

Emulating FIRST. and LAST. SAS® DATA Step Processing in SQL? Concepts and Review

Thomas E. Billings, MUFG Union Bank, N.A., San Francisco, California



This work by Thomas E. Billings is licensed (2018) under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/).

ABSTRACT

The questions of whether and how FIRST. and LAST. BY variable SAS® DATA step processing can be emulated in SQL qualify as FAQs: frequently asked questions. We begin with an overview of terminology and explain the different environments involved. Then we cover the common approaches to this issue. #1: if the data rows are uniquely ordered by existing variable(s), those can be used to support emulation. #2: in SAS PROC SQL, DATA step views can be used to inject the values of FIRST. and LAST. BY variables; alternately the undocumented SAS function `monotonic()` can be used to inject a row number into the data, which can be used in approach #1. However as the order of pull can change from run to run in SQL, the results with `monotonic()` are not reproducible, hence problematic for data validation and audits. #3: most other SQL dialects have non-standard SQL extensions that will supply a row number; again the order of pull can change from run to run. Then we discuss whether this type of emulation makes sense; that is why the article title includes a question mark (?). We discuss some non-SQL alternatives, and specify the very narrow contexts where the emulation is reasonable. Finally, we note that DS2 supports BY, FIRST., LAST. processing and it can run in-database, making it a possible alternative to SQL and the DATA step.

PREVIOUS DISCUSSIONS

The topic re: how to emulate FIRST. and/or LAST. BY variable SAS® DATA step processing in SQL has been discussed numerous times in the past, including:

- Multiple times on the SAS-L mailing list in the period 1996-2013;
- On the SAS Support Communities website in 2009;
- In a recent paper by Kirk Lafler (2017).

On SAS-L, some of the individuals who have commented on this issue include: Paul Dorfman, Michael Raithel, Ian Whitlock, Sigurd Hermansen, Mark Terjeson, and others.

This paper provides an overview of the concepts involved and describes some of the proposed solutions; we also discuss the constraints on the various solutions. Code snippets and pseudo-code are provided; however no specific examples are given here as the intent is to discuss the topic on a conceptual level. In the next section, we begin by clarifying terminology, followed by an overview of the different environments.

INTRODUCTION: SQL VS. DATA STEP ENVIRONMENTS

Terminology. The SAS system provides 2 native SQL dialects:

- PROC SQL
- PROC FEDSQL

PROC SQL supports 2 processing modes:

- Explicit pass-through of native (non-SAS) SQL dialects to the target RDBMS systems, e.g., PROC SQL serves as a “wrapper” to pass Oracle SQL directly to a connected Oracle data base;
- Implicit pass-through: a user writes native SAS SQL code and PROC SQL analyzes the code then divides it into logic that can be run on the SAS server and logic that must be exported to the connected RDBMS. In this situation, PROC SQL generates the SQL sent to the RDBMS.

Code intended for explicit pass-through should be in the SQL dialect of the target RDBMS. Code for implicit pass-through is in the native SAS PROC SQL dialect, and the SAS system translates that code into the SQL dialect for the target RDBMS; the generated code is passed to the connected database(s) and results are returned to SAS. Implicit PROC SQL code that operates only on SAS data sets runs only on SAS servers as there are no external RDBMS systems.

The SAS system allows users to operate on RDBMS files (and other types of files, e.g., XML) as if they were SAS data sets in the DATA step and PROCs. This is done by defining the RDBMS link via a LIBNAME.

The description above is provided for clarification, i.e., when we ask about emulating FIRST. and LAST. processing “SQL”, exactly which SQL environments are relevant? In the discussion here, we focus on 2 main environments:

1. PROC SQL implicit pass-through SAS SQL code and/or code that operates exclusively on SAS data sets.
2. Other, non-SAS SQL dialects:
 - PROC SQL explicit pass-through to an RDBMS,
 - non-SAS SQL running on an RDBMS.

