: BANK\_ID Group by:

Join type: Left Having (group filter): Where: [RISVIEW.REP\_DTE = &Y4M2D2]

**Figure 13.** The *Define calculated fields* window is shown with a *Lag Period* value assigned as &Y4M2D2-1 (circled). At run-time, the derived lagged value will precede any time-period value in the *Selected periods* list box by one quarter. The lagged data will then merge with non-lagged data by *BANK\_ID* (also shown circled) so lagged and non-lagged data will appear side-by-side in the query output.

Once a lag-period variable is defined this way, then other complex formulas such as growth rates may be defined and calculated. For example, consider the formula for the variable called *GROWTH* that calculates the growth rate of Total Assets between the lagged and non-lagged time-periods (see Figure 14).

At run-time, this calculation will create the following output, where the growth rates of member banks are calculated on-the-fly across time.

| REP_DATE | BANK_ID | STATE | ASSETS    | LAG_ASSETS | GROWTH |
|----------|---------|-------|-----------|------------|--------|
| 20050930 | 260022  | CA    | 4,550,100 | 4,446,000  | 2.34   |
| 20050630 | 260022  | CA    | 4,446,000 | 4,202,700  | 5.79   |
| 20050930 | 477064  | NY    | 3,130,000 | 2,990,500  | 4.66   |
| 20050630 | 477064  | NY    | 2,990,500 | 2,633,000  | 13.58  |
| 20050630 | 532200  | SC    | 33,000    | 58,540     | -43.63 |

## CONCLUSION

The challenge that system developers face is designing applications, which function in environments with opposing problem domains – applications must have sophisticated features yet be easy to use; they must satisfy users with different levels of IT experience, yet offer a simple and consistent framework for processing data. SAS gives users considerable problem-solving ability; but without custom interfaces or stored processes, users must master more than a few prerequisite IT skills. This paper demonstrated how one application attempted to fulfill its main objective, that is, provide a data querying interface; and to solve opposing problem domains by using SQL concepts as a design model; by using composite classes and event handling to maximize functionality; by using metadata to insulate users from repetitive and time-consuming processes; and lastly, by integrating SAS products in ways that offer users true BI capabilities.

**Figure 14.** The *Define calculated fields* window shows the calculation for a new variable called **GROWTH**. The variable will calculate the growth-rate between the lagged and non-lagged Total Assets variables.

## REFERENCES

Loshin, David Business Intelligence: The Savvy Manager's Guide San Francisco, CA: Morgan Kaufmann Publishers, 2003.

Cooper, Allan & Reinmann, Robert About Face 2.0: The Essentials Of Interaction Design Indianapolis, IN: Wiley Publishing, Inc., 2003.

SAS Institute Inc. SAS Language Reference: Dictionary, Version 8. Cary, NC: SAS Publishing, 2000.

SAS Institute Inc. SAS Macro Language: Reference, Version 8. Cary, NC: SAS Publishing, 1999.

SAS Institute Inc. SAS Procedures Guide, Version 8. Cary, NC: SAS Publishing, 1999.

SAS Institute Inc. SAS SQL Procedure User's Guide, Version 8. Cary, NC: SAS Publishing, 2000.

## ACKNOWLEDEGMENTS

I would like to thank my colleagues Michael Malarkey, whose object-oriented development (particularly with the dual list box class) inspired some of the concepts discussed in this paper; and Pat Relich, without whose support, the project could not have been possible.

## CONTACT INFORMATION

Brian Deitch  
[www.bdeitch@fdic.gov](mailto:bdeitch@fdic.gov)  
 Federal Deposit Insurance Corporation  
 550 17th Street, NW  
 Washington, DC 20429-0002

