Paper 417-2013

Update, Insert, and Carry-Forward Operations in Database Tables Using SAS® Enterprise Guide®

Thomas E. Billings, Union Bank, San Francisco, California
Sreenivas Mullagiri, iGATE, Fremont, California

Abstract

You want to use SAS® Enterprise Guide® to simulate database logic that includes any of: update, insert, carry-forward operations on old, changed, or new rows between two data sets, to create a new master data set. However, the Query Builder Task GUI does not have an Update/Insert option. Methods for simple types of update, insert, and/or carry-forward operations are described and illustrated using small data sets. First, we review Base SAS® methods, including DATA step and PROC SQL code. Then, two GUI-only/Task-based methods are described: one based on the Sort Data Task GUI; the other on the Query Builder Task GUI. The issue of whether integrity constraints are preserved is also discussed.

Introduction

In some organizations, SAS Enterprise Guide is being used by individuals who have extensive SQL experience but who are new to SAS and have little (or no) SAS programming experience or training. SAS Enterprise Guide may be used in conjunction with and in support of other SAS Solutions, e.g., SAS Data Integration Studio. In particular, SAS Enterprise Guide may be used for reporting and analyses of database-driven systems. Under these circumstances, analysts who have SQL experience but little or no SAS programming experience may need to perform update, insert, and carry-forward operations on two related data sets.

This paper describes simple approaches in SAS Enterprise Guide to accomplish these operations. The approaches discussed here can be emulated by users with little or no SAS programming skill, and can be done using the SAS Enterprise Guide Tasks. A number of related topics are also discussed.

Context and terminology

To avoid misunderstanding and for clarity, it is appropriate to specify the context of this paper and define the terms used. We have two data sets, let's call them Base_file and Revision_file. They have similar structure, i.e., they share a set of core variables with the same names, same data types, and their other attributes are the same or conformable. Base_file is supposed to include historical data and/or serve as an archive file.

The current version of Base_file was created at some point in the past. Revision_file is derived using Base_file and/or is derived from the same sources, and it is an updated version of Base_file or includes updates to Base_file. The objective is to use the data in Revision_file to create a new version of Base_file. Comparing the rows in the two tables using a primary key or a set of variables that uniquely identifies each row, we may observe that:

- Base_file has a set of rows that do not appear in Revision _file; we want these rows to appear in the new version of Base_file so they are *carry-forward rows* (i.e., "old" rows to carry-forward into new version),
- Revision_file has a set of rows that do not appear in Base _file; we want these rows to appear in the new version of Base_file so they are *insert rows* (i.e., "new" rows to be inserted in the new version),
- There is a set of rows that is common between the two files; these are *update rows,* and whether to pull forward the old version of the rows (as-is, from Base_file), use the rows in the new version (as-is, from Revision_file), or use some hybrid/transformation of the rows, will depend on the requirements of the program.

The transformations involved in updating rows can be complex; however, the update examples provided here are relatively simple. The next section defines some simple data sets and uses those to illustrate update, insert, and carry-forward operations in Base SAS and SAS Enterprise Guide. Database tables often contain integrity constraints, and whether such constraints are preserved or not in the operations illustrated is also reviewed here.

Example/test base and revision data sets

Two data sets, base and revisionfl, were created using SAS code (DATA step). These data sets are then copied, and the copies are assigned given different names, specifically basefl_* and revisionfl_* where * is unique for each example. Integrity constraints are then added to each new test file using PROC DATASETS or PROC SQL. This is done - all in Base SAS programming - to provide test files for each method of interest.

The reasons the example data sets were created in Base SAS code rather than using SAS Enterprise Guide Tasks are:

1. It keeps the process flow threads separate and “clean” for each method in the project, i.e., no multiple, crossed links to data sets that are used across multiple threads,
2. Integrity constraints cannot be assigned or managed using Tasks (a limitation that may be important to some database analysts). Of course, integrity constraints for files can be added, modified, or removed in SAS Enterprise Guide by writing and including the relevant SAS program code in projects.

The copies of the data sets created, with integrity constraints added are used below to illustrate the methods for update, insert, and carry-forward. The example data sets are as follows.

