Paper 185-2008

PROC SQL for DATA Step Die-hards

Christianna S. Williams, Abt Associates Inc, Durham, NC

ABSTRACT

PROC SQL can be rather intimidating for those who have learned SAS data management techniques using exclusively the DATA Step. However, when it comes to data manipulation, SAS often provides more than one method to achieve the same result, and SQL provides another valuable tool to have in one's repertoire. Further, Structured Query Language is implemented in many widely used relational database systems with which SAS may interface, so it is a worthwhile skill to have from that perspective as well.

This paper will present a series of increasingly complex examples and hands-on exercises. In each case I will demonstrate the DATA Step method with which users are probably already familiar, followed by SQL code that will accomplish the same data manipulation. The simplest examples include subsetting variables (columns, in SQL parlance) and observations (rows), while the most complex situations will include MERGEs (JOINS) of several types and the summarization of information over multiple observations for BY groups of interest. This approach can help to clarify for which situations the DATA STEP method or, conversely, PROC SQL is better suited. The reader should come away with a good grounding in Data Step to PROC SQL translation – it is my intent that even die-hard Data Step-ers may find themselves "thinking in SQL"!

INTRODUCTION

The DATA step is a real workhorse for virtually all SAS users. Its power and flexibility are probably among the key reasons why the SAS language has become so widely used by data analysts, data managers and other "IT professionals". However, at least since version 6.06, PROC SQL, which is the SAS implementation of Structured Query Language, has provided another extremely versatile tool in the base SAS arsenal for data manipulation. Still, for many of us who began using SAS prior to the addition of SQL or learned from hardcore DATA step programmers, change may not come easily. We are often too pressed for time in our projects to learn something new or venture from the familiar, even though it may save us time and make us stronger programmers in the long run. Often SQL can accomplish the same data manipulation task with considerably less code than more traditional SAS techniques.

This paper is designed to be a relatively painless introduction to PROC SQL for users who are already quite adept with the DATA step. Several examples of row selection, grouping, sorting, summation and combining information from different data sets will be presented. For each example, I'll show a DATA step method (recognizing that there are often multiple techniques to achieve the same result) followed by an SQL method. Throughout the paper, when I refer to "DATA step methods", I include under this term other base SAS procedures that are commonly used for data manipulation (e.g. SORT, SUMMARY). In each code example, SAS keywords are in ALL CAPS, while arbitrary user-provided parameters (i.e. variable and data set names) are in lower case.

THE DATA

First, a brief introduction to the data sets. Table 1 describes the four logically linked data sets, which concern the hospital admissions for twenty make-believe patients. The variable or variables that uniquely identify an observation within each data set are indicated in bold; the data sets are sorted by these keys. Complete listings are included at the end of the paper. Throughout the paper, it is assumed that these data sets are located in a data library referenced by the libref EX.

Table 1. Description of data sets for examples

Data set	Variable	Description
admissions	**pt_id**	patient identifier
	admdate	date of admission
	disdate	date of discharge
	hosp	hospital identifier
	bp_sys	systolic blood pressure (mmHg)
	bp_dia	diastolic blood pressure (mmHg)
	dest	discharge destination
	primdx	primary diagnosis (ICD-9)
	md	admitting physician identifier

Data set	Variable	Description
patients	**id**	patient identifier
	lastname	patient last name
	firstname	patient first name
	sex	gender (1=M, 2=F)
	birthdate	date of birth
	primmd	primary physician identifier
hospitals	**hosp_id**	hospital identifier
	hospname	hospital name
	town	hospital location
	nbeds	number of beds
	type	hospital type
doctors	**md_id**	physician identifier
	hospadm	hospital at which MD has admitting privileges
	lastname	physician last name

EXAMPLE 1:   SUBSETTING VARIABLES (COLUMNS)

In this first, extremely simple example, we just want to select three variables from the ADMISSIONS data set.

```
                               DATA step code:
DATA selvar1 ;
    SET ex.admissions (KEEP = pt_id admdate disdate);
RUN;
```

In the DATA step, the KEEP= option on the SET statement does the job.

```
                               SQL code:
PROC SQL;
    CREATE TABLE selvar2 AS
         SELECT pt_id, admdate, disdate
               FROM ex.admissions  ;
QUIT;
```

The SQL procedure is invoked with the PROC SQL statement. SQL is an interactive procedure, in which RUN has no meaning. QUIT forces a step boundary, terminating the procedure. An SQL table in SAS is identical to a SAS data set. The output table could also be a permanent SAS data set; in such case, it would be referenced by a two-level name (e.g. EX.SELVAR2). A few other features of this simple statement are worth noting. First, the variable names are separated by commas rather than spaces; this is a general feature of lists in SQL – lists of tables, as we'll see later are also separated by commas.   Second, the AS keyword signals the use of an alias; in this case the table name SELVAR2 is being used as an alias for the results of the query beginning with the SELECT clause. We'll see other types of aliases later. Third, the FROM clause names what entity we are querying. Here it is a single input data set (EX.ADMISSIONS), but it could also be multiple data sets, a query, a view (either a SAS view or a SAS/ACCESS view), or a table in an external database. Examples of the first two types will be presented below.

SQL can also be used to write reports, in which case the statement above would begin with the SELECT clause. The resulting report looks much like output from PROC PRINT. SAS views, which are stored queries, can also be created with SQL. To do this, the keyword TABLE in the CREATE statement above would simply be replaced with the keyword VIEW. In this paper, since I am focusing on the generation of new data sets meeting desired specifications, virtually all the SQL statements will begin with "CREATE TABLE…".