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

## APPENDIX A

The following Call PutList() output shows the SCL lists the application creates to store query selections. Selections are stored in SCL lists, where each sublist contains the items that populate a given screen object. For example, the sublist L\_FIELDS\_SELECTED stores the field names that populate the *Selected fields* list box shown in Figure 3.

```
L_SYSX( SYSTITLE='FDIC DATAbank SQL/SAS query engine v1.2.1152'
        QUERY_NAME='QUERY1'
        QUERY_DESC='QUERY EXAMPLE'
        L_DATA_SELECTED=( 'RIS.RISVIEW (MAINFRAME) joined
                           institution data.'
                           ) [39295]
        L_QUERY_TYPE=( 'Equijoin'
                        ) [39903]
        L_FIELDS_CALCULATED=() [39861]
        L_FIELDS_SELECTED=( 'RISVIEW.REP_DTE Label='Report Date'
                             Format=YYMMDD10.0 As REP_DTE'
                             'RISVIEW.BANK_ID Label='Certificate
                             Number' Format=Z6.0 As BANK_ID'
                             'RISVIEW.STATE Label='Fips State Alpha
                             Code' Format=$2.0 As STATE'
                             'RISVIEW.ASSETS Label='Total Assets'
                             Format=COMMA12.0 As ASSETS'
                             ) [39863]
        FIELDS_DISTINCT='Yes'
        L_GROUPBY_FORMATTED=() [39865]
        L_GROUPBY_SELECTED=() [39867]
        L_HAVING_FORMATTED=() [39869]
        L_HAVING_SELECTED=() [39871]
        L_ORDERBY_FORMATTED=( 'RISVIEW.BANK_ID'
                               ) [39881]
        L_ORDERBY_SELECTED=() [39883]
        OUTPUT_BATCHJOB='No'
        OUTPUT_CENTER='Yes'
        OUTPUT_DOWNLOAD='No'
        OUTPUT_FOOTER='No'
        OUTPUT_HEADER='No'
        OUTPUT_LABELS='No'
        OUTPUT_OBS='Yes'
        OUTPUT_PROC_PRINT='Yes'
        L_OUTPUT_BATCHJOB_DESC=() [39885]
        L_OUTPUT_FILENAME=( 'WORK.TEMP'
                              ) [39889]
        L_OUTPUT_FILETYPE=( 'SAS DATA SET'
                              ) [39891]
        L_OUTPUT_FOOTER=() [39893]
        L_OUTPUT_HEADER=() [39895]
        L_OUTPUT_OBS=( 'MAX'
                        ) [39897]
        L_PERIODS_SELECTED=( '20050930'
                              '20050630'
                              ) [39901]
        L_TOOLS_IN_QUERY=() [39921]
        L_TOOLS_SELECTED=() [39923]
        L_WHERE_CLAUSE=( '(RISVIEW.REP_DTE = &Y4M2D2)'
                          ) [39925]
) [39905]
```

## APPENDIX B

```

/*****
/* METHOD: ONKEY_TE_FINDTEXT1
/* CALLED FROM: _SELF_
/* CALLS: N/A
/* DESCRIPTION: PROCESS THE KEYSTROKE EVENT IN LIST BOX 1. THIS
/* METHOD OVERRIDES THE INHERITED _ONVALUECHANGED
/* METHOD. THE METHOD WILL RUN EACH TIME THE USER
/* ENTERS A LETTER INTO THE OBJECT. LIST BOX 1 WILL
/* BE SEARCHED. WHEN A MATCH IS FOUND, THE MATCHED
/* ITEM WILL SCROLL TO THE TOP OF THE LIST BOX.
*****/
ONKEY_TE_FINDTEXT1:
    PUBLIC METHOD;
/*
/* DECLARE VARIABLES.
/*
    DCL NUM      I
                RC,
    OBJECT W_LB_BOX1;
/*
/* GET THE _SELF_.OWNERID OBJECTS.
/*
_SELF_.OWNERID._GETWIDGET( 'LB_BOX1',W_LB_BOX1);
/*
/* IF THE TEXT LENGTH IS GREATER THAN 0, THEN PERFORM SEARCH.
/*
IF (LENGTH(_SELF_.TEXT)) THEN DO;
    RC = 0;
    DO I=1 TO W_LB_BOX1.ROWS;
/*
/* IF THE FIRST CHARACTER OF THE TEXT STRING IN THE W_TE_FINDTEXT1
/* OBJECT IS A SPACE, THEN PERFORM A COMPLETE WORD SEARCH.
/*
        IF (SUBSTR(_SELF_.TEXT,1,1) = ' ') THEN DO;
            IF (SCAN(UPCASE(GETITEMC(W_LB_BOX1.ITEMS,I)),1,' ') =
                LEFT(TRIM(UPCASE(_SELF_.TEXT)))) THEN
                RC = I;
        END;
/*
/* OTHERWISE, PERFORM A STRING SEARCH.
/*
        ELSE DO;
            IF (INDEX(UPCASE(GETITEMC(W_LB_BOX1.ITEMS,I)),
                LEFT(TRIM(UPCASE(_SELF_.TEXT)))) THEN
                RC = I;
            END;
            IF (RC) THEN LEAVE;
        END;
/*
/* IF AN ITEM IS FOUND, SCROLL THE ITEM TO THE TOP OF LIST BOX 2 AND
/* SELECT THE ITEM.
/*
        IF (RC) THEN DO;
            W_LB_BOX1.TOPROW = RC;
            W_LB_BOX1.SELECTEDINDEX = RC;
        END;
        END;
        ELSE W_LB_BOX1._DESELECTALL();
    ENDMETHOD;

