

Paper 269-29

## DATA Step vs. PROC SQL: What's a neophyte to do?

Craig Dickstein, Tamarack Professional Services, Jackman, ME  
Ray Pass, Ray Pass Consulting, Hartsdale, NY

### ABSTRACT

*"What's all the buzz about Proc SQL? I am just now beginning to understand the power of the DATA step within the SAS® System and I am being told that it can all be done bigger and better in the SQL procedure. Do I have to use it? Should I use it? When and where is the appropriate use for PROC SQL?"*

For the beginner/novice SAS programmer, this paper will answer the above questions and attempt to demystify the *must* use, *should* use, and *nice to use* aspects of the DATA step as it compares and contrasts with Structured Query Language (SQL). General efficiencies of machine resources and programmer resources, as well as the proficiencies of the programmer, as choice criteria need to be considered. Business functions/needs and technology standards may also play a role as they apply to developing data management routines. Primary data management tasks to be considered are: creating, inputting, sub-setting, merging, transforming, and sorting. This paper will not discuss platform dependencies or benchmark efficiencies. Advanced data management/manipulation techniques will not be discussed. This is not an attempt to teach PROC SQL, but rather to inform the user wanting to make an educated choice of available techniques.

### INTRODUCTION

Structured Query Language (SQL) is a widely used language for retrieving and updating data in tables and/or views of those tables. It has its origins in and is primarily used for retrieval of tables in relational databases. PROC SQL is the SQL implementation within the SAS System. Prior to the availability of PROC SQL in Version 6.0 of the SAS System, DATA step logic and a few utility procedures were the only tools available for creating, joining, sub-setting, transforming, and sorting data.

As a Beginning Tutorial topic, this paper will attempt to compare and contrast the data management elements of the SQL procedure with analogous methods in the DATA step and other non-SQL base SAS techniques, and discuss the advantages and disadvantages of each for a given purpose. For the beginner SAS programmer, i.e., those without strong biases about one method or another, an attempt will be made to show several ways to accomplish the same task.

Finally, the benefits and advantages of either non-SQL base SAS techniques or PROC SQL will be discussed along with some thoughts on choosing the best tool for the job at hand.

### SIMILARITIES AND DIFFERENCES

#### ORIGINS

To understand the relative similarities and differences between PROC SQL and DATA step programming, one has only to recognize the origins of the two techniques. SQL has its roots in the world of relational databases whereas SAS was developed to manage and analyze "flat" files. For all intents and purposes, the following elements of the two languages are equivalent:

SAS	SQL
Data sets	Tables
Observations	Rows
Variables	Columns
Merge	Join
Extract	Query

SAS implemented a version of SQL so as to be able to access relational database tables and create SAS data sets as an outcome of such access. Since all RDBMSs are based on rectangular tables as their basic building block and SAS data sets are just another type of rectangular data table, PROC SQL works quite nicely with them.

#### SYNTAX

One important foundation in the understanding of the differences between PROC SQL and the DATA step (or non-SQL base SAS) is the syntactical construct of each method.

The PROC SQL statement, and any associated options, initiates a process much like any other base SAS procedure. At this point, Structured Query Language syntax takes over leaving behind the standard structure of SAS. A QUIT

statement is required to terminate a SQL step as opposed to the implicit step boundaries (PROC, DATA) or explicit step boundary (RUN) of non-SQL base SAS.

SQL procedure statements are divided into clauses. These clauses begin with known keywords and contain arguments (both for the statement and clauses) separated by commas. For example, the most basic SELECT statement contains the SELECT and FROM clauses. Other clauses are optionally available. Each clause may contain one or more components. Components can also contain other components. The analogous situation in the DATA step has statements beginning with a keyword and arguments are separated by blanks or unique symbols. Commas in PROC SQL and spaces in non-SQL base SAS separate "lists" of variables or tables.

Aliases are available in PROC SQL to be associated with field and table names; this concept is not known in non-SQL base SAS. They are defined by the AS clause. The AS clause is not always necessary (with table aliases) but explicit use is good coding technique – a matter of preference for the programmer.

SAS Log messages, while expressing the same thought, differ depending on the use of SQL or the DATA step. (E.g., table vs. data set, rows vs. observation, and columns vs. variables).

### CAVEATS

A data "view" is a named virtual data set or table that contains no data but rather describes a set of instructions for surfacing SAS data sets or RDBMS tables. This paper will assume interchangeable reference to tables and views.