One final point before we move on to some more challenging examples: interestingly, although the results of the DATA step and the PROC SQL are identical (neither PROC PRINT nor PROC COMPARE reveal any differences), slightly different messages are generated in the log.

> ***For the DATA step:***
>
> ```
> NOTE: The data set WORK.SELVAR1 has 25 observations and 3 variables.
> ```
>
> ***For PROC SQL:***
>
> ```
> NOTE: Table WORK.SELVAR2 created, with 25 rows and 3 columns.
> ```

This demonstrates a distinction in the terminology that stems from the fact that SQL originated in the relational data-base arena, while, of course, the DATA step evolved for "flat file" data management. Table 2 shows these equivalencies.

Table 2. Equivalent SAS terms between the DATA step and PROC SQL

DATA step	PROC SQL
data set	table
observation	row
variable	column

EXAMPLE 2:   SELECTING OBSERVATIONS (ROWS)

Almost all of the rest of the examples involve the selection of certain observations (or rows) from a table or combina-tions of tables. Here we simply want to select admissions to the Veterans Administration hospital (HOSP EQ 3 on the ADMISSIONS data set).

> ***DATA step code:***
>
> ```
> DATA vahosp1 ;
>     SET ex.admissions (WHERE = (hosp EQ 3));
> RUN;
> ```

The WHERE clause on the SET statement is used to choose those observations for which the hospital identifier cor-responds to the VA. This is more efficient than a subsetting IF, though the result is the same.

> ***SQL code:***
>
> ```
> PROC SQL FEEDBACK;
>     CREATE TABLE vahosp2 AS
>         SELECT *
>             FROM ex.admissions
>         WHERE hosp EQ 3;
> QUIT;
> ```

Here, the WHERE clause performs the same function as the subsetting IF above. Note that it is still part of the CREATE statement. A few additional features of SQL are demonstrated here in this simple query. First, the * is a "wild card" syntax, which essentially means "Select all the columns". The FEEDBACK option on the PROC SQL statement requests an expansion of the query in the log. Useful in conjunction with the asterisk wild card, this results in the following statement in the SAS log:

```
NOTE: Statement transforms to:
select   ADMISSIONS.PT_ID,   ADMISSIONS.ADMDATE,   ADMISSIONS.DISDATE,
ADMISSIONS.MD,  ADMISSIONS.HOSP,  ADMISSIONS.DEST,  ADMISSIONS.BP_SYS,
ADMISSIONS.BP_DIA, ADMISSIONS.PRIMDX
from EX.ADMISSIONS
where ADMISSIONS.HOSP=3;

NOTE: Table WORK.VAHOSP2 created, with 6 rows and 9 columns.
```

A subset of variables is shown in the output at the top of the next page.

```
Example 2:  Selecting observations: VA Admits

PT_ID    ADMDATE    DISDATE      HOSP
------------------------------------
003  15MAR2006  15MAR2006        3
008  01OCT2006  15OCT2006        3
008  26NOV2006  28NOV2006        3
014  17JAN2007  20JAN2007        3
018  01NOV2006  15NOV2006        3
018  26DEC2006  08JAN2007        3
```

EXAMPLE 3: CREATING A NEW VARIABLE

In this example we want to create a variable called DXGRP that categorizes the primary diagnosis into one of three categories (myocardial infarction [MI], congestive heart failure [CHF] or other), based on the ICD-9 code.

```
                          DATA step code:

DATA grouping ;
    SET ex.admissions ;
LENGTH dxgrp $5 ;
        IF primdx EQ: '410' THEN dxgrp = 'MI' ;
    ELSE IF primdx EQ: '428' THEN dxgrp = 'CHF';
    ELSE dxgrp = 'other' ;
RUN;
```

The useful EQ: comparison operator (or, equivalently =:) allows us to select all values of PRIMDX that begin with the specified string of characters, regardless of subsequent characters.

```
                          SQL code:

PROC SQL;
    CREATE TABLE grouping2 AS
        SELECT *,
            CASE
                WHEN primdx LIKE '410%' THEN 'MI'
                WHEN primdx LIKE '428%' THEN 'CHF'
                ELSE 'other'
            END AS dxgrp
        FROM ex.admissions;
QUIT;
```

Here, the CASE clause of PROC SQL is used in conjunction with the LIKE keyword and the % wildcard to define the new variable DXGRP. Below is a partial listing of the results.

```
EXAMPLE 3:  Creating a character variable – DXGRP

PT_ID    ADMDATE    PRIMDX   DXGRP
---------------------------------
001  07FEB2006  410.0    MI
001  12APR2006  428.2    CHF
001  10SEP2006  813.90   other
001  06JUN2007  428.4    CHF
003  15MAR2006  431      other
004  18JUN2006  434.1    other
005  19JAN2006  411.81   other
005  10MAR2006  410.9    MI
005  10APR2006  411.0    other
007  28JUL2006  155.0    other
007  08SEP2006  155.0    other
```

There is one tricky thing to remember about the CASE clause. It cannot be used to assign new values to a variable that already exists on the data set – it is only for creating "new" variables.