FIRST. and LAST. BY Variables. The BY command in SAS allows the processing of pre-sorted or pre-grouped data. The 9.4 documentation for BY is at:

<http://documentation.sas.com/?docsetId=lrcon&docsetTarget=p135g8pyd7sdaen1cnr6altwb9ye.htm&docsetVersion=9.4&locale=en>

BY is supported in the DATA step, the DS2 language (PROC DS2), and many procedures. In the DATA step, BY works with the SET, MERGE, MODIFY, UPDATE statements, with the 1st 2 being the most common applications. If your data set is sorted or grouped (the latter refers to use of the BY statement options NOTSORTED, GROUPFORMAT), then when a DATA step (or PROC DS2) runs with BY, the SAS system creates temporary variables for each BY variable. For example, if appropriate SAS code includes:

```
BY var1 var2;
```

Then when the code executes, SAS creates temporary numeric variables first.var1, last.var1; first.var2, last.var2. These variables have 0/1 values for false/true, and indicate if the current row is the first or last row being input into a DATA or PROC step from the BY-group defined by the specified variable. The relevant 9.4 documentation is at:

<http://documentation.sas.com/?docsetId=lrcon&docsetTarget=n01a08zkzy5igbn173zjz82zsi1s.htm&docsetVersion=9.4&locale=en>

The main focus here is on the DATA step, though we note that BY is also supported in the DS2 language. We also note that “first” and “last” suggest an implicit ordering, a point addressed below.

SQL and the DATA step are very different environments. The differences relevant to this topic are summarized in the following table.

	DATA Step with BY, operating on SAS data sets	SQL
Input data set at rest, before processing	Partially or fully ordered	No order
Input process	Partially or fully ordered	No order
FIRST. & LAST. variables	Created on input; temporary variables	N/A
Output process	Partially or fully ordered	Partially or fully ordered only if ORDER BY used
FIRST. & LAST. variables	In output file only if values saved in permanent variables	N/A
Derived output data set at rest, after processing	Partially or fully ordered	Order not preserved in SQL RDBMS files.
RETAIN variable functionality	Available and commonly used with FIRST., LAST.	N/A
Row number variable	Automatic variable _N_	Undocumented SAS monotonic() function; similar SQL extensions in other dialects

Next, we examine some of the solutions proposed for emulating FIRST. and LAST. processing in SQL.

#1: TARGET DATA SET CONTAINS VARIABLES THAT DEFINE A UNIQUE ORDERING

This approach will work with any SQL dialect, but it is constrained to data sets that include variable(s) that provide a unique ordering within the row sets/groups defined by the sorting or grouping variables. Here the SQL aggregate functions MIN() and MAX() are used with GROUP BY to identify the FIRST. and LAST. unique values for each group. These values are then joined with the original data using relevant SQL logic to create analogues of the FIRST. and LAST. indicator variables.

To clarify, let's consider an example, a SAS data set sorted as follows:

BY V1 V2;

with order variable D, and we want to create analogues of indicator variables FIRST.V2 and LAST.V2. The following steps in SQL will accomplish this (pseudo-code follows):

```
/* Get values of D that are FIRST., LAST. in the row sets defined by variable V2 */
```

```
CREATE TABLE (or VIEW) my_file as  
SELECT V2, min(D) as first_V2, max(D) as last_V2  
from target_data_set  
GROUP BY V2  
/* Optional – ORDER BY V2 */
```

If the data set is sorted in ascending order of variable V2, then MIN() will be the FIRST. value, and MAX() the LAST. value. If the data set is sorted descending order, then MAX() is FIRST. and MIN() is the LAST. value. The MIN, MAX functions also work with character data, i.e., the ordering variable does not have to be numeric -- but be cognizant of the character sort order in the locale/language you are working in, and other relevant issues like (possibly varying) string length.

Similar code is needed for V1 if FIRST.V1 and LAST.V1 indicator variables are needed. Depending on circumstances, the ordering variable for V1 can be the same as V2 (i.e., D) or a different unique ordering variable present in the target data set.

The next step is to join the files that contain the first/last values with the target data set (join on V1, V2) and use CASE statements (or in SAS SQL, IFN, IFC, or other similar functions) to create indicator variables FIRST_D, LAST_D. These indicator variables can then be used in SQL joins.

The SQL above is described as multiple steps for clarity; of course you can use in-line views, subqueries, etc. to do it all in a single SQL process. The uniqueness of the ordering variable(s) is important, as otherwise the logic to create the indicator variables will identify multiple rows as first/last (for at least one value of say V1 or V2) in the target data set.

This method is illustrated in Lafler (2017), where an additional indicator is defined: BETWEEN. which is analogous to FIRST. and LAST.

[Remark: for information on IFC, IFN functions, see Billings \(2012\).](#)

#2: PROC SQL/SAS NATIVE APPROACHES

The approaches in this section are limited to SAS PROC SQL operating on SAS data sets. These approaches rely on native SAS system features.

