Proc SQL versus The Data Step

JoAnn Matthews, Highmark Blue Shield, Pittsburgh, PA

ABSTRACT
Most data within organizations is stored in relational databases. Structured Query Language (SQL) has evolved as the standard for accessing, updating, and modifying data stored in relational databases. PROC SQL is a powerful procedure available in SAS® that can minimize keystrokes and maximize CPU resources. PROC SQL syntax looks very similar to SQL and can be used in place of traditional SAS data steps.

In this Hands–On Workshop, PROC SQL will be compared to the traditional data step. First, PROC SQL will be compared to a simple data step. SELECT coupled with FROM, WHERE, GROUP BY and ORDER BY statements will be demonstrated. CASE statements will be compared to IF THEN ELSE logic. Summary functions using SUM, MIN, AVG, MAX and COUNT will be demonstrated. The true power of PROC SQL will be demonstrated with UNION ALL, UNION DISTINCT and JOINS. Three types of join will be discussed, EQUIJOINS, based on one common value, INNER JOINS, that discard all rows from the resultant table not having a corresponding row in the source table, and OUTER JOINS, joins that exclude unmatched data. This workshop will also be helpful to those who are new to SAS or Display Manager in Version 9.

INTRODUCTION

Given the complex data that is used by many large corporations, most corporations store their data in relational data base systems or DMBS, such as Oracle, DB2, Access, Teradata tables, or permanent SAS® datasets. Structured Query Language (SQL) is defined is a standard interactive programming language for getting information from and updating a database. Many database products support SQL commonly called '*sequel.*' SQL has evolved as the standard for defining, accessing, updating, manipulating and modifying data stored in relational databases SQL is carefully controlled and standardized by guidelines set up by the American National Standards Institute (ANSI).

SQL is one of the most commonly used query languages for relational data base management systems (DBMS). A data base management system (DBMS) is a set of programs used to define, administer and process data in a database. DBMS programs run on a multitude of platforms including the mainframe, minicomputers and personal computers. In many instances DBMS's reside on a several platforms within an organization, and contain all three classes of machine, mainframe, mini and PC. A DBMS that runs on multiple platforms, large and small, is called '*scalable*'. The flow of information within the DBMS is from the user at the keyboard, to the interface, to the program, through the DBMS, and finally to the database. SQL is a set of user friendly commands that are flexible and easy to understand. The query is written in syntax that is like English, making it easy to interpret. A query is a question to the database. The beauty of SQL is that when the data in the database satisfies your query, SQL retrieves the data in the most efficient way possible. SQL is "*nonprocedural*" meaning that you state what you want, and the DBMS decides how best to retrieve the answer. In contrast, a procedural language, such as FORTRAN, BASIC, C, or Java, requires that you to write a "*procedure*" to program what you want and need.

PROC SQL and SAS

SQL married SAS sometime after Version 5. PROC SQL was developed by SAS to make use of some of the powerful components of SQL. PROC SQL is so similar to SQL that anyone who has programmed using standard query language will quickly understand the basics. The PROC SQL procedure available in SAS can minimize keystrokes and maximize CPU resources, particularly when working with very large databases. There are some subtle differences with naming and security conventions between ANSI SQL statements and the SAS SQL procedure but these differences are so subtle that the average SAS user need not be concerned about them. For anyone who has ever used SQL, the syntax of the PROC SQL statement looks similar to standard SQL and can be used in place of the traditional SAS data step.

When we compare the SQL procedure to the data step, there are some differences that should be taken into consideration. One big difference is that you do not need to use the run after the procedure. PROC SQL will run without the run statement. If you include the RUN statement in the PROC SQL, SAS ignores the RUN and will give you a warning statement.