File: basefl_* (all versions of the base file)

Obs	keyvar	nvar1	nvar2	cvar
1	1	1	2	Los_Angeles
2	2	3	4	San_Diego
3	3	5	6	San_Francisco
4	4	7	8	Berkeley
5	5	11	10	San_Jose
6	6	13	12	Irvine
7	7	17	14	Long_Beach
8	8	19	16	Pasadena

File: revisionfl_* (all versions of the revision file)

Obs	keyvar	nvar1	nvar2	cvar
1	2	6	4	Irvine
2	4	14	8	San_Diego
3	6	26	12	Long_Beach
4	8	38	16	Pasadena
5	9	46	18	Sacramento
6	10	59	20	Fresno

In the example files, keyvar is a numeric key variable with values = 1,2,3,…, and nvar1, nvar2 are numeric variables. The variable cvar is a character variable, containing the names of select cities in California, USA. There are some differences in the test file variables between the 2 data sets for the same value of keyvar. The following integrity constraints were applied to the basefl_* data set (for all methods) and also revisonfl_* for some methods. These constraints were applied right after file creation and before the method processing:

Obs	Name	Type	Recreate
1	_NM0001_	Not Null	ic create Not Null (cvar);
2	primkey	Primary Key	ic create primkey = Primary Key (keyvar);
3	keyvar	Index	Index create keyvar / Unique Updatecentiles=5;

The following base SAS code was used to assign these constraints. PROC DATASETS was used for the DATA step and SORT methods, PROC SQL for all other methods. Here are sample SAS logs from the test runs, edited for two versions of basefl_*:

```
proc datasets nolist lib=WORK;
* add integrity constraints;
modify basefl_DS;
```

```
ic create primkey = primary key(keyvar);
NOTE: Integrity constraint primkey defined.
ic create not null (cvar);
NOTE: Integrity constraint _NM0001_ defined.
quit;

NOTE: MODIFY was successful for WORK.BASEFL_DS.DATA.
NOTE: PROCEDURE DATASETS used (Total process time):

proc sql;
alter table basefl_PS add constraint primkey primary key(keyvar),
                      cvar_not_null not null (cvar);
NOTE: Table WORK.BASEFL_PS has been modified, with 4 columns.
quit;
NOTE: PROCEDURE SQL used (Total process time):
```

Examination of the data reveals that the rows in the example data sets can be classified as:

- 4 rows are carry-forward rows, keyvar = 1,3,5,7 in basefl_*,
- 4 rows are update rows, keyvar = 2,4,6,8 in both files (with some columns changed),
- 2 rows are insert rows, keyvar = 9,10 in revisionfl_*.

The goal is to perform update, insert, and carry-forward operations to yield a new version of the basefl_* data set. The final, derived version should have 8+2=10 rows. Different approaches to accomplish this are examined in the sections that follow.

Base SAS programming methods: DATA step and PROC SQL

Note: This section is intended for those with some (minimal) SAS and/or SQL programming experience. Readers who lack this experience can skip this section and go straight to the other methods.

Individuals with SAS programming experience can write programs to perform these operations in the SAS Enterprise Guide environment, as illustrated in the following.

DATA step using MERGE

Applying a revision file to a base or master file is an application for which the DATA step with the MERGE statement is – in general - well suited. For this test, example files were created: basefl_DS, revisionfl_DS, and integrity constraints added to both files (see above) using PROC DATASETS. Using a small Base SAS program, both files were sorted BY keyvar (the sort could instead have been done using SAS Enterprise Guide Tasks). Next, a simple program was run:

```
data basefl_DS;
  merge basefl_DS revisionfl_DS;
  by keyvar;
run;

NOTE: There were 8 observations read from the data set WORK.BASEFL_DS.
NOTE: There were 6 observations read from the data set WORK.REVISIONFL_DS.
NOTE: The data set WORK.BASEFL_DS has 10 observations and 4 variables.
NOTE: DATA statement used (Total process time):
```

The following file was created:

Obs	keyvar	nvar1	nvar2	cvar
1	1	1	2	Los_Angeles
2	2	6	4	Irvine
3	3	5	6	San_Francisco
4	4	14	8	San_Diego
5	5	11	10	San_Jose
6	6	26	12	Long_Beach
7	7	17	14	Long_Beach
8	8	38	16	Pasadena
9	9	46	18	Sacramento
10	10	59	20	Fresno

which reflects the expected results, i.e., the two data sets merged, with total of 10 rows with update rows from revisionfl replacing the relevant rows from basefl.