EXAMPLE 4: SELECTING ROWS BASED ON A CREATED VARIABLE

In this example we want to create a variable corresponding to the number of days of the hospital stay and select only those stays with duration of at least 14 days. Usually, both the admission date and discharge date are considered days of stay.

```
                              DATA Step code:
DATA twowks1 ;
    SET ex.admissions (KEEP = pt_id hosp admdate disdate) ;

    ATTRIB los LENGTH=4 LABEL='Length of Stay';
    los = (disdate - admdate) + 1;

    IF los GE 14 ;
RUN;
                                SQL code:

PROC SQL;
    CREATE TABLE twowks2 AS
      SELECT pt_id, hosp, admdate, disdate,
             (disdate-admdate) + 1 AS los LENGTH=4 LABEL='Length of Stay'
      FROM ex.admissions
           WHERE CALCULATED los GE 14;
QUIT;
```

Here, we see the creation of a new column and the assignment of a column alias (LOS). Attributes can also be added; they could include a FORMAT as well. There is also one more subtle feature here: the CALCULATED key-word is required to indicate that the column LOS doesn't exist on the input table (EX.ADMISSIONS) but is calculated during the query execution.

```
Example 4:  Selecting observations based on created variable

PT_ID   HOSP    ADMDATE       DISDATE     LOS
---------------------------------------------
001      1     12APR2006    25APR2006     14
007      2     28JUL2006    10AUG2006     14
008      3     01OCT2006    15OCT2006     15
009      2     15DEC2006    04JAN2007     21
018      3     01NOV2006    15NOV2006     15
018      3     26DEC2006    08JAN2007     14
020      1     08OCT2007    01NOV2007     25
```

On the other hand, it is not required to assign an alias to a calculated column. The following would be perfectly valid and would select the same observations:

```
SELECT  pt_id, hosp, admdate, disdate, (disdate - admdate) + 1
   FROM ex.admissions
        WHERE (disdate - admdate) + 1 GE 14;
```

However, SAS will assign an arbitrary, system-dependent variable name to this column in the resulting table. If this column had a LABEL, it would print at the top of the column in the output, though the underlying variable name would still be the undecipherable _TEMA001 or something similar.

EXAMPLE 5: SELECTING ROWS IN ONE TABLE BASED ON INFORMATION FROM ANOTHER TABLE

Returning to the example of selecting admissions to the Veterans Administration hospital, let's say we didn't know which value of the HOSP variable corresponded to the VA hospital. The information that provides a "cross-walk" between the hospital identifier code and the hospital name is in the HOSPITALS data set.

```
                              DATA Step Code:
PROC SORT DATA = ex.admissions OUT=admits;
BY hosp ;
RUN;
```

```
DATA vahosp1d (DROP = hospname) ;
    MERGE admits (IN=adm)
          ex.hospitals (IN=va  KEEP = hosp_id hospname
                               RENAME = (hosp_id=hosp)
                               WHERE = (hospname EQ: 'Veteran'));
    BY hosp ;
    IF adm AND va;

RUN;

PROC SORT;
BY pt_id admdate;
RUN;
```

We first need to sort the ADMISSIONS data set by the hospital code, and then merge it with the HOSPITALS data set, renaming the hospital code variable and selecting only those observations with a hospital name beginning "Veteran". If we want the admission to again be in ascending order by patient ID and admission date, another sort is required. The resulting data set is the same as in Example 2.

```
                           PROC SQL Code:

PROC SQL ;
    CREATE TABLE vahosp2d AS
          SELECT *
    FROM ex.admissions
          WHERE hosp IN
                (SELECT hosp_id
                 FROM ex.hospitals
                 WHERE hospname LIKE "Veteran%")
    ORDER BY pt_id, admdate ;
QUIT;
```

This procedure contains an example of a subquery, or a query-expression that is nested within another query-expression. The value of the hospital identifier (HOSP) on the ADMISSIONS data set is compared to the result of a subquery of the HOSPITALS data set. Using IN (rather than EQ) in the WHERE clause allows for the possibility that the subquery might return more than a single value (i.e. if more than one hospital had a name beginning "Veteran"). Note that no columns are added to the resulting table from the HOSPITALS data set, although this could be done too, as we'll see in a later example. No explicit sorting is required for this subquery to work. The ORDER BY clause dictates the sort order of the output data set. The output is identical to that shown for Example 2A.

EXAMPLE 6: USING SUMMARY FUNCTIONS

Our next task is to count the number of admissions for each of the patients with at least one admission. We also want to calculate the minimum and maximum length of stay for each patient.

```
                           DATA Step Code:
DATA admsum1 ;
    SET ex.admissions ;
    BY pt_id;

    ** (1) Initialization;
    IF FIRST.pt_id THEN DO;
         nstays = 0;
         minlos = .;
         maxlos = .;
    END;

    ** (2) Accumulation;
    nstays = nstays + 1;
    los = (disdate - admdate) + 1;
    minlos = MIN(OF minlos los) ;
    maxlos = MAX(OF maxlos los) ;
```

```
    ** (3) Output;
    IF LAST.pt_id THEN OUTPUT ;
    RETAIN nstays minlos maxlos ;
    KEEP pt_id nstays minlos maxlos ;
RUN;
```

We process the input data set by PT_ID. The DATA step has three sections. First, when the input observation is the first one for each subject, we initialize each of the summary variables. Next, in the accumulation phase we increment our counter and determine if the current stay is the longest or shortest for this patient. The RETAIN statement permits these comparisons. Finally, when it is the last input observation for a given PT_ID, we output an observation to our summary data set, keeping only the ID and the summary variables. If we kept any other variables, their values in the output data set would be the values they had for the last observation for each subject, and the output data set would still have one observation for each patient in the ADMISSIONS file (i.e. 14).