Nuances of interactive SAS vs. batch SAS will not be discussed. No operating system specifics will be discussed.

Any mention of efficiencies is qualitative and not quantitative. It is left to the reader to consider these gains or losses within their own environment based on platform, file size, use of indices (or not), and use of RDBMSs.

The conventions used in the ensuing code examples are as follows: All SAS and SQL keywords, options, and formats are upper case. Any user defined "words" such as variable names, table/data set names, and data values are lower case. Generic lists and parentheticals are italicized. Ellipses (...) will signal incomplete code.

## CREATING DATA SETS

### NEW DATA SETS FROM NON-RDBMS SOURCE

One very distinct and important difference between PROC SQL and the DATA step is that the former cannot create tables from non-relational external data sources such as EBCDIC flat file structures (e.g., VSAM files, sequential data sets, partitioned data sets), spreadsheets, or ASCII files.

Although SQL can input data from in-stream record images, this is rarely used to initially create tables; it is rather usually used to modify limited portions of existing tables. The difficulty is demonstrated here. When faced with this particular task from within the SAS environment, DATA step coding is the method of choice. The following example shows the creation of a SAS data set from "in-stream" data, and the subsequent PRINTing of the data set.

#### DATA step:

```
DATA table1;
    INPUT    charvar1    $3.
           +1 charvar2    $1.
           +1 numvar1
           +1 numvar2    DATE7.
    ;
DATALINES;
me1 F 35786 10oct50
me3 M 57963 25jun49
fg6 M 25754 17jun47
fg7 F . 17aug53
;
PROC PRINT DATA=table1;
RUN;
```

In the above example, the data records stand alone as separate lines in the program. The keyword DATALINES alerts the internal processor that data will continue to follow until a line with a semicolon is encountered.

With SQL, the VALUES clause is the key element for INSERTing data values INTO a TABLE.

**PROC SQL:**

```

PROC SQL;
  CREATE TABLE table1
    ( charvar1 CHAR(3)
      , charvar2 CHAR(1)
      , numvar1 NUM
      , numvar2 NUM INFORMAT=DATE7.)
  ;
  INSERT INTO table1
    VALUES('me1','F',35786,'10oct50'd)
    VALUES('me3','M',57963,'25jun49'd)
    VALUES('fg6','M',25754,'17jun47'd)
    VALUES('fg7','F',.,'17aug53'd)
  ;
  SELECT *
    FROM table1;
QUIT;

```

The SELECT statement retrieves and displays all fields from the created table. This is equivalent to the PROC PRINT above.

**NEW DATA SETS FROM RELATIONAL DATABASES**

The SAS System supports reading, updating, and creating RDBMS tables with both non-SQL base SAS and PROC SQL.

For non-SQL base SAS coding techniques, the LIBNAME statement is crucial for the setup. If a RDBMS SAS/ACCESS product is installed, then the LIBNAME statement, accompanied by a RDBMS engine specification, can be used to associate a *libref* with a referenced RDBMS. Tables in the RDBMS can then be employed in SAS procedures or DATA steps as though they were SAS data sets, by using two-level SAS names.

**DATA step:**

```

LIBNAME olib ORACLE
      SAS/ACCESS-engine-connection-options
  ;
DATA table1;
  SET olib.oracle_table;
  IF var1 = "value1" THEN...
...

```

In the above example, *olib* is the user defined SAS libref that points to an Oracle database via the ORACLE access engine. Additional connection options (identification credentials) are required and are RDBMS engine specific. The two-level name in the SET statement then dynamically brings in a specific table from the Oracle database.

PROC SQL also works very nicely with RDBMS to both read and write external RDBMS tables. Using a SAS/ACCESS interface engine, the SQL Pass-Through Facility enables establishment and termination of connections with a RDBMS. You must have the requisite SAS/ACCESS software installed for your RDBMS. While connected, both query and non-query SQL statements can be sent to the RDBMS for processing.

The SQL procedure Pass-Through Facility performs the following tasks:

- ◆ Establish a connection with the RDBMS using a CONNECT statement and terminate the connection with the DISCONNECT statement.
- ◆ Send non-query RDBMS-specific SQL statements to the RDBMS using the EXECUTE statement.
- ◆ Retrieve data from the RDBMS via a SQL query with the CONNECTION TO component in the FROM clause of the SELECT statement.