- You do need a QUIT statement at the end of the PROC SQL procedure.
- You can execute a number of SQL steps within the procedure without having to repeat the PROC SQL statement. This is an advantage when manipulating data. In traditional data step processing, each step must begin with a procedure.
- In PROC SQL, variables (columns) are separated with commas, not blanks as in a data step.
- The Select statement in PROC SQL outputs the data automatically, so that you do not have to execute a PROC PRINT statement to see your output.
- One of the most significant differences in the two methods is that when using PROC SQL, the table does not have to be sorted. When using traditional data step processing, each table must be sorted before merging. This may not be a major issue when using small datasets, but when using millions of observations, presorting a dataset can be a major space issue. Using PROC SQL can be a major advantage when working with large volumes of data.

BASIC SQL SYNTAX

The most common components used in the SQL procedure are: *SELECT, UPDATE, DELETE, CREATE, DROP, INSERT, RESET, VALIDATE, ALTER and DESCRIBE.* In alphabetical order, these SQL statements are described:

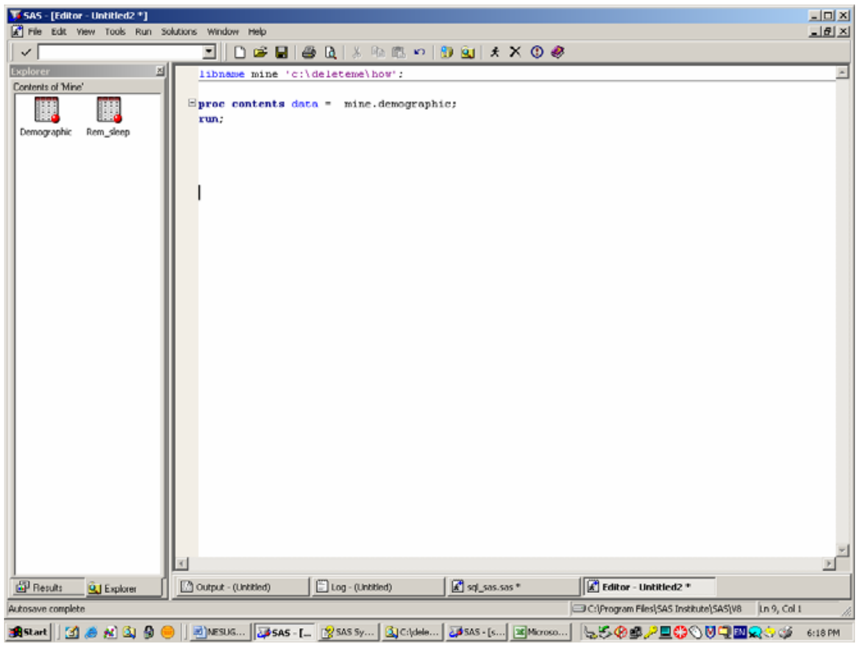
ALTER	<is used to add, delete or alter columns in a table>
CREATE	<is used to create a view or table>
DELETE	<is used to delete rows from a table>
DESCRIBE	<is used to explain how a view has been defined>
DROP	<is used to delete a table or view>
INSERT	<is used to add rows to a table>
UPDATE	<is used to update a table>
VALIDATE	<is used to validate SQL syntax>
DESCRIBE	<is used to describe the table>
RESET.	<is used to add, change or alter options on the PROC SQL line>
SELECT	<is used to generate a report and is the most important statement as it evaluates the query, formats rows and sends the output to the output window>

Within the SELECT statement there are subcomponents:

SELECT	<FIRST AND MOST IMPORTANT COMPONENT – identifies what your query will contain>
FROM	<is used to identify the source table for the query>
WHERE	<is used to select specific rows or subset the query>
ORDER BY	<is used to define the order of the data>
GROUP BY	<is used to group the data>

Let’s begin with a table of sleep participants’ demographic information. The name of this table is mine.demographic and it resides on the c:drive. This table contains 20 observations (rows) and 7 variables (columns). In order to confirm this data, let’s do a PROC CONTENTS in a traditional data step. Go to the program editor and type the following code, then highlight the code, put your cursor over the run icon, and execute the code.
:

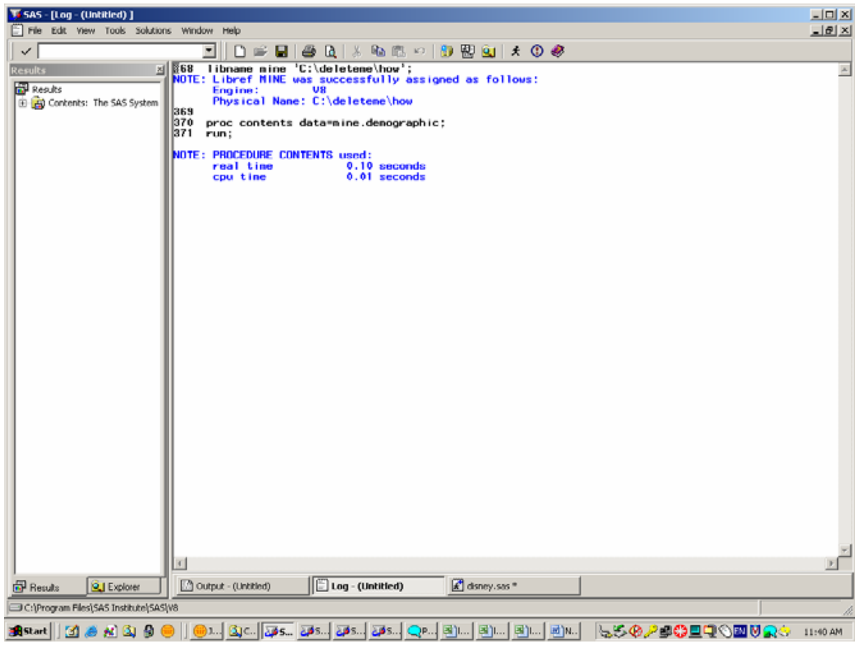
```
libname mine 'c:\deleteme\how';
proc contents data=mine.demographic;
run;
```



```
libname mine 'c:\deleteme\how';

proc contents data =  mine.demographic;
run;
```

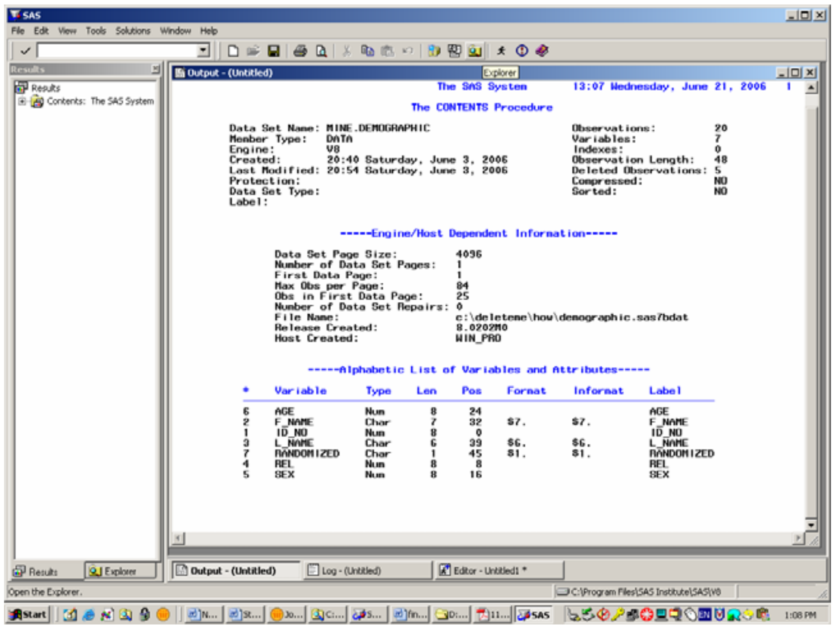
The PROC CONTENTS tells you everything you need to know about your data including the number of observations, the number of variables, the names and characteristics of the variables. PROC CONTENTS is a robust method to learn characteristics of the dataset. If the data is stored in a permanent dataset, running a PROC CONTENTS allows the programmer to become familiar with a dataset. And a word of caution, get in the habit of checking your log every single time you run a bit of code. Checking for errors and warnings is critical.