*** PROC SQL code: ***

```
PROC SQL;
    CREATE TABLE admsum2 AS
          SELECT pt_id, COUNT(*) AS nstays,
                  MIN(disdate - admdate + 1) AS minlos,
                  MAX(disdate - admdate + 1) AS maxlos
    FROM ex.admissions
    GROUP BY pt_id ;
QUIT;
```

Two new features of PROC SQL are introduced here. First, the GROUP BY clause instructs SQL what the groupings are over which to perform any summary functions. Second, the summary functions include COUNT, which is the SQL name for the N or FREQ functions used in other SAS procedures. The COUNT(*) syntax essentially says count the rows for each GROUP BY group. The summary columns are each given an alias. The output is below.

```
Example 6:  Using Summary Functions


PT_ID     NSTAYS     MINLOS     MAXLOS
---------------------------------
001         4         2          14
003         1         1           1
004         1         7           7
005         3         4           9
007         1        14          14
008         3         3          15
```

If we selected any columns other than the grouping column(s) and the summary variables, the resulting table would have a row for every row in the input table (i.e. 23) with the summary information duplicated across rows with a common value of the grouping variable (PT_ID), and we'd get the following messages in the log:

```
NOTE:  The  query  requires  remerging  summary  statistics  back  with  the
original data.
NOTE: Table WORK.ADMSUM2 created, with 23 rows and 5 columns.
```

Sometimes this "re-merging" is useful (see Example 4b below), but it is not what we want for this situation.

EXAMPLE 7: SELECTION BASED ON SUMMARY FUNCTIONS

Let's say we want to identify potential blood pressure outliers. We'd like to select all those observations that are two standard deviations or further from the mean.

*** DATA Step Code: ***

```
PROC SUMMARY DATA= ex.admissions ;
    VAR bp_sys ;
    OUTPUT OUT=bpstats  MEAN(bp_sys)= mean_sys  STD(bp_sys) = sd_sys ;
RUN;
```

```
DATA hi_sys1 ;
    SET bpstats (keep=mean_sys sd_sys)
        ex.admissions ;
    IF _N_ EQ 1 THEN DO;
        high = mean_sys + 2*(sd_sys) ;
        low = mean_sys - 2*(sd_sys) ;
        DELETE;
    END;
    RETAIN high low;
    IF (bp_sys GE high) OR (bp_sys LE low) ;
    DROP mean_sys sd_sys high low ;
RUN;
```

PROC SUMMARY generates the desired statistics. We then concatenate this one-observation data set (BPSTATS) with ADMISSIONS, RETAINing the high and low cutoffs to allow the necessary comparison to select potential outliers.

<div align="center">

PROC SQL Code:

</div>

```
PROC SQL ;
    CREATE TABLE hi_sys2 AS
        SELECT * FROM ex.admissions
    WHERE (bp_sys GE
        (SELECT MEAN(bp_sys)+ 2*STD(bp_sys)
            FROM ex.admissions))
    OR    (bp_sys LE
        (SELECT MEAN(bp_sys) - 2*STD(bp_sys)
            FROM ex.admissions));
QUIT;
```

In the SQL version, the summary functions are used in two similar subqueries of the same table to generate the values against which the systolic blood pressure for each observation in the outer query is compared. There is no GROUP BY clause because we are generating the summary values for the entire data set. The result is shown below.

```
Example 7:  Selection based on Summary Functions

PT_ID   ADMDATE      BP_SYS     BP_DIA      DEST
--------------------------------------------
001    12APR2006      230        101         1
003    15MAR2006       74         40         9
009    15DEC2006      228         92         9
```

EXAMPLE 8: SELECTION BASED ON SUMMARY FUNCTION WITH "RE-MERGE"

This example adds a small twist to the prior one by requiring that we select admissions with extreme systolic blood pressure values within groups defined by discharge destination. The variable DEST is 1 for those who are discharged home, 2 for those discharged to a rehabilitation facility and 9 for those who die.

<div align="center">

DATA Step Code:

</div>

```
PROC SUMMARY DATA= ex.admissions NWAY;
    CLASS dest ;
    VAR bp_sys ;
    OUTPUT OUT=bpstats2 MEAN(bp_sys)=mean_sys STD(bp_sys)=sd_sys ;
RUN;

PROC SORT DATA = ex.admissions OUT=admissions;    BY dest ;
RUN;

DATA hi_sys3 ;
    MERGE admissions (KEEP = pt_id bp_sys bp_dia dest)
          bpstats2 (KEEP = dest mean_sys sd_sys);
    BY dest ;
    IF bp_sys GE mean_sys + 2*(sd_sys) OR
       bp_sys LE mean_sys - 2*(sd_sys) ;

FORMAT mean_sys sd_sys 6.2;
RUN;
```

To obtain the desired summary statistics for blood pressure we again use PROC SUMMARY. However, this time we include a CLASS statement to obtain the mean and standard deviation for each category of discharge destination and include the NWAY option so the BPSTATS2 data set does not include the overall statistics. The ADMISSIONS data set must be sorted by DEST before merging in the destination-specific means and standard deviations. A subsetting IF selects the desired observations.

```
                              PROC SQL Code:

PROC SQL;
    CREATE TABLE hi_sys4 AS
          SELECT pt_id, bp_sys, bp_dia, dest,
                  MEAN(bp_sys) AS mean_sys FORMAT=6.2,
                  STD(bp_sys)  AS sd_sys   FORMAT=6.2
    FROM ex.admissions
    GROUP BY dest
          HAVING bp_sys GE (mean_sys + 2*sd_sys)
              OR bp_sys LE (mean_sys – 2*sd_sys) ;
QUIT;
```

In some ways the SQL code for this example, in which the statistics are generated and the selection of rows are made separately for each BY group, is simpler than the last one where the process was done for the sample as a whole. This example doesn't require a subquery. Rather it relies on a "re-merging" of the summary statistics for each GROUPing back with the ungrouped data, permitting the row-by-row comparisons needed to select the outliers. A new keyword is introduced here as well. HAVING acts on groups in a manner analogous to the way a WHERE clause operates on rows. A HAVING expression usually is proceeded by a GROUP BY clause, which defines the group that the HAVING expression evaluates, and the query must include one or more summary functions.

```
Example 8: Select using Summary Functions with re-merge

PT_ID BP_SYS  BP_DIA  DEST  MEAN_SYS  SD_SYS
---------------------------------------------
001    230     101     1    165.82    30.48
018    199       9     2    151.09    21.28
```