To reduce data movement and translation, PROC SQL will use the Pass-Through Facility to take advantage of the capabilities of a RDBMS by passing it certain operations whenever possible. For example, before implementing a join, PROC SQL checks to see if the RDBMS can do the join. If it can, PROC SQL will pass the processing of the join to the RDBMS. If the RDBMS cannot do the join, PROC SQL does it.

Using the CONNECT/DISCONNECT construct, PROC SQL establishes a connection with the CONNECT statement, communicates between the RDBMS and the SAS environment and then terminates the connection with the DISCONNECT statement.

While “connected” to the RDBMS, database specific queries and non-query statements can be passed directly for execution in the native environment using the EXECUTE statement. Query results are then passed back to the SAS environment. The procedure also has the ability, when connected directly to a RDBMS, to send query statements to the RDBMS as an argument in the CONNECTION TO statement [e.g. CONNECTION TO *RDBMS-name (RDBMS-query)*]. This expands upon the power of DBMS manipulation from within SAS.

The discussion of RDBMS access engines and their use is beyond the scope of this paper. See the SAS/ACCESS documentation for more information. However, a simple example of accessing an Oracle table will suffice to make the point that RDBMS tables can be brought into a SAS program and treated as SAS data sets.

The following example shows how to connect to a particular database, pass queries for action by that RDBMS (the parenthetical code), return table results to SAS, and then disconnect from the database.

### PROC SQL:

```
PROC SQL;
  CONNECT TO rdbms AS dbref (dbms-definitions);
  CREATE TABLE table3 AS
  SELECT col1, col2, col3
     FROM CONNECTION TO dbref
        (SELECT col1, col2, col3
          FROM table1, table2
          WHERE table1.col1 = table2.col5
          ORDER BY col1
        );
  DISCONNECT FROM dbref;
QUIT;
```

### SORTING DATA

Given that a prime difference between PROC SQL and non-SQL base SAS processing is the way each handles the sorting of data, that functionality will be addressed next. The discussion will then be exemplified in most of the ensuing discussion's example code.

The SORT procedure is the base SAS utility that sorts SAS data sets by a single variable or list of variables (a nested sort). While a powerful yet simple procedure to code, it is potentially resource intensive. In the following example, *table1* is sorted and stored as *table2* with *var1* being the primary sort key and *var2* the secondary sort key, i.e., within each unique value of *var1*, the observations are sorted by *var2*.

### Non-SQL Base SAS:

```
PROC SORT DATA=table1
  OUT=table2 NODUPLICATES;
  BY var1 var2;
RUN;
```

One highly touted benefit of PROC SQL is its ability to process unsorted data and create tables in a sorted fashion. While it is arguable as to the efficiencies of PROC SORT vs. PROC SQL for this functionality, it is clear that PROC SQL requires far less attention to the detail of program design and considerably less coding.

When SELECTing or CREATEing tables with PROC SQL, the ORDER BY clause sorts the resultant data table by the specified columns. This is equivalent to using PROC SORT after a data set is created or using the OUT= option to store a new sorted instance of the data set. Tables do not need to be presorted for use with PROC SQL. Therefore, the use of the SORT procedure with PROC SQL programs is not needed as is usual and customary with other SAS procedures.

**PROC SQL:**

```
PROC SQL;
  CREATE TABLE table2 AS
  SELECT DISTINCT *
    FROM table1
    ORDER BY var1, var2;
QUIT;
```

A special use case of sort routines is the removal of duplicate data records. As demonstrated in the PROC SORT example above, the NODUPPLICATES option accomplishes this task. PROC SQL provides the same functionality with the DISTINCT component of the SELECT clause as is shown. Please note that this is not the same as the NODUPKEY option of PROC SORT, which checks for and eliminates observations with duplicate BY values.

Another special case of sorting should be mentioned here, the grouping of data for summarization. When requesting summary statistics on groups of data (discussed later in detail) the GROUP BY clause is used to define the "groups" within which summary statistics are desired. This ability is not available directly in the DATA step, but is generally equivalent to using BY statement processing in base SAS procedures such as MEANS and UNIVARIATE. As with any BY statement processing, the SAS data sets would need to be presorted. See the example on page 7.