The first approach is easy – use a SAS DATA step VIEW to create “permanent” variable analogues of the temporary FIRST., LAST. variables, e.g.; code like this:

```
DATA my_view / VIEW=my_view;  
  SET myfile;  
  BY V1 V2;  
  First_V1 = FIRST.V1;  
  Last_V1 = LAST.V1;  
  First_V2 = FIRST.V2;  
  Last_V2 = LAST.V2;  
RUN;
```

Permanent is in quotes above because views create temporary files, hence the contents of those files are also temporary. Of course similar permanent variables could be added to a SAS data set instead of a view (although the indicator variables would need to be updated every time the base data set is changed).

If the variable names are so long that prefixing them with FIRST_ and LAST_ will exceed the maximum variable name length, then F_, L_, or simply F, L can be used as prefixes. In the worst case (at least one sort/grouping variable name is already maximum length), then new variable names can be defined for those indicator variables.

The DATA step VIEW above can then be used in SAS PROC SQL. Note however that the row input in PROC SQL might not be ordered, and this may impact the logic needed for emulation in SQL. In particular, the lack of ordered input can (negatively) impact logic that includes RETAIN variables.

Another approach is to use the SAS monotonic() function in PROC SQL to create a row number in the run stage. The row numbers can be used to order the data set and then apply approach #1 above to create analogues of FIRST_, LAST_, variables.

The constraints of this method are as follows.

- The monotonic() function is non-production, hence not supported by SAS Institute; i.e., no Tech Support if it breaks.
- SQL input order is not fixed and can vary from run to run, meaning that different rows may be chosen as first/last if this function is used in code reruns (even if there are no changes in the input data set). In short: not reproducible, creates variation in output/results that can be problematic in audits.

If you want to use this function, do a single pass through the data, create row numbers, save the file, and then use the updated file with row numbers in later, separate SQL steps. Similarly, a row number can be inserted in a DATA step (or DATA step VIEW) using the automatic variable _N_, i.e., code like this:

```
Perm_N = _N_;
```

And save the new version of the data set with the indicator variables and use it in downstream applications.

#3: MANY OTHER SQL DIALECTS HAVE ROW NUMBER CAPABILITY

Other, non-SAS SQL dialects often have non-standard extensions that provide capability similar to the SAS monotonic() function. For example, Oracle has:

- A pseudo-column (variable) ROWNUM similar to the SAS _N_ automatic variable
- If you have a variable that fails as an ordering variable due to duplicate values, you may be able to use DENSE_RANK to insert rank numbers which can serve as an artificial ordering variable in some cases.

Microsoft T-SQL also has a ROWNUM() function and DENSE_RANK. Check the documentation for the target SQL to see if it supports row numbers and/or ranks to create artificial ordering variables. Once you have an updated file with row or rank numbers, it can be used in approach #1 to create analogues of FIRST_, LAST_, variables.

Remark: SAS PROC RANK has no exact analogue of DENSE_RANK; it handles tied values differently. PROC SQL also does not have an analogue of DENSE_RANK.

Savvy readers may note that row number could be injected into a permanent variable in an RDBMS table and then it would be stable/reproducible when used for first/last emulation. If the table does not change this is possible; however, if rows are added to or dropped from the table then updating the row number will be a hassle. Also, most SAS programmers don't have the access privileges to add a variable to a production RDBMS table – especially a variable that the relevant DBA may consider superfluous and refuse to allow to be added.

DOES THIS TYPE OF EMULATION MAKE SENSE?

This is a fair question, given the major differences in environments, SQL vs. DATA step, i.e., the DATA step has ordered or grouped input and output, vs. random input in SQL and ordered output is not preserved in RDBMS files.

Simple vs. complex. Most of the discussion on this topic uses simple examples, e.g. a DATA step that has a SET of a single file with very few BY variables, and relatively simple SQL code to create analogue indicator variables. However, DATA steps can be extremely complex with multiple instances of SET, MERGE operating on multiple files plus UPDATE, MODIFY, and large numbers of BY variables. Similarly, SQL can be very complex, involving multiple files and joins, logical layers, and different join methods (which you often cannot control).

Alternatives. Let's examine some alternatives to first/last processing. A common application of first/last is to create an unduplicated row set or to select a sample row from multiple/grouped rows. PROC SORT supports this functionality, via the options:

1. NODUPKEY
2. DUPOUT=
3. NOUNIQUEKEY
4. UNIQUEOUT=

The first 2 above are long-standing features of SAS and are well-known; the latter 2 are relatively new (specifically, new as-of SAS 9.3) and are discussed in Billings (2017) with analogue SQL code.