EXAMPLE 9: IDENTIFYING DUPLICATES

This example demonstrates another use of a HAVING expression. We wish to select observations from the DOCTORS data set that are not unique with respect to the physician identifier. In other words we want to pull out all the records for the doctors who have admitting privileges at more than one hospital. We'd like them in order by the physician's last name.

```
                              DATA Step Code:

DATA selmd1 ;
    SET ex.doctors (KEEP = md_id lastname hospadm
                      RENAME = (hospadm=hospital));
    BY md_id ;
    IF NOT (FIRST.md_id AND LAST.md_id) ;
RUN;

PROC SORT DATA=selmd1;   BY lastname hospital ;
RUN;
```

Processing BY md_id with this subsetting IF will produce the desired result.

```
                              PROC SQL Code:

PROC SQL ;
    CREATE TABLE selmd2 AS
          SELECT md_id, lastname, hospadm AS hospital
    FROM ex.doctors
          GROUP BY md_id
              HAVING COUNT(*) GE 2
    ORDER BY lastname, hospital ;
QUIT;
```

Applying the GROUP BY clause, the query first counts how many rows are associated with each doctor. The HAVING expression then selects the rows that meet the following condition: being part of a group having more than one row. The output is shown below.

```
Example 9:  Identifying duplicates

MD_ID    LASTNAME          HOSPITAL
-----------------------------------
7803     Avitable                2
7803     Avitable                3
1972     Fitzhugh                1
1972     Fitzhugh                2
3274     Hanratty                1
3274     Hanratty                2
3274     Hanratty                3
2322     MacArthur               1
2322     MacArthur               3
```

EXAMPLE 10: CREATION OF TWO DATA SETS FROM ONE

For the next example suppose we want to create separate data sets for the admissions in 2006 and 2007.

```
                              DATA Step Code:

DATA admit06 admit07 ;
    SET ex.admissions ;
        IF YEAR(admdate) = 2006 THEN OUTPUT admit06;
  ELSE IF YEAR(admdate) = 2007 THEN OUTPUT admit07;
RUN;
```

This is readily accomplished by including two data set names in the DATA statement and directing observations to the appropriate output data set based on the value of the YEAR function.

```
                              PROC SQL Code:

PROC SQL ;
    CREATE TABLE admit06 AS
          SELECT * FROM ex.admissions
               WHERE YEAR(admdate) = 2006;
    CREATE TABLE admit07 AS
          SELECT * FROM ex.admissions
               WHERE YEAR(admdate) = 2007;
QUIT;
```

This invocation of PROC SQL includes two separate but nearly identical CREATE statements, one for each output table. Almost all of the functions that are available in the DATA step are also available in PROC SQL. A partial listing of each of the output tables is shown below.

```
Example 10: Two data sets from one (2006)

pt_id    admdate    md      hosp
-------------------------------
001  07FEB2006  3274        1
001  12APR2006  1972        1
001  10SEP2006  3274        2
003  15MAR2006  2322        3
004  18JUN2006  7803        2

Example 10: Two data sets from one (2007)

pt_id    admdate    md      hosp
-------------------------------
001  06JUN2007  3274        2
010  30NOV2007  2322        1
014  17JAN2007  7803        3
015  25MAY2007  4003        5
015  17AUG2007  4003        5
```

10

EXAMPLE 11: CONCATENATION

If we were starting with separate data sets for each year and wanted to combine into a single data set, ordered as in our original data set (by PT_ID and ADMDATE), the following code would serve.

```
                              DATA Step Code:
DATA alladm1 ;
    SET admit06 admit07 ;
    BY pt_id ;
RUN;
```

The BY statement is needed to ensure the desired ordering. Without it, all the 2006 admissions would precede all the 2007 admissions.

```
                              PROC SQL Code:
PROC SQL ;
    CREATE TABLE alladm2 AS
         SELECT * FROM admit06
             UNION
         SELECT * FROM admit07
    ORDER BY pt_id;
QUIT;
```

Generally the UNION set operator concatenates the component data sets so that the resulting table is in the same sort order as each of the original components. However, the ORDER BY clause ensures that this is the case.

EXAMPLE 12: INNER JOIN OF TWO TABLES

A join combines data from two or more tables to produce a single result table; the table resulting from an inner join contains rows that have one or more matches in the other table(s).

```
                              DATA Step Code:
DATA admits1 ;
    MERGE ex.admissions (IN=adm KEEP = pt_id admdate disdate hosp md)
         ex.patients   (IN=pts KEEP = id lastname sex primmd
                                RENAME = (id=pt_id));
    BY pt_id ;
    IF adm AND pts;
RUN;
```

Selection based on the IN= temporary variables does the trick. Note that this produces the desired result partly because although there may be multiple admissions for each patient, the PATIENTS data set has only one observation for each value of the key variable PT_ID. The information on each record for a given PT_ID in the PATIENTS data set is replicated onto each observation in the output data set.

```
                              PROC SQL code:
PROC SQL ;
    CREATE TABLE admits2 AS
         SELECT pt_id, admdate, disdate, hosp, md, lastname, sex, primmd
    FROM ex.admissions AS a,
         ex.patients AS b
              WHERE a.pt_id = b.id
    ORDER BY a.pt_id, admdate ;
QUIT;
```

The table aliases A and B are used here to clarify which ID variables are coming from which data set. They are not required here because there are no columns being selected here that exist on both input data sets. Note that the AS keyword is not required, but it emphasizes that an alias is being assigned. The code above might be more commonly used for a simple inner join, but the following syntax produces the same result.