```

## APPENDIX C

```

/*+++++++*/
/* METHOD:  ONRETURNKEY_LB_BOX1 */
/* CALLED FROM:  _SELF_ */
/* CALLS:  N/A */
/* DESCRIPTION:  PROCESS THE RETURN KEY EVENT IN LIST BOX 1.  THIS */
/* METHOD OVERRIDES THE INHERITED _ONRETURNKEY METHOD. */
/* IF AN ITEM IN LIST BOX 1 IS SELECTED, AND THE */
/* RETURN KEY IS PRESSED ON THE COMPUTER KEYBOARD, */
/* THEN MOVE THE ITEM TO LIST BOX 2.  THEN PERFORM A */
/* LOOKUP SEARCH FOR THE NEXT ITEM IN LIST BOX 1. */
/*+++++++*/
ONRETURNKEY_LB_BOX1:
  PUBLIC METHOD;
/*
/* DECLARE VARIABLES.
/*
DCL LIST  LIST2={},
  NUM  I,
  OBJECT W_LB_BOX1
        W_LB_BOX2
        W_TE_LB_BOX1_FINDTEXT
        W_TE_LB_BOX2_NUMBER;
/*
/* GET THE _SELF_.OWNERID OBJECTS.
/*
_SELF_.OWNERID._GETWIDGET('LB_BOX1',W_LB_BOX1);
_SELF_.OWNERID._GETWIDGET('LB_BOX2',W_LB_BOX2);
_SELF_.OWNERID._GETWIDGET('TE_LB_BOX1_FINDTEXT',W_TE_LB_BOX1_FINDTEXT);
_SELF_.OWNERID._GETWIDGET('TE_LB_BOX2_NUMBER',W_TE_LB_BOX2_NUMBER);
/*
/* IF THE SELECTED ITEMS ARE BLANK, THEN RETURN.
/*
IF (LISTLEN(W_LB_BOX1.SELECTEDITEMS)) THEN;
ELSE RETURN;
/*
/* ERASE THE CONTENTS OF THE MESSAGE SCL LIST.
/*
IF (CLEARLIST(_SELF_.OWNERID.L_MSG,'Y') = 0) THEN;
/*
/* COPY THE LIST ITEMS IN W_LB_BOX2 TO SCL LIST LIST2.
/*
LIST2 = COPYLIST(W_LB_BOX2.ITEMS,'Y',LIST2);
/*
/* IF AN ITEM IN LIST BOX 1 IS SELECTED, AND THE RETURN KEY IS
/* PRESSED, THEN MOVE THE ITEM TO LIST BOX 2.
/*
IF (W_LB_BOX1.SELECTEDCOUNT) THEN DO;
  IF (UPCASE(W_LB_BOX1.SELECTIONMODE) = 'SINGLE SELECTION') THEN DO;
    IF (CLEARLIST(LIST2,'Y') = 0) THEN;
      LIST2 = INSERTC(LIST2,GETITEMC(W_LB_BOX1.SELECTEDITEMS,1),-1);
  END;
  ELSE DO I=1 TO W_LB_BOX1.SELECTEDCOUNT;
    LIST2 = INSERTC(LIST2,GETITEMC(W_LB_BOX1.SELECTEDITEMS,I),-1);
  END;
/*
/* POPULATE THE MESSAGE SCL LIST.
/*
_SELF_.OWNERID.L_MSG = INSERTC(_SELF_.OWNERID.L_MSG,'One or '||
                             'more items have been selected '||
                             'or updated.',-1);
END;
/*
/* SET VARIOUS LIST BOX ATTRIBUTES.
/*

```

```

W_LB_BOX1._DESELECTALL();
W_LB_BOX2._DESELECTALL();
W_LB_BOX1.TOPROW = W_LB_BOX1.TOPROW + 1;
W_TE_LB_BOX1_FINDTEXT.TEXT = '';
IF (CLEARLIST(W_LB_BOX2.ITEMS, 'Y') = 0) THEN;
W_LB_BOX2.ITEMS = COPYLIST(LIST2, 'Y', W_LB_BOX2.ITEMS);
IF (DELLIST(LIST2) = 0) THEN;
IF (W_LB_BOX2.ROWS = 0) THEN W_TE_LB_BOX2_NUMBER.TEXT = '';
ELSE W_TE_LB_BOX2_NUMBER.TEXT = PUT(W_LB_BOX2.ROWS, 5.);
/* _____ */
/* ASSIGN THE TOP ROW OF THE LIST BOX. _____ */
/* _____ */
IF (W_LB_BOX2.ROWS) THEN DO;
    RC = W_LB_BOX2.HEIGHT - 6;
    RC = INT(RC);
    IF (W_LB_BOX2.ROWS > RC) THEN
        W_LB_BOX2.TOPROW = W_LB_BOX2.ROWS - (RC - 1);
    ELSE W_LB_BOX2.TOPROW = 1;
END;
ELSE W_LB_BOX2.TOPROW = 1;
ENDMETHOD;