Integrity constraints were checked by using PROC CONTENTS with the OUT2= option; this yields a data set that contains the integrity constraints for a file. The results of that are:

```
proc contents data=basefl_DS noprint out2=ICs;
run;

NOTE: The data set WORK.ICS has 0 observations and 0 variables.
NOTE: PROCEDURE CONTENTS used (Total process time):
```

Notice that the file containing the integrity constraints has 0 rows and 0 columns. This means that DATA step with MERGE has deleted the integrity constraints that were on the file. PROC SORT, when there is no OUT= file specified, preserves integrity constraints, so the ICs were not removed during sorting. (A citation and link to the relevant SAS documentation on when integrity constraints are preserved in Base SAS is provided in the References section at the end.)

The simple example above overlays complete update rows (i.e., all columns) in basefl with complete rows from revisionfl. However, if only select columns are to be changed and/or transformations are required during the update, then additional programming is required. In that case, use of data set options (keep=, drop=, rename=) and/or extra code for required transformations must be included in the DATA step.

PROC SQL with INSERT INTO and UPDATE

Two example files were created, basefl_PS and revisionfl_PS, and integrity constraints were applied to basefl_PS using PROC SQL. The row updates and inserts were applied in PROC SQL using the code:

```
23          proc sql;
24          INSERT INTO basefl_PS
25          select * from revisionfl_PS where
26          keyvar > 8;
NOTE: 2 rows were inserted into WORK.BASEFL_PS.

27          quit;
NOTE: PROCEDURE SQL used (Total process time):

28
29          * CARRY-FORWARD with update of only 1 column (cvar) using subquery;
30
31          proc sql;
32          UPDATE basefl_PS
33          set cvar = (select cvar from revisionfl_PS where
34              basefl_PS.keyvar = revisionfl_PS.keyvar)
35            where exists
36              (select cvar from revisionfl_PS where
37              basefl_PS.keyvar = revisionfl_PS.keyvar);
```

```
NOTE: 6 rows were updated in WORK.BASEFL_PS.

38         quit;
NOTE: PROCEDURE SQL used (Total process time):
```

The SQL above yields the expected 10 rows. However, the end result is different from the DATA step MERGE approach as the MERGE replaced entire rows but SQL UPDATE replaced only the variable cvar. The clumsy WHERE EXISTS syntax in the UPDATE step is required for the UPDATE to work correctly; see the paper by Hermansen & Legum (2008) for more information on this topic. After the operations above, the integrity constraints were checked using the same method (PROC CONTENTS OUT2=), and the integrity constraints were preserved.

The bottom line on integrity constraints in Base SAS is that, if they are important to your application, you must pay attention to them and use PROC SQL UPDATE and INSERT INTO instead of DATA step MERGE. If you cannot avoid deleting integrity constraints, then you will need to restore them at some point. Depending on the constraints and the processing performed, restoration may be easy or may be difficult.

SAS Enterprise Guide:
Sort-based Method

This is a simple method that replaces all update rows in the base file with the corresponding row in the revision file, or vice-versa, by doing an append followed by 2 sorts – the last of which is an unduplication sort. The project diagram is shown in figure 1. Example files were created and integrity constraints added, as in the preceding cases. After the test files are created, the remainder of the project processing is as follows:

#1: Add a sort id variable to the base file, sortid=2, to create an extract that is a new version of the base file.
#2: Add a sort id variable to the revision file, sortid=1, to create an extract that is a new version of the revision file.

The above are accomplished using the Query Builder task, for each file:
- Drag table name (basefl or revisionfl) into the Select Data area; this will select all columns
- Specify Add a New Computed Column, then choose Advanced Expression, then enter 2 or 1 for the expression, and finally specify sortid as column name.
- Finally, choose Run to create the data set.

If instead you want to keep the rows in the base file that are common with the revision file, then swap the values for sortid in steps 1,2. If the files already contain a "last updated datetime", a "version number", or other, similar variable that is always and consistently different between the 2 files, then steps 1,2 are not required. In that case, use the other variable in place of sortid in step #4. (Alternately, f you trust the results of the append and the data sets are being appended in the correct order, you can skip steps 1-3.)

#3: Append the modified base and revision files

- Use Append Table task to combine the 2 files.
- Highlight one table and Add the other in the task (Tables > Add Table).

This step combines the two files into one, prior to the sorts. Under some circumstances PROC APPEND will preserve integrity constraints. However, the Append Table Task uses PROC SQL, OUTER UNION CORR, instead of PROC APPEND.

