Rollback of SAS® Relational Databases to Former Historical States

Keith Adams, BARRA Inc., Berkeley CA

Abstract

This paper presents a method to setup relational databases using SAS data sets such that, at your will, the data in the databases appear either up-to-date, or appear exactly as they would have appeared at any historical database date. This paper illustrates (in simplified form) the methodology we have developed.

The methodology uses SAS Macro Language, SAS/AF®, and SAS/FSP®. Readers should have some understanding of DATA-step views and PROC SQL views. The presentation is targeted towards people responsible for setting up relational databases.

Introduction

The rollback capability was developed as part of a comprehensive tool-set of SAS macros intended to be used to setup, manage and access flexible relational databases using SAS data sets. In order to adequately explain how rollback was enabled in this tool-set, it is necessary both to provide some background into how the databases are constructed, as well as to document the motivation behind developing such a tool-set using the SAS System. Why not use a commercial DBMS?

BARRA has experimented, over a number of years, with various solutions to storing the data we use to analyze financial markets. However, there has been much internal dissatisfaction with the results of these experiments. The consensus seems to be:

- We have somewhat unique data requirements - commercial DBMS can never completely meet our needs as they are usually jacks of all trades.
- Access to large tables is too slow!!!
- The commercial DBMS do not offer sufficient flexibility in the construction of new user tables, and the storage of data across multiple file systems.

The idea was broached of writing our own DBMS. Right from the start, the SAS System was an obvious candidate: the internal clients of our databases are almost entirely SAS programs. If the data was maintained in SAS data sets, then an additional access layer would not be necessary. After an internal review of previous experiences with commercial DBMS, we came up with the following design requirements:

- The data must be available exactly as delivered by the data-vendor, with the same variable names, records and identifiers.
- Users must have the capability to rollback the databases to previous historical states.
- Data access must be faster than the commercial relational-database software we used in the past.
- Users must be able to easily create their own databases and tables, as well as get access to the databases created by other users.
- The location of the data should be invisible to users, so that users would not have to worry about librefs and data set names. Moreover, as some of the tables could be several gigabytes in size, the capability of distributing a table across multiple data sets should also not affect the users' perception that they are accessing a single table.

In this presentation, please keep in mind the nature of the data we are dealing with: it is almost always time-series (although the methodology is more broadly applicable). When we talk about modifying existing database records, therefore, we are talking about ammending records FOR a historical date ON a later date, with corrected or more up-to-date information.

Design Realization

There is not space here to discuss the full system design we arrived at, so only the machinery that enables the rollback will be discussed. The raw, unmodified vendor data is stored in "tables". Each table actually consists of multiple elements, tied together purely by the use of naming convention. The elements pertinent to this discussion are:

- a **RETRO** data set holding former values of fields that have been updated
- a **GRAVEYARD** data set holding records the user deletes from the database
- the data set or view. It is built as a view if you want to distribute data over multiple data sets due to space considerations.

How the Tables are Loaded

The rollback machinery begins at the point where the tables are loaded. A 'load' involves either the insertion of new records into the database, the modification of existing records from a transaction data set, or the deletion of existing records. A single, complex macro is used to perform all loading: a database is not modifiable by any other means than through this macro. This ensures that all modifications are recorded using the correct protocols, thus ensuring later rollback security.

Insertion of New Records

As each empty table start with the insertion of the first records, this is a good point to start. Records are inserted 'as-is' from the transaction data set. A variable recording the actual date the record was inserted is attached to each record.

Loading Transaction Data Sets

We will disregard deletions for the moment. Subsequent loads are usually a combination of inserts and modifications. In the usual mode of operation of the macro, the program takes on the responsibility of determining which records are new and which records are already in the database. First, new records are identified, and inserted, in the same way the first records were inserted. Then, transaction records corresponding to records which are already in the database are 'delta-checked' and, if found to be different from the records already in the database are used to modify those existing records. More on this later, as this step is too crucial to be described briefly.

Deletions

If it is necessary to delete records, then you run the macro in DELETE mode, and all records that are both in the transaction data set and the database are removed from the database and placed in the graveyard data set. Added to each record is a new variable recording the actual date the record was removed. This information will be useful in reconstructing the database as it appeared before the deletion.