11

```
                            Alternative PROC SQL code:
PROC SQL ;
    CREATE TABLE admits2 AS
        SELECT pt_id, admdate, disdate, hosp, md, lastname, sex, primmd
    FROM ex.admissions INNER JOIN
        ex.patients
            ON pt_id = id
    ORDER BY pt_id, admdate ;
QUIT;
```

This is also an example of an "equijoin" because the selection criteria is equality of a column in one table with a column in the second table. SAS MERGEs are always equijoins. In the abbreviated output below, only a subset of the 25 selected rows and 8 columns are shown.

```
Example 12:  Inner Join of two tables


PT_ID   ADMDATE   HOSP   MD   LASTNAME   PRIMMD
----------------------------------------------
001  07FEB2006    1   3274   Williams     1972
001  12APR2006    1   1972   Williams     1972
001  10SEP2006    2   3274   Williams     1972
001  06JUN2007    2   3274   Williams     1972
003  15MAR2006    3   2322   Gillette      .
004  18JUN2006    2   7803   Wallace      4003
005  19JAN2006    1   1972   Abbott       1972
005  10MAR2006    1   1972   Abbott       1972
005  10APR2006    2   1972   Abbott       1972
```

EXAMPLE 13: JOIN OF THREE TABLES WITH ROW SELECTION

Let's make it a little trickier! Let's say we now wish to identify patients who died in the hospital (DEST = 9); we want our resulting data set to include their age at death and the number of beds in the hospital. This requires obtaining information from three of our tables, with differing key fields.

```
                               DATA Step Code:
DATA died1 (RENAME = (disdate=dthdate)) ;
    MERGE ex.admissions (IN=dth KEEP = pt_id disdate hosp dest
                            WHERE = (dest=9))
        ex.patients (IN=pts KEEP = id birthdate RENAME = (id=pt_id));
    BY pt_id ;
    IF dth AND pts ;

    agedth = FLOOR((disdate-birthdate)/365.25) ;

     DROP dest birthdate ;
RUN;

PROC SORT DATA=died1;    BY hosp;
RUN;

DATA died1b ;
    MERGE  died1  (IN=dth RENAME=(hosp=hosp_id))
          ex.hospitals (IN=hsp KEEP=hosp_id nbeds);
    BY hosp_id ;

    IF dth AND hsp ;
    DROP hosp_id;
RUN;

PROC SORT;
    BY pt_id ;
RUN;
```

The DATA step version requires two DATA steps and two SORTs.

<div style="text-align:center">***PROC SQL code:***</div>

```
PROC SQL ;
    CREATE TABLE died2 AS
         SELECT pt_id, nbeds, disdate AS dthdate,
                 INT((disdate-birthdate)/365.25) AS agedth
    FROM ex.admissions, ex.hospitals, ex.patients
         WHERE (pt_id = id) AND (hosp = hosp_id) AND dest EQ 9
    ORDER BY pt_id ;
QUIT;
```

With PROC SQL, we can query the combination of the three tables in one step because there is no requirement of a single key that links all of the inputs. The output data set contains 3 rows and the desired variables.

```
Example 13:  Join of three tables with row selection

PT_ID    DTHDATE     AGEDTH     NBEDS
-----------------------------------
001  12JUN2007         75         645
003  15MAR2006         78        1176
009  04JAN2007         88         645
```

EXAMPLE 14: LEFT OUTER JOIN

A left outer join is an inner join of two or more tables that is augmented with rows from the "left" table that do not match with any rows in the "right" table(s). For this example we want to produce a table that has a row for each hospital with an indicator of whether there were any admissions at that hospital.

<div style="text-align:center">***DATA Step Code:***</div>

```
PROC SORT DATA = ex.admissions (KEEP = hosp)
           OUT=admits (RENAME=(hosp=hosp_id)) NODUPKEY;
    BY hosp ;
RUN;

DATA hosps1 ;
    MERGE ex.hospitals (IN=hosp)
          admits (IN=adm);
    BY hosp_id ;
         IF hosp ;
               hasadmit = adm ;
RUN;
```

If the duplicates were not removed from the ADMISSIONS data set, the output data set would have multiple observations for each hospital; hence NODUPKEY is used on the SORT. The temporary boolean IN= variable is made permanent in the MERGE step to create our indicator of having at least one record in the ADMISSIONS data set.

<div style="text-align:center">***PROC SQL code:***</div>

```
PROC SQL ;
    CREATE TABLE hosps2 AS
         SELECT DISTINCT a.*, hosp IS NOT NULL AS hasadmit
    FROM ex.hospitals a LEFT JOIN
         ex.admissions b
      ON a.hosp_id = b.hosp  ;
QUIT;
```

The keyword DISTINCT causes SQL to eliminate duplicate rows from the resulting table. The expression "hosp IS NOT NULL AS hasadmit" assigns the alias HASADMIT to a new column whose value is TRUE (i.e. 1) if a given HOSP_ID from the HOSPITALS table has a matching HOSP value in the ADMISSIONS table. The resulting data set – shown on the top of the next page – has one row for each hospital and a binary indicator of admissions as desired.