#4: Sort the appended file from (#3) by keyvar and sortid

- Recommend use of Sort Data task and not the Filter and Sort or Query Builder tasks.
- Sort by the variables: keyvar and sortid.

#5: Resort the sorted file (#4) by keyvar only; this is an unduplication sort

- You must use the Sort Data task and not the Filter and Sort or Query Builder tasks.
- Sort by the variables: keyvar.
- To make the sort an unduplication, select: Options > Duplicate records > Keep only the first record for each 'Sort by' group. (This will produce SAS code for a PROC SORT NODUPKEY.)

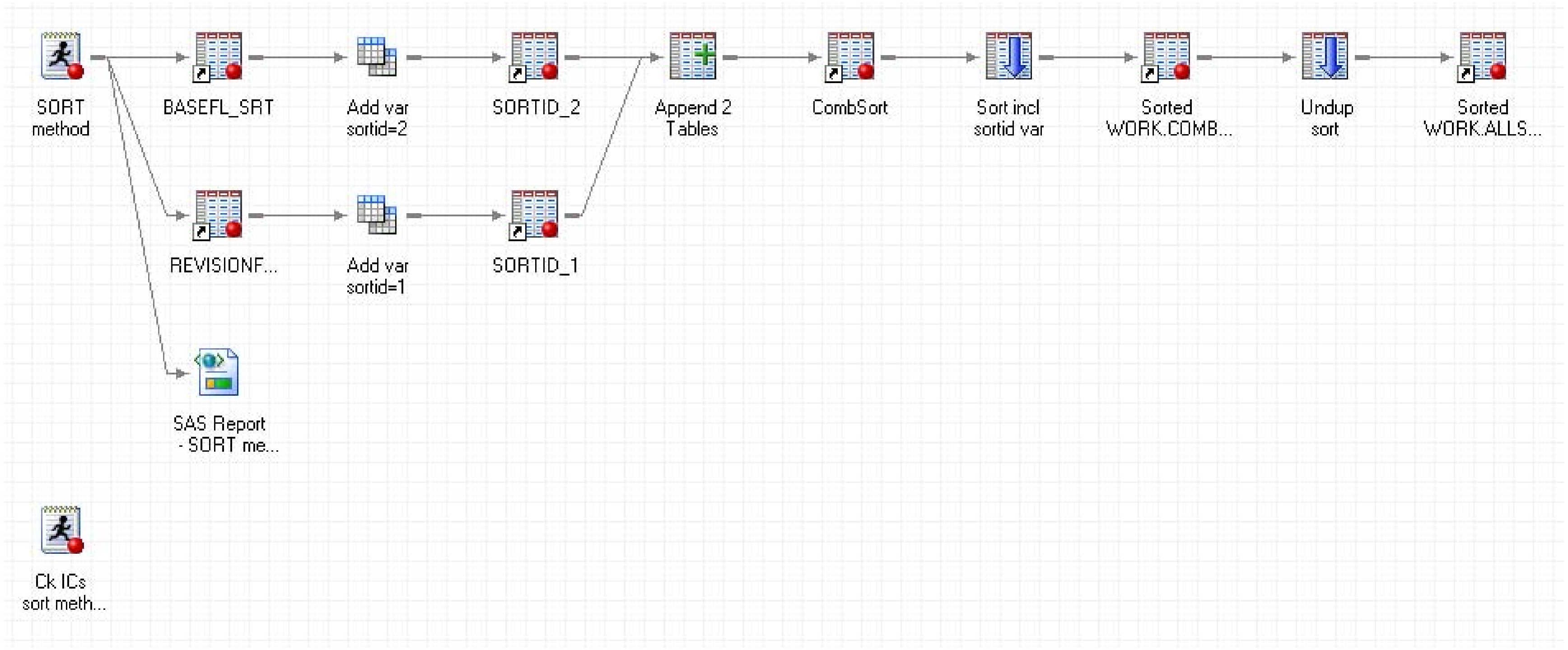


Figure 1. Process flow thread for sort-based method

This method yields the same 10 rows as the two preceding methods. The final file produced was checked and it turns out that integrity constraints are not preserved by this method. This creates an opportunity to make the method potentially more efficient by creating views instead of physical tables in steps #1 - #3 inclusive. (Views do not preserve integrity constraints. This option is relevant when the underlying files are large.) To create views instead of tables:

- #1, 2: Query Builder > Options > Results, select Override the corresponding default… and choose Data View.
- #3: Append Table > Results > Result set format > Data View.