Now check the log. The LOG window contains valuable information about your session, as well as notes about the code that you submitted. When doing a PROC CONTENTS, the log only tells you that the procedure ran. The output has all the stats about the dataset.



```
368   libname mine 'C:\deleteme\how';
NOTE: Libref MINE was successfully assigned as follows:
      Engine:        V8
      Physical Name: C:\deleteme\how
369
370   proc contents data=mine.demographic;
371   run;

NOTE: PROCEDURE CONTENTS used:
      real time           0.10 seconds
      cpu time            0.01 seconds
```

Now go to the output window. Your output from the PROC CONTENTS should look like this:

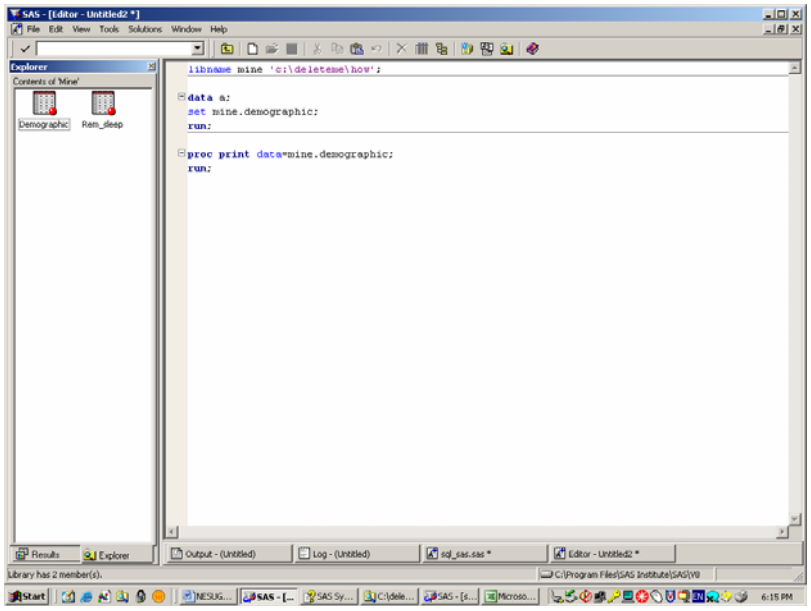


```
                    The SAS System       13:07 Wednesday, June 21, 2006   1

                         The CONTENTS Procedure

Data Set Name: MINE.DEMOGRAPHIC                    Observations:          20
Member Type:   DATA                                Variables:             7
Engine:        V8                                  Indexes:               0
Created:       20:40 Saturday, June 3, 2006        Observation Length:    48
Last Modified: 20:54 Saturday, June 3, 2006        Deleted Observations:  5
Protection:                                        Compressed:            NO
Data Set Type:                                     Sorted:                NO
Label:

                 -----Engine/Host Dependent Information-----

      Data Set Page Size:         4096
      Number of Data Set Pages:   1
      First Data Page:            1
      Max Obs per Page:           84
      Obs in First Data Page:     25
      Number of Data Set Repairs: 0
      File Name:                  c:\deleteme\how\demographic.sas7bdat
      Release Created:            8.0202M0
      Host Created:               WIN_PRO

             -----Alphabetic List of Variables and Attributes-----

 #    Variable      Type    Len    Pos    Format    Informat    Label
 ____________________________________________________________________
 6    AGE           Num       8     24                          AGE
 2    F_NAME        Char      7     32    $7.       $7.         F_NAME
 1    ID_NO         Num       8      0                          ID_NO
 3    L_NAME        Char      6     39    $6.       $6.         L_NAME
 7    RANDOMIZED    Char      1     45    $1.       $1.         RANDOMIZED
 4    REL           Num       8      8                          REL
 5    SEX           Num       8     16                          SEX
```

Notice the top right portion of the output. Observations and Variables are described. In this dataset we have 20 observations (rows) and 7 variables (columns). Observe the bottom of the output where attributes of the variables are described. Let's print all the variables using a traditional data step. Go into the program editor, clear the log and output window, and type the following code, then execute the code by using the submit icon on your toolbar.