<div style="text-align:center">*13*</div>

```
Example 14: Left Outer Join

HOSP_ID  HOSPNAME                        HASADMIT
---------------------------------------------
1  Big University Hospital              1
2  Our Lady of Charity                  1
3  Veterans Administration              1
4  Community Hospital                   1
5  City Hospital                        1
6  Children's Hospital                  0
```

EXAMPLE 15: INNER JOIN WITH A SUBQUERY

One of the items combined in a join can itself be a query. In this case we want to identify and select the admissions for which patients were treated by their primary physicians (PRIMMD on the PATIENTS data set). We want the output data set to include the doctor's name and the patient's name.

```
                            DATA Step Code:
DATA primdoc (DROP = primmd);
    MERGE ex.admissions (IN=adm KEEP = pt_id admdate disdate hosp md)
          ex.patients (IN=pts KEEP = id lastname primmd
                        RENAME = (id=pt_id));
    BY pt_id ;
        IF adm AND pts AND (md EQ primmd) ;
RUN;

PROC SORT DATA=primdoc;  BY md; RUN;

DATA doctors ;
    SET ex.doctors (KEEP = md_id lastname);
    BY md_id ;
        IF FIRST.md_id ;
RUN;

DATA primdoc1a ;
    MERGE primdoc (IN=p RENAME=(lastname=ptname md=md_id))
        doctors (RENAME = (lastname=mdname));
    BY md_id ;
        IF p ;
RUN;

PROC SORT DATA=primdoc1a ;
BY pt_id admdate;
RUN;
```

The first DATA step above selects the admissions for which patients saw their primary physicians. The second DATA step eliminates duplicate records for the same physician. If this were not done, the final MERGE would be a many-to-many merge and would not produce the desired result. (Note that we could also have done a SORT with NODUPKEY). The final DATA step (with MERGE) simply adds the physician name to the selected admissions. Both LASTNAME variables are RENAMEd to prevent the physician name from overwriting the patient name.

```
                            PROC SQL Code:
PROC SQL ;
    CREATE TABLE primdoc2 AS
        SELECT pt_id, admdate, disdate, hosp, md_id,
            b.lastname AS ptname,
            c.lastname AS mdname
        FROM ex.admissions a, ex.patients b,
            (SELECT DISTINCT md_id, lastname
                FROM ex.doctors) c
                    WHERE (a.pt_id EQ b.id)  AND
                        (a.md EQ b.primmd) AND
                        (a.md EQ c.md_id)
    ORDER BY a.pt_id, admdate ;
QUIT;
```

The third "table" listed in the FROM clause is itself a query that selects non-duplicate physician ID's and names from the DOCTORS data set. The result of this subquery can be aliased just like a table, and here the aliases (b and c) are required so that the two LASTNAME columns can be distinguished. The ultimate row selection is very straight-forward. Sometimes for a complicated query like this it is helpful to break it down into separate queries.

```
Example 15: Inner Join with a subquery

PT_ID    ADMDATE    PTNAME       MDNAME
-------------------------------------------
001    12APR2006   Williams    Fitzhugh
005    19JAN2006   Abbott      Fitzhugh
005    10MAR2006   Abbott      Fitzhugh
005    10APR2006   Abbott      Fitzhugh
007    28JUL2006   Nickelby    Hanratty
007    08SEP2006   Nickelby    Hanratty
010    30NOV2007   Alberts     MacArthur
018    01NOV2006   Baker       Fitzhugh
018    26DEC2006   Baker       Fitzhugh
```

EXAMPLE 16: A CORRELATED SUBQUERY

For the final example, we'll take it up one more notch in complexity! A correlated subquery is a subquery for which the values returned by the inner query depend on values in the current row of the outer query. For example, we want to display the names of physicians who had admissions to the VA hospital.

```
                            DATA Step Code:
PROC SORT DATA = ex.admissions (KEEP=md hosp)  OUT = admits;
    BY md;
RUN;

PROC SORT DATA = ex.doctors OUT=doctors  NODUPKEY ;
    BY md_id ;
RUN;

DATA vadocs1 (DROP = hosp);
    MERGE doctors (IN=docs KEEP=md_id lastname)
          admits (IN=adm  WHERE=(hosp = 3) RENAME = (md=md_id)) ;
    BY md_id;
        IF docs AND adm AND FIRST.md_id ;
RUN;

PROC SORT;
    BY lastname;
RUN;
```

First, we need to sort the ADMISSIONS data set by its link to the PHYSICIAN data set and eliminate duplicate records from the PHYSICIAN data set. Then we merge the VA admission records into the physician data; we must again "de-dup" because some of these physicians have more than one admission, and the information we are interested in would be redundant. Finally, we SORT once more so the resulting data set is in alphabetical order by physician LASTNAME.

```
                            PROC SQL Code:
PROC SQL;
    CREATE TABLE vadocs2 AS
        SELECT DISTINCT md_id, lastname
    FROM ex.doctors AS d
        WHERE 3 IN
            (SELECT hosp FROM ex.admissions AS a
                WHERE d.md_id = a.md)
    ORDER BY lastname;
QUIT;
```

Because the subquery refers to a column in the outer query (MD_ID), it is evaluated for each row of the DOCTORS table. So, for each row of the DOCTORS table that has a match in the ADMISSIONS table the WHERE clause

checks if 3 equals HOSP is TRUE; if so, the row is selected. Four MD's had admissions to the VA hospital, as shown in the output below.

```
Example 16:  A correlated subquery

MD_ID  LASTNAME
------------------
7803  Avitable
1972  Fitzhugh
3274  Hanratty
2322  MacArthur
```

CONCLUSION

I hope that the examples presented in this paper have convinced you that PROC SQL is an extremely versatile tool for the manipulation of data sets. Row selection, summarization, the combination of information from multiple input sources, and the ordering of the output can often be achieved in a single statement! Another compelling reason for becoming comfortable with SQL is that many information systems store data in foreign databases, such as ORACLE, Microsoft Access or SQL Server. If these data are to be manipulated and analyzed in SAS, frequently PROC SQL provides the link (e.g. through ODBC).