A link to the SAS documentation on this topic (integrity constraints in tables and views) is provided in the references below.

The result of the above two sorts in sequence, the last an unduplication, is to keep the update rows from the revision file while discarding the same rows from the base file. The concatenation done in step #3 captures the carry-forward and insert rows, so all rows (update, insert, carry-forward) are included in the final result. This method is best suited for applications where update rows come from only one of the source files, with no transformation of columns required.

SAS Enterprise Guide:
SQL Outer Join Method

This is the most complex but also the most powerful/flexible method. The approach is to do a full outer join (on keyvar) of the two files, basefl and revisionfl, and create indicators to support follow-up joins that identify whether each row is an update, insert, or carry-forward. The indicators are then used to split the input files by row type, which can then be recombined as needed. Carry-forward extracts are created for both input files, and they are combined in a join that illustrates the infrastructure available to create a derived, transformed carry-forward extract. The project is shown in Figure 2.

After the test files are created, the remainder of the project processing is as follows:

#1: Outer join of the two files to create a single file with multiple versions of keyvar, plus indicator variables that classify the rows as carry-forward, update, insert. Using the Query Builder Task, starting with basefl:

1. Add Tables: add revisionfl
2. Join Tables: join the two tables on keyvar, specify outer join
3. Select Data: drag keyvar from each table into the Select Data listing, rename as keyvar_base and keyvar_rev
4. Using New Computed Column, Advanced Expression, create 3 new columns, emulating the expressions shown in the source code below (structure is: variable name in comments: expression AS column name; t1 is basefl, t2 is revisionfl):

```
/* keyvar */
  (COALESCE(t1.keyvar,t2.keyvar)) AS keyvar,
/* UICF_ind */
  (1.0 - MISSING(t1.keyvar) + 2 * (1.0 - MISSING(t2.keyvar))) LABEL=
  "update-insert-carry-forward indicator, numeric" AS UICF_ind,
/* UICF_char */
  (CHOOSEC((1.0 - MISSING(t1.keyvar) + 2 * (1.0 - MISSING(t2.keyvar))),
          'carry-forward','insert','update'))
   FORMAT=$16. LABEL="update-insert-carry-forward indicator, character"
   AS UICF_char
```

   - Keyvar is the coalesce (or coalesce**c** when keyvar is a character variable) of the two keyvar variables. It is needed for the joins that follow in step #2.
   - UICF_ind is a numeric indicator with values 1,2,3; it classifies each row by type.
   - UICF_char is UICF_ind recoded into character form. You can avoid retyping the expression for UICF_ind by selecting it as the first operand for the CHOOOSEC function.