The data do not have to be sorted in the order of the group-by values because PROC SQL handles sorting automatically. You can still however use the ORDER BY clause to specify the order in which rows are displayed in the resulting table. If you specify a GROUP BY clause in a query that does not contain a summary function, your clause is transformed into an ORDER BY clause and a message to that effect is written to the SAS log.

**JOINING DATA**

Combining data sets or tables is a common and frequent technique for data management prior to performing analytics. Data are combined primarily for two reasons:

- ◆ to combine files with similar variable composition layouts and different sets of observations
- ◆ to combine files with different variable composition layouts and similar sets of observations

Once combined, many actions can be taken on the resultant set of data:

- ◆ extracting a subset of records
- ◆ extracting a subset of variables
- ◆ calculating a new set of variables

Be aware that all SQL join processing operates by initially producing a full "Cartesian product". Simply stated this is an intermediate table made up of *all* combinations of *all* rows from *all* the selected contributing tables. Selection criteria are then applied to this intermediary table as coded in the PROC SQL to yield the table subset that is actually desired. The intermediate full table is generally *not* the required result, and this Cartesian processing is different from the way a DATA step MERGE operates. However, if this total crossing of *all* records from *all* data is exactly what is needed, PROC SQL is hands down the way to go since this is what happens by default. The important lesson here - ***Know thy data!!***

**CONCATENATING**

The concatenating or "stacking" of data files is done for the purpose of making one large file from two or more files of a *similar variable* structure. It can be done as a simple concatenation of one after another (stacking) or based on some key set of variables (interleaving). Interleaving is important if the resultant file needs to be in a known order. To accomplish interleaving in some situations, concatenating and then sorting may be more efficient (e.g., for large files) than pre-sorting the component parts.

The DATA step SET statement, the APPEND procedure, or the APPEND statement of PROC DATASETS are three non-SQL base SAS methods for simply stacking data. The APPEND route is the more efficient technique providing that no DATA step processing is required. An analogous technique is available in PROC SQL with the OUTER UNION statement.

The use of a BY variable in conjunction with the SET statement accomplishes an interleaving of data sets. The data sets being "set" need to be pre-sorted by the key variable(s). The resultant data set is then in the known sort order. PROC APPEND does not interleave data sets.

The following examples demonstrate the simple stacking of two data sets with similar variable structure.

**DATA step:**

```
DATA table3;
    SET table1 table2;
```

**PROC SQL:**

```
PROC SQL;
    CREATE TABLE table3 AS
        SELECT *
            FROM table1
        OUTER UNION CORRESPONDING
        SELECT *
            FROM table2;
QUIT;
```

Performing an OUTER UNION is very similar to the DATA step with a SET statement referencing two or more data sets. The OUTER UNION concatenates the intermediate results from the table-expressions. Thus, the resultant table for the query-expression contains all the rows produced by the first table-expression followed by all the rows produced by the second table-expression.

As with the DATA step SET statement, if a table has fewer columns than the one(s) with which it is being 'set', PROC SQL extends the rows of the smaller table with columns containing missing values of the appropriate data type, before it builds the resulting table.

The CORRESPONDING, or CORR, keyword causes PROC SQL to match the columns in table-expressions by name and not by ordinal position. Without the use of CORRESPONDING, matching columns from the contributing columns with the same name would be created in separate columns in the result table. This table would then include duplicate columns (names and contents).

**MATCH MERGING**

Merging of data is done for the purpose of combining records from two or more source tables into a new data file with a new combined record layout. For simplicity sake, we will assume that the merging is done based on a key variable(s). There may be reasons to do otherwise (non-keyed merging), but this technique is fraught with potential problems and is rarely a desired result.

Also of major importance is that, except for the key variable(s), the data to be merged should usually contain different fields. If the data sets being merged have commonly named variables, the left-most value(s) is(are) overwritten with the right-most value. This may be an intended result for "updating" purposes. The UPDATE statement is a special case of merging in the DATA step and requires just two data sets, a master and a transaction. This functionality will not be discussed here.

Again, with DATA step merging, the data sets must be sorted *prior* to merging with a BY variable. With PROC SQL this is not a requirement. In fact, the key variable on which the data are joined with SQL does *not* need to have the same variable name in all data sets being merged. Employing the RENAME= data set option will correct for dissimilar BY variables in the DATA step merging of data sets.