The Heart of the Matter - Delta-Checking

By 'delta-checking' we mean the comparison of transaction records with existing database records to determine differences and act upon them. The mechanics are fairly straight-forward. First, an image is made of the database, holding only those records which are also in the transaction data set. Then this image is merged with the transaction data set, and the transaction data set information variables (i.e. all variables except the BY variables) are renamed as they are read in. Arrays are setup for both the variables from the image and the corresponding transaction values, and the arrays are stepped through to determine if the values are different. For each different value, a 'retro' record is generated (this will be explained later). If any differences are found in a record, the record is tagged to be used to modify the database.

Retro records contain the values of the BY variables (to identify which record was modified), the name of the variable being modified, and the former value of that variable before modification, along with the actual date of the modification. Note that the actual insert date that the original record was inserted into the database is only retained in the database, and the date of the modification is only recorded in the retro records. This ensures that only the minimum amount of information necessary for a rollback is maintained, and there is no duplication of such information.

The retro records are added to the retro data set associated with the main table, and this data set is compressed (to be able to store information about any variable, either character or numeric, requires a large character field to store the former value of the field - compression will minimize the costly effects of storing values with little information content in a large character field).

Once the delta-check is complete, the tagged records replace the original database records. A new image of the database, holding the records modified by the transaction data set is created and once-again delta-checked, again creating retro records. These retro records are compared to the original retro records already captured. This ensures that the retro records accurately record actual database modifications.

How to Rollback a Table

The most common type of rollback is required when you say, for example: "Show me what value the database gave for the total 1978 capitalization of security A on Jan 12, 1990, even though we deleted this security from the database in 1993."

Universal Rollback Data Set

In order to allow a table to be rolled-back, a "universal rollback data set" needs to be created. This duplicates the record format of the table. But whereas the table contains only one record

per identifier and date, the universal rollback data set may contain multiple records for each original record in the table.

For example: if the Jan 92 closing price for Security A was amended in June 1994, then the universal rollback data set will contain two records for the Security A, January 92 information. If the Jan 92 return for Security A was originally quoted incorrectly and subsequently corrected in April 95, then a third record will be generated in the universal rollback data set, with all three records being for the same security on the same date. Each additional record shows the table information for the security as of a different table modification date. Only records which have never been updated would have a single record in the universal rollback data set.

When you create this data set, you can optionally specify a date range over which rollback is desired (thus limiting the size of the data set somewhat).

The "engine" for the creation of the universal rollback data set is one of the key successes in our drive to create an efficient rollback methodology. The process is begun by combining the current records in the table (and the deleted records from the graveyard data set) with the retro records, simultaneously adding start and end date variables (d_start and d_end respectively). See overleaf for an illustrated example.

Multiple Database States

The goal is to make rollback as transparent as possible. In our system, you typically access a database table through a macro variable. For example:

```
PROC MEANS DATA=&VCMPYANN
```

For this reason, the design goal for rollback was that you should be able to access the rolled-back version of the table exactly the same way, and be able to switch back and forth between different rollback dates and the present.

This is partly accomplished by manufacturing multiple objects, each of which represents the database on every time period you request the potentiality to rollback to. So if the macro variable representing the table is reassigned to point to the appropriate object, then you will see the rolled-back data not the current data.

These objects are simply data set views of the universal rollback data set, each subsetting the univeral rollback dataset to read only those data set records that were valid on the date the object represents. Each data set view is in a separate directory. As the macro variable that represents the table just resolves to a two-level SAS data set name, then simply redefining the libref to the correct directory will be sufficient to allow you transparent access to rolled-back data.

How You Perceive Rollback

Let us suppose that you, as the table owner, have setup a universal rollback data set, and a directory structure of data set views offering multiple database states for tables VCMPYANN and VCMPQFND. We have designed a SAS/AF interface to the database system which, amongst other things, lists all tables you have access to. If you get into this application, you may see VCMPYANN and VCMPQFND, as well as other tables. If you click on table VCMPYANN and choose "See data" you will use FSVIEW to browse the current table.