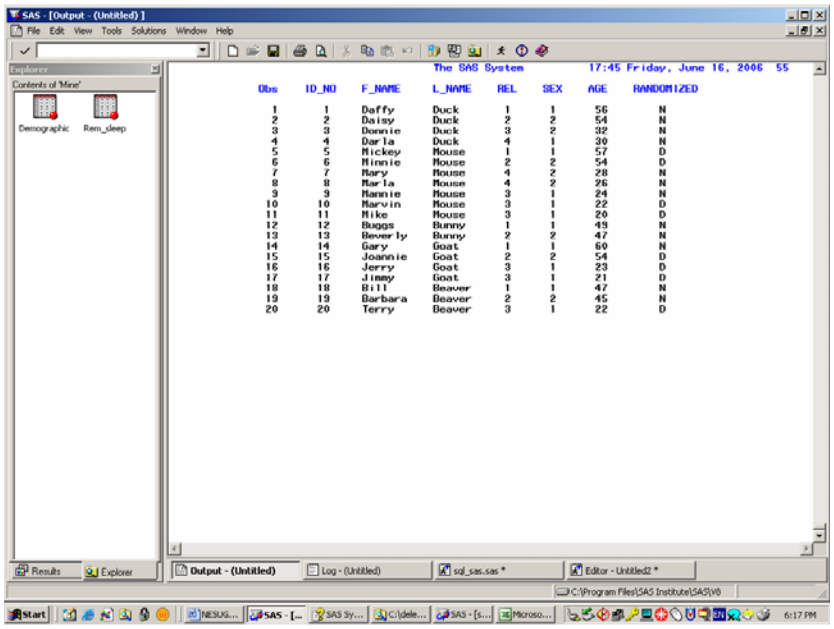
```
libname mine 'c:\deleteme\how';
data a;
set mine.demographic;
run;
proc print data=mine.demographic;
run;
```

Notice that to see the data, you must run a second procedure, a PROC PRINT.



```
libname mine 'c:\deleteme\how';

data a;
set mine.demographic;
run;

proc print data=mine.demographic;
run;
```

You should see the following output.

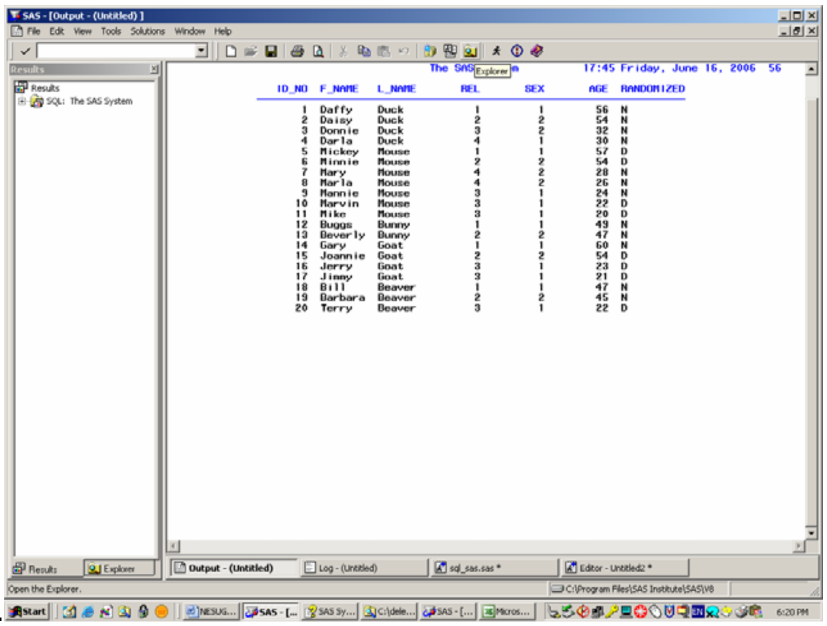


```
                              The SAS System         17:45 Friday, June 16, 2006  55

     Obs    ID_NO    F_NAME      L_NAME     REL    SEX    AGE    RANDOMIZED

       1       1     Daffy       Duck        1      1      56        N
       2       2     Daisy       Duck        2      2      54        N
       3       3     Donnie      Duck        3      2      32        N
       4       4     Darla       Duck        4      1      30        N
       5       5     Mickey      Mouse       1      1      57        D
       6       6     Minnie      Mouse       2      2      54        D
       7       7     Mary        Mouse       4      2      28        N
       8       8     Marla       Mouse       4      2      26        N
       9       9     Mannie      Mouse       3      1      24        N
      10      10     Marvin      Mouse       3      1      22        D
      11      11     Mike        Mouse       3      1      20        D
      12      12     Buggs       Bunny       1      1      49        N
      13      13     Beverly     Bunny       2      2      47        N
      14      14     Gary        Goat        1      1      60        N
      15      15     Joannie     Goat        2      2      54        D
      16      16     Jerry       Goat        3      1      23        D
      17      17     Jimmy       Goat        3      1      21        D
      18      18     Bill        Beaver      1      1      47        N
      19      19     Barbara     Beaver      2      2      45        N
      20      20     Terry       Beaver      3      1      22        D
```