```

## APPENDIX D

The following shows the technical metadata used by the application for a single data source selection.

|                   |                                                                     |
|-------------------|---------------------------------------------------------------------|
| DATA_ALIAS:       | RISVIEW                                                             |
| DATA_BASE:        | RIS                                                                 |
| DATA_DESCRIPTION: | (RISVIEW) joined institution data.                                  |
| DATA_DICTIONARY:  | <a href="http://www2.fdic.gov/dict/">http://www2.fdic.gov/dict/</a> |
| DATA_ENVIRONMENT: | MAINFRAME                                                           |
| DATA_VIEW:        | RISVIEW                                                             |
| DATASET_FIELDS:   | RISVIEW                                                             |
| DATE_END:         | 20050930                                                            |
| DATE_FORMAT:      | &Y4M2D2                                                             |
| DATE_PERIODICITY: | QUARTERLY                                                           |
| DATE_START:       | 19721231                                                            |
| STMT_DFLT_WHERE:  | (RISVIEW.REP_DTE = &Y4M2D2)                                         |
| STMT_LIBON:       | %RISON(&Y2M2);                                                      |
| STMT_LIBOFF:      | %RISOFF(&Y2M2);                                                     |
| STMT_LIBREF:      | RISVIEW.                                                            |
| STMT_SASNAME:     | RIS&Y2M2..RISVIEW                                                   |

## APPENDIX E – THE GENERATED SAS PROGRAM THAT CALCULATES LAG\_ASSETS

The following SAS program is generated by the application to merge Total assets (ASSETS) with lagged Total assets (LAG\_ASSETS) by one quarter relative to the time-periods 20050930 and 20050630. The SAS macro %SETDATES declares all macro variables, including &Y4M2D2 and &Y2M2. The program code which captures Total assets is placed inside the macro %MAIN, which is executed twice – once for each time-period. A temporary dataset is created with each macro execution (and each dataset will include the selected variables REP\_DTE, BANK\_ID, STATE, ASSETS, and an execution-sequence variable called \_\_\_COUNT that the application creates to merge lagged and non-lagged data).

The application uses an algorithm (not shown) to determine the lagged time-period values, which in this case will resolve to 20050630 and 20050331. The program code which captures lagged Total assets is then placed inside a macro called %\_LGM1 and executed once for each lagged time-period. Temporary datasets are created during each %\_LGM1 execution and contain the same variables created in the macro %MAIN described above.

Each temporary dataset of lagged and non-lagged data is then merged by the variable BANK\_ID (the By variable assigned in the *Define calculated fields* window) and by the execution-sequence variable \_\_COUNT. After the merge, each dataset is appended to the final output dataset called WORK.TEMP (but without the variable \_\_COUNT).

```

* _____;
* Generated by: "FDIC DATAbank SQL/SAS query engine v1.2.1152";
* Query name: "Untitled";
* Query desc: "User query";
* _____;
OPTIONS NOCAPS NOTES NOMPRINT NOMRECALL NOSYMBOLGEN OBS=MAX; RUN;
* _____;
* Macro definition: "SETDATES";
* _____;
%Macro SETDATES(DATEPARM);
%Global D2
      M2
      Y2
      Y2M2
      Y4
      Y4M2
      Y4M2D2;
%If (&DATEPARM = 0) %Then %Goto Macexit;
%Let M2 = %Substr(&DATEPARM,5,2);
%Let Y2 = %Substr(&DATEPARM,3,2);
%Let Y2M2 = %Substr(&DATEPARM,3,4);
%Let Y4 = %Substr(&DATEPARM,1,4);
%If ("&M2" = "03") %Then %Let D2 = 31;
%If ("&M2" = "06") %Then %Let D2 = 30;
%If ("&M2" = "09") %Then %Let D2 = 30;
%If ("&M2" = "12") %Then %Let D2 = 31;
%Let Y4M2 = &Y4&M2;
%Let Y4M2D2 = &Y4M2&D2;
%Macexit: %Mend SETDATES;
* _____;
* End macro defintion: SETDATES;
* _____;
* Setup the COUNT macro variable;
* _____;
%Global COUNT;
%Let COUNT = 0;
* _____;
* Macro definition: "MAIN";
* _____;
%Macro MAIN(PERIOD=);
%SETDATES(DATEPARM=&PERIOD);
%Let COUNT = %Eval(&COUNT + 1);
* _____;
* Macro definition: "_LGM1";
* _____;
%Macro _LGM1(PERIOD=, OUTNAME=, COUNT=);
%SETDATES(DATEPARM=&PERIOD);
%RISON(&Y4M2D2);
Proc SQL;
Create Table &OUTNAME As
Select Distinct
  BANK_ID
, RISVIEW.REP_DTE Label='Report Date' Format=YYMMDD10.0 As REP_DTE
, RISVIEW.BANK_ID Label='Certificate Number' Format=Z6.0 As BANK_ID
, RISVIEW.STATE Label='Fips State Alpha Code' Format=$2.0 As STATE
, RISVIEW.ASSETS Label='Total Assets' Format=COMMA12.0 As ASSETS
, &COUNT As __COUNT
, ASSETS
  As ASSETS_LAG
  Label='Lag period calculated field'