5. Run the Task to create the indicator file.

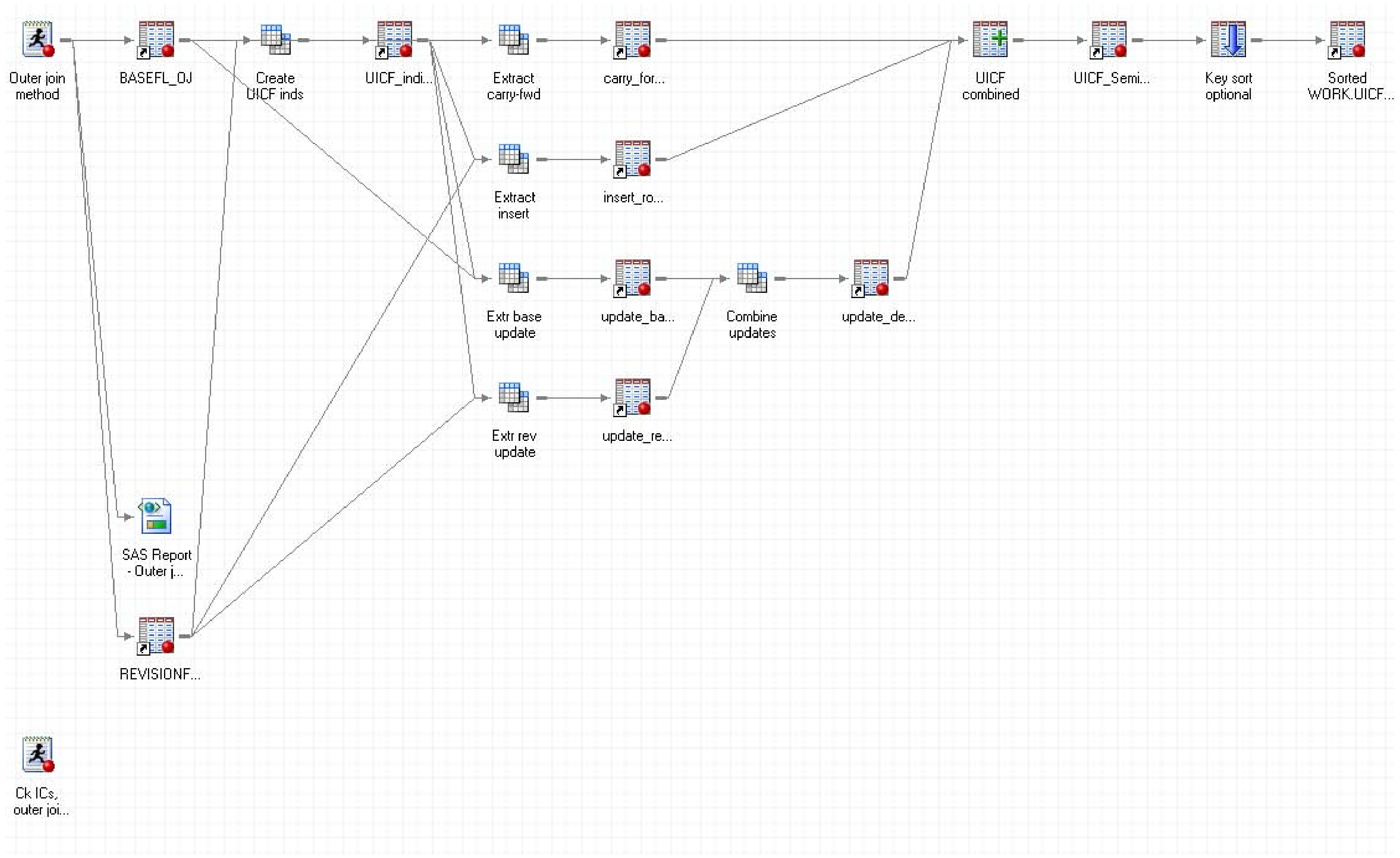
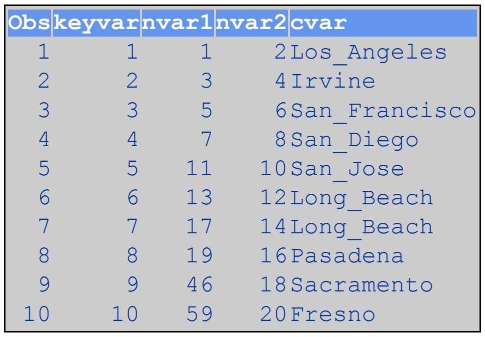


Figure 2: Process flow thread for outer join SQL method

#2 - #5: Using the Query Builder Task, do inner joins -- basefl/revisionfl joined with the indicator file (from #1), using a filter WHERE UICF_char = insert, update, carry-forward. This creates 4 extracts: carry-forward rows from basefl, insert rows from revisionfl, plus 2 versions of update rows, 1 for each of basefl and revisionfl.
#6: The two files containing update rows are combined via an inner join that uses all columns from basefl except cvar, which comes from revisonfl. (Example processing only; other processing possible in this step.)
#7: The component extracts are recombined using the Table Append Task.
#8: Optional: the combined file is sorted by keyvar, using the Sort Data Task.

The resultant output file is:



Obs	keyvar	nvar1	nvar2	cvar
1	1	1	2	Los_Angeles
2	2	3	4	Irvine
3	3	5	6	San_Francisco
4	4	7	8	San_Diego
5	5	11	10	San_Jose
6	6	13	12	Long_Beach
7	7	17	14	Long_Beach
8	8	19	16	Pasadena
9	9	46	18	Sacramento
10	10	59	20	Fresno

which has 10 rows and reflects the intended processing: update rows have all columns from basefl except for cvar, which comes from revisionfl. More complicated transformations and assignments can be done in step #6, if desired, to determine the column values for the derived update rows. The integrity constraints were tested and it was determined that they are not preserved in this method.

Limitations of SAS Enterprise Guide

The fact that integrity constraints are not preserved by most SAS Enterprise Guide Tasks may – in some contexts - limit its use for database applications. The focus here is on SAS files as most projects work primarily with SAS files or, if external databases are involved, work with SAS files that are extracts from the target databases. If your infrastructure supports it, and you have the relevant permissions, SAS Enterprise Guide can write to external databases. However, note that the method used here to check integrity constraints (PROC CONTENTS OUT2=) does not work with relational database files.