Perhaps seeing some familiar DATA step techniques followed by a call to the SQL procedure that achieves the same result will give you the impetus to try SQL or dig into it a bit more deeply. I'll close with three observations that I hope will provide some encouragement.

First, it is always useful to have many different techniques to draw on when tackling a challenging data management task. And using a new project or assignment as an opportunity to learn some new methods makes you a more valuable employee – and probably a more fulfilled one as well.

Second, on the technical side, the most complicated nested query can usually be broken down into manageable parts – start from the "inside" (the most nested expressions) and work your way out. While it may be possible to do it all in one statement, you don't have to. Try making each level of nesting into a separate SELECT statement, using aliases liberally, with a final statement that connects the results of these simpler statements. Once this is working, you can start building the parts back together again – if you wish.

Finally, in constructing these examples, I was struck that using SQL forces one to think about data sets in a slightly different way, focusing more on the *relationships* among tables than the structure of any one table. In fact, it makes one realize that a database is defined not only by the component tables but just as importantly by the linkages among them. This broadened perspective can provide insight into building better databases as well as writing better programs to access and manipulate them.

ACKNOWLEDGMENTS

Many thanks to Peter Charpentier, Evelyne Gahbauer and Virginia Towle for their careful reading of and extremely helpful comments on earlier versions of this paper.

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ®indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.

CONTACT INFORMATION

I welcome your comments or questions.

Christianna S. Williams, PhD
Email: Christianna_Williams@abtassoc.com

APPENDIX: COMPLETE LISTING OF EXAMPLE DATA SETS

```
EX.ADMISSIONS
pt_id  admdate    disdate    md    hosp  dest  bp_sys  bp_dia  primdx
001    07FEB2006  08FEB2006  3274  1     1     188     85      410.0
001    12APR2006  25APR2006  1972  1     1     230     101     428.2
001    10SEP2006  19SEP2006  3274  2     2     170     78      813.90
001    06JUN2007  12JUN2007  3274  2     9     185     94      428.4
003    15MAR2006  15MAR2006  2322  3     9      74     40      431
004    18JUN2006  24JUN2006  7803  2     2     140     78      434.1
005    19JAN2006  22JAN2006  1972  1     1     148     84      411.81
005    10MAR2006  18MAR2006  1972  1     1     160     90      410.9
005    10APR2006  14APR2006  1972  2     1     150     89      411.0
007    28JUL2006  10AUG2006  3274  2     2     136     72      155.0
007    08SEP2006  15SEP2006  3274  2     2     138     71      155.0
008    01OCT2006  15OCT2006  3274  3     1     145     74      820.8
008    26NOV2006  28NOV2006  2322  3     2     135     76      V54.8
008    10DEC2006  12DEC2006  2322  9     2     132     78      V54.8
009    15DEC2006  04JAN2007  1972  2     9     228     92      410.1
010    30NOV2007  06DEC2007  2322  1     1     147     84      E886.3
012    12AUG2006  16AUG2006  4003  5     1     187     106     410.52
014    17JAN2007  20JAN2007  7803  3     1     162     93      414.10
015    25MAY2007  06JUN2007  4003  5     2     142     81      820.8
015    17AUG2007  24AUG2007  4003  5     2     138     79      038.2
016    25JUL2007  30JUL2007  7803  2     1     189     101     412.1
018    01NOV2006  15NOV2006  1972  3     2     170     88      428.1
018    26DEC2006  08JAN2007  1972  3     2     199     93      428.1
020    04JUL2007  08JUL2007  2998  4     1     118     75      414.0
020    08OCT2007  01NOV2007  2322  1     2     162     99      434.0

EX.PATIENTS
id   sex   primmd   lastname    firstname   birthdate
001  1     1972     Williams    Hugh        10AUG1931
002  2     1972     Franklin    Susan       17MAR1938
003  1     .        Gillette    Michael     02JUL1927
004  1     4003     Wallace     Geoffrey    25MAY1925
005  2     1972     Abbott      Celeste     31AUG1940
006  1     2322     Mathison    Anthony     12APR1908
007  1     3274     Nickelby    Nicholas    07FEB1909
008  2     4003     Lieberman   Marianne    09NOV1944
009  2     3274     Jacobson    Frances     15SEP1918
010  2     2322     Alberts     Josephine   14OCT1948
011  2     1972     Erickson    Karen       04NOV1926
012  1     7803     Collins     Elizabeth   16JUN1935
013  1     4003     Greene      Riley       03AUG1946
014  2     8034     Marcus      Emily       14DEC1941
015  2     3274     Zakur       Hannah      .
016  1     1972     DeLucia     Antonio     17JUN1913
017  1     2322     Cohen       Adam        17APR1931
018  1     1972     Baker       Shelby      13FEB1947
019  2     4003     Wallace     Judith      01FEB1933
020  2     7803     Nelson      Caroline    07AUG1915

EX.HOSPITALS
HOSP_ID  HOSPNAME                   TOWN            NBEDS    TYPE
1        Big University Hospital    New Mitford       841    1
2        Our Lady of Charity        North Mitford     645    2
3        Veterans Administration    West Mitford     1176    3
4        Community Hospital         Derbyville        448    1
5        City Hospital              New Mitford      1025    1
6        Children's Hospital        East Mitford      239    2

EX.DOCTORS
MD_ID  LASTNAME          HOSPADM
1972   Fitzhugh             1
1972   Fitzhugh             2
2322   MacArthur            1
2322   MacArthur            3
2998   Rosenberg            4
3274   Hanratty             1
3274   Hanratty             2
3274   Hanratty             3
4003   Colantonio           5
7803   Avitable             2
7803   Avitable             3
```