```

```

From RIS&Y2M2..RISVIEW As RISVIEW
Where (RISVIEW.CALLYMD = &Y4M2D2)
  Order by BANK_ID;
Quit
%RISOFF(&Y4M2D2);
%Mend _LGM1;
* _____;
* End macro definition:  "_LGM1";
* _____;
* Execute lag period macros;
* _____;
%If (&COUNT = 1) %Then %Do;
  %_LGM1(PERIOD=20050630,OUTNAME=_LGM_20050630_1,COUNT=1);
  %_LGM1(PERIOD=20050331,OUTNAME=_LGM_20050331_2,COUNT=2);
%End;
* _____;
* Re-assign the time-period macro variables;
* _____;
%SETDATES(DATEPARM=&PERIOD);
* _____;
* Allocate primary data source(s);
* _____;
%RISON(&Y4M2D2);
* _____;
* Process the primary query;
* _____;
Proc SQL;
Create Table _MAIN As
Select
RISVIEW.REP_DTE Label='Report Date' Format=YYMMDD10.0 As REP_DTE
,RISVIEW.BANK_ID Label='Certificate Number' Format=Z6.0 As BANK_ID
,RISVIEW.STATE Label='Fips State Alpha Code' Format=$2.0 As STATE
,RISVIEW.ASSETS Label='Total Assets' Format=COMMA12.0 As ASSETS
,&COUNT As ____COUNT
From RIS&Y2M2..RISVIEW As RISVIEW
Where (RISVIEW.CALLYMD = &Y4M2D2);
Quit;
* _____;
* De-allocate primary data sources;
* _____;
%RISOFF(&Y4M2D2);
* _____;
* Join all lag period datasets to the primary data source;
* _____;
%If (&COUNT GE 1) AND (&COUNT LE 2) %Then %Do;
Proc Sort Data=_MAIN Tagsort;
  By BANK_ID;
Run;
Data _MAIN;
  Merge _MAIN(In=A)
        _LGM_20050630_1(In=B)
        _LGM_20050331_2(In=B)
        _LGM_20050630_1(In=B)
        _MAIN(In=A);
  By BANK_ID
     ____COUNT;
  If A;
Run;
%End;
* _____;
* Perform house-cleaning;
* _____;
%If (&COUNT = 2) %Then %Do;
  Proc Datasets;
    Delete _LGM_20050630_1
          _LGM_20050331_2;

```

```

Quit;
%End;
Data _MAIN;
    Set _MAIN;
    Drop ____COUNT;
Run;
%If (&COUNT = 1) %Then %Do;
    * Perform house-cleaning;
Proc Datasets;
    Delete TEMP;
Quit;
%End;
* _____;
* Concatenate dataset _MAIN to dataset TEMP;
* _____;
Data TEMP;
%If (&COUNT LE 1) %Then %Do;
    Set _MAIN;
%End;
%Else %Do;
    Set TEMP _MAIN;
%End;
Run;
    * Perform house-cleaning;
Proc Datasets;
    Delete _MAIN;
Quit;
%Mend MAIN;
* _____;
* End macro definition: "MAIN";
* _____;
* Execute macro "MAIN";
* _____;
%MAIN(PERIOD=20050930);
%MAIN(PERIOD=20050630);
* _____;
* "Proc print" the output;
* _____;
Options Center Obs=MAX;
Title;
Footnote;
Proc Print Data=TEMP;
Run;
* _____;
* Create output dataset;
* _____;
Data WORK.TEMP;
    Set TEMP;
Run;

```

\*\*\*\*\*