If integrity constraints are important to your application and you can write simple SAS programs, then you can (try to) restore the constraints by writing code and including it in the project. However, this might not be as easy as it may seem, as processing with no constraints may yield results that fail to satisfy the constraints later, when you try to restore them. (It is also advisable to use natural keys where available, when performing update, insert, and/or carry-forward operations.)

The bottom line here is that if you need to do serious database work and preserve integrity constraints, then you probably should be using SAS Data Integration Studio or other SAS data management products. SAS Enterprise Guide, even though it is sometimes referred to as “the Swiss Army knife of SAS”, is not a substitute or replacement for those products.

The transformations in SAS Data Integration Studio (analogous to SAS Enterprise Guide Tasks) provide ways to manage integrity constraints, including the ability to temporarily remove and/or refresh them (before/after data load) in the Table Loader Transformation. Starting with version 4.4, SAS Data Integration Studio also supports Insert Rows and Update transformations to accomplish the operations discussed here.

Optimal database-related applications of SAS Enterprise Guide

Integrity constraints are not an issue for many (perhaps most) applications. In that case, SAS Enterprise Guide is an excellent tool for the following database-related applications:

- Reporting
- Analysis of data, e.g., use the Query Builder to investigate anomalies and other issues of concern

- Combining data from multiple databases and files for cross-validation
- Modeling, prototyping, and testing of joins that are in-scope and can be efficiently done in the Query Builder Task.

SAS Enterprise Guide can be used with the SAS Data Integration Studio as a complementary tool. The native reporting capabilities of SAS Data Integration Studio are extremely limited. You can determine the WORK library for a SAS Data Integration Studio job by inserting in precode (for each job when running manually):

```
%put WORK library path = %sysfunc(pathname(work));
```

which gives you the path to the WORK library used for the SAS Data Integration Studio job. Run the job manually and harvest the WORK library path from the log. Then, in SAS Enterprise Guide, point to the WORK library with a libname (not “WORK”) and copy the files into your project to surface them for use with Tasks. This gives you all the power of SAS Enterprise Guide for analysis (and validation) of your SAS Data Integration Studio files. You can do the same thing *after* batch runs of SAS code deployed by SAS Data Integration Studio, if you specify the NOWORKTERM option in your job (precode) so your WORK library is not erased after your run completes.

The “copy...into” aspect mentioned above may require some work on the SAS Data Integration Studio side, e.g., you may need to create physical files instead of views. Also, it is often possible to copy/paste code from the SAS Data Integration Studio into a program window in SAS Enterprise Guide, modify it, and test run it.

Epilogue

So long as integrity constraints are not a major issue, SAS Enterprise Guide can be productively used for a wide variety of update, insert, and carry-forward operations on files. In cases where integrity constraints are a major issue, SAS Data Integration Studio is probably the better tool to use. However, the two tools are complementary, and work very well together.

References:

Hermansen, Sigurd; Legum, Stanley E. “Existential Moments in Database Programming: SAS® PROC SQL EXISTS and NOT EXISTS Quantifiers, and More” Proceedings of SAS Global Forum 2008.
http://www2.sas.com/proceedings/forum2008/084-2008.pdf

SAS Institute, Inc. *SAS(R) 9.2 Language Reference: Concepts, Second Edition.* Online documentation:
- Understanding Integrity Constraints:
  http://support.sas.com/documentation/cdl/en/lrcon/62955/HTML/default/viewer.htm#a000403555.htm
-

SAS Institute, Inc. *SAS(R) 9.2 SQL Procedure User's Guide.* Online documentation:
- Creating and Using Integrity Constraints in a Table
  http://support.sas.com/documentation/cdl/en/sqlproc/62086/HTML/default/viewer.htm#a001396785.htm

Paper presentation history:

Versions of this paper are being or were presented at the following SAS User Group conferences:
- Western Users of SAS Software, 2012, Long Beach, California
- SAS Global Forum 2013, San Francisco, California.

Contact Information:

Thomas E. Billings
Union Bank
Basel II - Retail Credit BTMU
400 California St.; 9th floor
MC 1-001-09
San Francisco, CA 94104

Phone: 415-765-2567
Email: tebillings@yahoo.com