Practically speaking, several types of joins or merges are used:

- ◆ **Inner join** – Retrieves only matching rows that meet the selection criteria.
- ◆ **Outer joins** – Outer joins can be left, right, or full and will return all rows from one or more files in the selection, depending upon the selection criteria.
  - ◆ **Left join** – The first dataset mentioned (left) is the master and the second (right) is the transactional. All rows from the "master" are returned with information, as available, added from the "transaction" when rows from both tables "match" on the matching column(s).
  - ◆ **Right join** – The second dataset mentioned is the master and the first is the transactional.
  - ◆ **Full join** – The resultant table has records from all data sets, including "matching" rows as well as "non-matching" rows.

With the MERGE statement and DATA step processing, the IN= data set option, in conjunction with IF/THEN logic is employed to control the type of join.

**DATA step:**

```

DATA table3;
  MERGE table1 (IN=l)
        Table2 (IN=r);
  BY keyvar;
  IF l AND r;          *** inner join ;

```

The expression "IF l;" would result in a left join and the expression "IF r;" would yield a right join. The expression "IF l or r;", or the lack of an IF statement altogether, would produce a full outer join.

With PROC SQL, several key clauses are used to control the join type: LEFT JOIN, RIGHT JOIN, OUTER JOIN, or INNER JOIN. The WHERE clause or ON clause describes the conditions (SQL-expression) under which the rows in the Cartesian product are kept or eliminated in the result table. WHERE is used to select rows from inner joins. ON is used to select rows from inner or outer joins.

**PROC SQL:**

```

PROC SQL;
  CREATE TABLE table3 AS
  SELECT *
  FROM   table1 AS l
  INNER JOIN table2 AS r
        ON l.keyvar=r.keyvar;
QUIT;

```

Whenever possible, PROC SQL passes the processing of joins to the RDBMS rather than doing the joins itself. This enhances performance.

**SUBSETTING DATA**

The subsetting of data is the bread and butter of data preparation for subsequent analysis. Specific record types may be selected, and needed variables retained or dropped as required by the ensuing analysis.

**ROWS**

SAS data set observations are generally selected based on the value of a variable or set of variables. For example, only females of a certain age group may be required. All others are discarded.

In non-SQL base SAS this may be accomplished with a variety of techniques:

- ◆ The WHERE statement in procedures, DATA steps, or data set options
- ◆ The subsetting IF statement
- ◆ IF / THEN / DELETE code

**DATA step:**

```

DATA table2;
  SET table1(WHERE=(var1=value1));

DATA table4;
  SET table3
  IF var1=value1 AND
     var2 IN (value-list);

DATA table7;
  MERGE table5 table6;
  BY var1;
  IF MOD(var4,3) NE 0 THEN DELETE;

```

In the SQL procedure, several simple techniques are also available:

#### PROC SQL:

```
PROC SQL;
  CREATE TABLE table3 AS
  SELECT var1, var2, var3, var4
     FROM table1 AS a
        , table2 AS b
  WHERE a.var1=b.var1
        AND a.var1 IN (value-list);
QUIT;

PROC SQL;
  CREATE TABLE table4 AS
  SELECT *
     FROM table3;
  DELETE
     FROM table4
  WHERE var1 IS MISSING;
QUIT;
```

#### COLUMNS

As with the retention or deletion of observations, variables are easily included in or excluded from a table.

The DROP and KEEP statements or data set options are workhorses of base SAS. They are most efficiently used as data set options.

#### DATA step:

```
DATA table2;
  SET table1(DROP=var4-var6);

DATA table4;
  MERGE table2 table3;
  BY keyvar;
  KEEP keyvar var1 var2;
```

In the SQL procedure, it is a simple matter of SELECTing only those columns that are required.

#### PROC SQL:

```
PROC SQL;
  CREATE TABLE table1 AS
  SELECT var1, var2, var3, var6
     FROM table2;
QUIT;
```

Being another base SAS procedure working with SAS data sets, albeit in a unique and hybrid way, PROC SQL can apply most of the SAS data set options. The options such as KEEP= and DROP= can work with the SAS tables being read (SELECT FROM) or written (CREATE TABLE). As in DATA step code, the options are separated by spaces and are enclosed in parentheses following immediately after the table name.



**PROC SQL:**

```
PROC SQL;
  CREATE TABLE table2(DROP=var4-var6) AS
  SELECT *
    FROM table1;
QUIT;
```

**TRANSFORMING DATA**

A common data management function is to create new data by transforming the value(s) of one or several variables or by summarizing data to key points of information stored in smaller tables.