Now for rollback! You click on the rollback option, and choose a date to rollback to. After a few seconds, a revised list of tables is displayed: only those that have been setup for rollback. Unless you have setup rollback for any other tables, this list might just include VCMPYANN and VCMPQFND. Now choosing the "See data" option will display the data as it appeared on the rollback date. Your interaction with rolled-back tables is transparent!

Rollback ... Tested

We have tested the methodology on a large database of 200,000 records with 87 variables on a SUN IPC workstation. Creating the universal rollback data set took less than an hour, and creating the 'multiple database states' a matter of minutes. Once created, bringing up a window on any of the views of the database on different historical states through the FSEDIT procedure took a couple of minutes.

Example: Creating a Universal Rollback Dataset

Suppose the database is an index of security prices. It might look something like table 1.

Table 1: Database

identifiers		data items		date-stamp
security	date	hiprice	loprice	d_insert
A	9408	10.25	9.5	950324
A	9409	10.5	10.5	950324
B	9408	16.0	15.25	950324

The retro records might look something like table 2, thus recording the fact that the value **hiprice** for security A for 9408 was adjusted twice by later updates to the database (i.e. none of the other fields or records were ever modified subsequent to their insertion into the database).

Table 2: Retro Records

identifiers		data items		date stamp
security	date	itemname	oldvalue	d_retro
A	9408	hiprice	10.5	950401
A	9408	hiprice	10.25	950502

The combined data set would look like table 3. If you remember how the retro records were constructed, the retro-date is actually the date on which the value of the 'itemname' **stopped** being the 'oldvalue', so in the joined data set above we use the retro-date minus one as the 'd_end'. This simple MERGE provides the basis of a relatively painless construction of the universal rollback data set.

Table 3: Intermediate Data Set

identifiers		data items				date stamp	
security	date	hiprice	loprice	item-name	old-value	d_start	d_end
A	9408			hiprice	10.5		950331
A	9408			hiprice	10.25		950501
A	9408	10.25	9.5			950324	end-of-time
A	9409	10.5	10.5			950324	end-of-time
B	9408	16.0	15.25			950324	end-of-time

In the same DATA step, the following fixes are made:

1. The database variables are looped through, and when the 'itemname' in a record which came from the retro data set (i.e. in this case the first two records) is found to match the current variable, the value in the 'oldvalue' variable is put into the field whose name is in the 'itemname' variable.
2. The 'd_start' values for the first two records are created by simply incrementing the previous record's 'd_end' values by one.
3. We output the third record separately to a utility data set, keeping only the 'security', 'date', 'd_start' and 'd_end' variables. This will be reMERGEd by 'd_end' into the data set later to correctly set the d_start value for the first record., so in this output record, 'd_end' is actually set to the retained 'd_end' value captured from the first record.
4. The 'd_start' variable for the third record is set to one day after the previous 'd_end' value.

The final data set is shown in table 4, and MERGEs back in the record output in step 3 above, to fix 'd_start' for the first record, and also fills in the missing values for variables that did not change (in this case 'loprice' for the first two records) by re-SETting the data set shown in table 3 to pick up the appropriate value of 'loprice', in this case from the third record.

Table 4: Universal Rollback Data Set

identifiers		data items		date stamp	
security	date	hiprice	loprice	d_start	d_end
A	9408	10.5	9.5	950324	950331
A	9408	10.25	9.5	950430	950501
A	9408	10.25	9.5	950502	end-of-time
A	9409	10.5	10.5	950324	end-of-time
B	9408	16.0	15.25	950324	end-of-time

There you have it! With just two DATA-steps, and no sorting, we have constructed the entire history of the database.

Conclusion

It is possible to maintain and modify SAS data sets in such a manner as to make available the capability to rollback these data sets quickly so that they appear as they did on a chosen historical date..

Moreover, this functionality can be made available in a transparent fashion: all you have to do, if the rollback structure has been setup, is to set a date switch, and then access the data in the same way you always access the data.

Acknowledgments

The design and programming could not have been accomplished without the persistence and inspiration of Tom Wood and Miguel de Avila.

Author

Keith Adams
BARRA, Inc.
1995 University Avenue
Berkeley CA 94704
Tel: (510) 649-4686
E-Mail: keitha@barra.com