Another common use of first/last is to create intermediate values which are summed. Appropriate use of the SUM() aggregate function in SQL paired with GROUP BY may accomplish the same end result, with no need for first/last. However, additional logic may be needed here if the target is to create sums and then insert them in a "sample" record that has additional variables.

RETAIN variables are frequently used with first/last. When used to form sums, the preceding comment applies. For select other applications, it may be possible to replace RETAIN variables with intermediate variables and do multiple passes through the data to accomplish the desired processing.

Complex applications. For complex applications with first/last, the following points should be investigated to determine the optimal approach:

- The target processing should be reviewed against high-level objectives/requirements to make sure that first/last processing is really necessary.
- Consider the alternatives: What approach is optimal to accomplish the high-level objectives?
- Consider: Does (all of) the processing have to be done in SQL? The DATA step is the optimal environment for first/last processing. Is a DATA step + SQL approach feasible? Can you use DS2 in-database instead (discussed below)?
- Review: Is SQL cost-effective here or required for other reasons?

The last point refers to the cost-benefit issue and is very important in deciding whether complex first/last DATA step logic should be migrated to SQL.

Reproducible first/last emulation in SQL is limited to narrow contexts. If your application is relatively simple **and** the tables/files you are using include ordering variables **and** there are no better alternatives (see above), then analogues of FIRST. and LAST. variables can be created in SQL, i.e., the emulation makes sense in this narrow context.

If there are no natural ordering variables in the target database tables, it may be possible to create synthetic ordering variables and force the emulation, but the lack of order in SQL input (also order is not retained in output database tables) means the results will not be reproducible. This is a major constraint

and problem in audits and reproducible research applications. First/last emulation in these circumstances should be avoided.

DS2 in-database processing – an alternative to SQL. While the primary emphasis in this paper is on the DATA step, the DS2 language also supports BY with FIRST., LAST., and DS2 can run in-database, inputting RDBMS tables via PROC FEDSQL code that executes in the SET statement. (Your site may need to license additional SAS software products to run in-database.) Recoding a DATA step to DS2 and running in-database may be an appropriate alternative to SQL in some cases. However, the requirement that the target data files contain ordering variables also applies to DS2 operating on RDBMS tables, as otherwise the results are not reproducible.

CONCLUSION

In certain narrow contexts, it is possible to emulate first/last DATA step processing in SQL. In other contexts, emulation can be forced but the lack of order in the SQL input process makes the results non-reproducible. SQL and the DATA step are very different environments, the emulation process in SQL is cumbersome and messy, and emulation is something one should do only if it is really necessary. Switching to DS2 and running in-database is a possible alternative to SQL; DS2 supports BY and first/last processing, but the results from DS2 are subject to the same constraints and are reproducible only in certain narrow contexts.

REFERENCES

Note: all URLs quoted or cited herein were accessed in March 2017.

Billings, T; (2012) IFC and IFN Functions: Alternatives to Simple DATA Step IF-THEN-ELSE, SELECT-END Code and PROC SQL CASE Statements. *Western Users of SAS Software Conference Proceedings*. URL: <https://www.lexjansen.com/wuss/2012/28.pdf>

Billings, T; (2017) An Easy Way to Split a SAS® Data Set into Unique and Non-Unique Row Subsets. *Western Users of SAS Software Conference Proceedings*. URL: https://www.lexjansen.com/wuss/2017/126_Final_Paper_PDF.pdf

Lafler, K. (2017). Advanced Programming Techniques with PROC SQL. *SAS Global Forum Conference Proceedings*. URL: <http://support.sas.com/resources/papers/proceedings17/0930-2017.pdf>

Disclaimer: The contents of the paper herein are solely the author's thoughts and opinions, which do not represent those of MUFG Union Bank N.A. The bank does not endorse, recommend, or promote any of the computing architectures, platforms, software, programming techniques or styles referenced in this paper.

CONTACT INFORMATION

A list of the author's SAS-related papers, including URLs for free access, is available at the URL (hosted by Google Drive): <https://goo.gl/uCUHoa>

Note: Your enterprise web filter might prevent access to this URL from work, in which case you will need to access via a personal device.

Thomas E. Billings
MUFG Union Bank, N.A.
San Francisco, CA 94104

Remote from:
Merritt Island, FL 32952

Phone: 321-453-4694
Email: tebillings@gmail.com

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.