**CREATING NEW VARIABLES**

In non-SQL base SAS, new variables are created from existing ones via the assignment statement. Assignment statements generally use SAS functions or mathematical expressions to create (or re-create) variables from other variables or constants.

**DATA step:**

```
DATA table2;
  SET table1;
  newvar1 = oldvar2/oldvar3;
  newvar2 = SUBSTR(charvar5,3,5);
```

PROC SQL provides similar constructs within the SELECT statement. Please note that the order of the expression is reversed.

**PROC SQL:**

```
PROC SQL;
  CREATE TABLE table2 AS
  SELECT var1
         , var2
         , oldvar2/oldvar3      AS newvar1
         , SUBSTR(charvar5,3,5) AS newvar2
    FROM table1;
QUIT;
```

**SUMMARIZING DATA**

Simple summarization techniques are available in PROC SQL via several useful functions, some of which are shown in the following table, along side the analogous non-SQL base SAS procedure. The SQL procedure supports most of the functions found in non-SQL base SAS. In conjunction with the GROUP BY clause, a table is produced with the requisite group level statistic. Although convoluted, similar results may be obtained with DATA step programming.

SQL Function	Base SAS
COUNT(field)	PROC FREQ
MIN(field)	PROC MEANS
MAX(field)	PROC MEANS
SUM(field)	PROC MEANS

Let's look at a simple example of finding the sum, non-missing count, and missing value count for a numeric variable within a classification variable.

**Non-SQL Base SAS:**

```
PROC SORT DATA=table1;
  BY catvar1;
```

```

PROC MEANS DATA=table1;
  BY catvar1;
  VAR var4;
  OUTPUT OUT=table2
         SUM=totvar4
         N=cntvar4
         NMISS=missvar4 ;

```

### PROC SQL:

```

PROC SQL;
  CREATE TABLE table2 AS
  SELECT catvar1,
         , SUM(var4) AS totvar4
         , COUNT(var4) AS cntvar4
         , NMISS(var4) AS missvar4
  FROM table1
  GROUP BY catvar1
  ORDER BY catvar1
  ;
QUIT;

```

Counts could have been obtained from PROC FREQ, but for the purposes of also acquiring sums, PROC MEANS was employed. Note that the GROUP BY functionality in PROC SQL is analogous to BY statement processing in PROC MEANS. The ORDER BY clause performs the functionality of PROC SORT: returning the resultant table in a known sort order. If GROUP BY is omitted, all the rows in the table or view are considered to be a single group.

### COMPARATIVE FUNCTIONALITY

The following table is intended to provide the reader with a summary of the above-discussed comparative techniques.

Function	Non-SQL Base SAS STATEMENT / OPTION	PROC SQL STATEMENT / CLAUSE
Create a table	DATA	CREATE TABLE
Create a table from raw data	INPUT	INSERT
Add columns	Assignment statement	SELECT ... AS ...
Drop columns	DROP KEEP	SELECT
Add rows	OUTPUT	INSERT INTO
Delete rows	WHERE IF/THEN DELETE	DELETE FROM
Sorting	PROC SORT	ORDER BY
De-dupe records	NODUPLICATE	DISTINCT
Establish a connection with a RDBMS	LIBNAME	CONNECT DISCONNECT

Function	Non-SQL Base SAS STATEMENT / OPTION	PROC SQL STATEMENT / CLAUSE
Send a RDBMS-specific non-query SQL statement to a RDBMS		CONNECTION TO
Concatenating	SET	OUTER JOIN
Match merging	MERGE / SET BY IF in1 IF in2 IF in1 & in2	FROM LEFT JOIN RIGHT JOIN FULL JOIN WHERE / ON
Rename column	RENAME	AS
Displaying resultant table	PROC PRINT	SELECT
Summarizing	PROC / BY	GROUP BY

### BENEFITS/ADVANTAGES

The following list of benefits is not intended to be complete as much as a more general comment on issues of which the beginning SAS programmer should be aware. The lists are in no particular priority order.

#### PROC SQL

- ◆ PROC SQL provides the combined functionality of the DATA step and several base SAS procedures.
- ◆ Less complex and lengthy, but not as legible, code can be written in PROC SQL.
- ◆ PROC SQL code may execute faster for smaller tables.
- ◆ PROC SQL code is more portable for non-SAS programmers and non-SAS applications.
- ◆ PROC SQL processing does not require explicit code to presort tables.
- ◆ PROC SQL processing does not require common variable names to join on, although same type and length are required.
- ◆ By default, a PROC SQL SELECT statement prints the resultant query; use the NOPRINT option to suppress this feature.
- ◆ Knowledge of relational data theory opens the power of SQL for many additional tasks.
- ◆ PROC SQL processing forces attention to resultant data set structures, as SQL is unforgiving of "errors of design".
- ◆ Efficiencies within specific RDBMS are available with Pass-thru code for the performance of joins.
- ◆ Use of aliases for shorthand code may make some coding tasks easier.

#### NON-SQL BASE SAS

- ◆ DATA step set operators can handle more data sets at a time than PROC SQL outer joins.
- ◆ Non-SQL techniques can open files for read and write at the same time.
- ◆ Customized DATA step report writing techniques (DATA \_NULL\_) are more versatile than using PROC SQL SELECT clauses.
- ◆ The straightforward access to RDBMS tables as if they were SAS data sets negates the need to learn SQL constructs.
- ◆ Input of non-RDBMS external sources is easier.

## CHOOSING A TECHNIQUE

Here comes the tricky part. A premise of this paper from the outset was that there are many ways to 'skin a cat' using SAS software, with advantages and disadvantages to each. We have demonstrated that this is certainly true when comparing PROC SQL and non-SQL base SAS for data management techniques.

While elegance vs. functionality is always a consideration, the choice of one technique over another should be made based on the following criteria:

- ◆ Familiarity – Given the fast pace of today's business world, demanding faster and more accurate answers, it is best that the programmer employ those techniques that are most familiar and comfortable.
- ◆ Correctness – Care in the choice of tools should be a dominant theme. While all techniques will provide results, not all results are correct relative to the process design. See as a clear example the discussion of "Cartesian products" above.
- ◆ Maintenance – An often-overlooked feature of program development is its maintainability. While succinct code is often desirable, emphasis should be placed on clear, concise (but not necessarily terse), and maintainable code.
- ◆ Efficiency of human resources – Efficient and effective effort is defined as "doing the right things right". From a programmer perspective this relates to the comfort level and skill sets of the individual programmer.
- ◆ Efficiency of processing resources - Any mention of computing efficiencies in this paper has been qualitative and not quantitative. It is left to the readers to consider these gains or losses within their own environment based on platform, file size, use of indices (or not), and use of RDBMSs.
- ◆ Futures – What is the ultimate goal of the effort? Where will the code reside? These are pertinent design questions that should be considered when moving data or porting code to and from an RDBMS environment.

While not expressly discussed herein, it can be understood that data management and extraction with PROC SQL can become as powerful (and convoluted) as non-SQL base SAS programming. Sub-queries, complex grouping, and complicated summarizations can be accomplished with either facility.

## CONCLUSION

So... What's a neophyte to do? Expand your horizons! Explore alternate methodologies! Find out about those other ways to skin that proverbial SAS cat! Don't get tied to the only way you know now. Heeding this advice will start you on your journey to better data manipulation techniques on the small scale and to optimal methodologies when using the SAS System in general.

## REFERENCES

The reader is directed to the following papers for additional and important information on the contrast between non-SQL base SAS programming and PROC SQL.

*How Should I Combine My Data, Is the Question*, Thompson, S. and Sharma, A., Proceedings of the 12<sup>th</sup> Annual NorthEast SAS Users Group Conference, Washington, DC, 1999.

*Alternatives to Merging SAS Data Sets ... But be Careful*, Wieczkowski, M., Proceedings of the 12<sup>th</sup> Annual NorthEast SAS Users Group Conference, Washington, DC, 1999.

*PROC SQL for DATA Step Die-Hards*, Williams, C.S., Proceedings of the 15<sup>th</sup> Annual NorthEast SAS Users Group Conference, Buffalo, NY, 2002.

Scerbo, M., Dickstein, C., and Wilson, A. (2001). *Health Care Data and the SAS System*, Cary, NC: SAS Institute Inc. *SAS SQL Procedure User's Guide, Version 8*, Cary, NC: SAS Institute Inc.

## CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Craig Dickstein  
207-668-7607  
[tamarack-llc@att.net](mailto:tamarack-llc@att.net)

Ray Pass  
914-693-5553  
[raypass@att.net](mailto:raypass@att.net)

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.