To accomplish the same thing with PROC SQL, we would use the following code with an asterisk in the SELECT statement to bring in all the rows and columns. Think of the asterisk as a wild card that says "Give me everything." Go into the program editor, clear the log and output window, and type the following code, then execute the code by using the submit icon on your toolbar.

```
PROC SQL;
        Select *
        From  mine.demographic;
        run;
        quit;
```

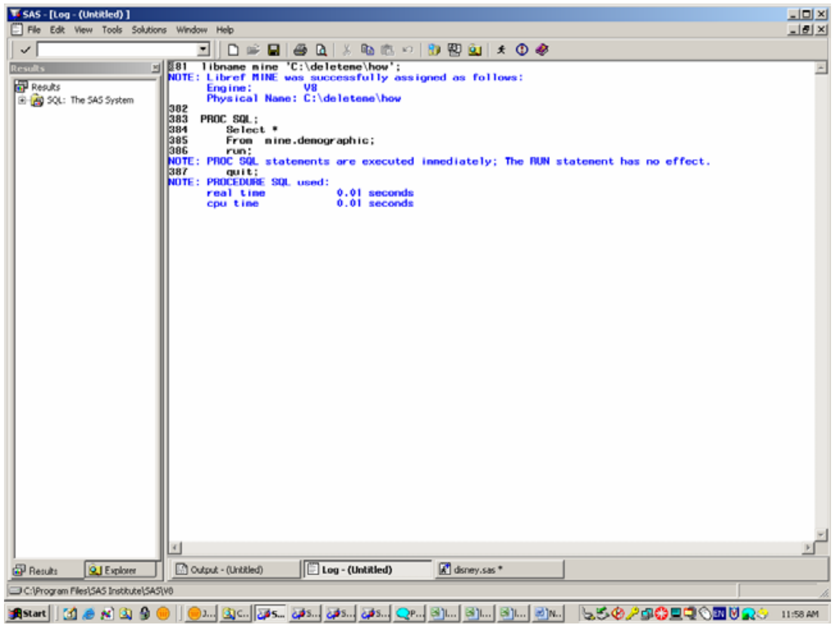
Notice that if this PROC SQL statement is correct, the output appears automatically in the output window without having to type proc print. This is one of the differences cited in the section describing differences between PROC SQL and the data step.



```
                          The SAS System          17:45 Friday, June 16, 2006  56

     ID_NO  F_NAME    L_NAME         REL        SEX        AGE  RANDOMIZED
     ---------------------------------------------------------------------
         1  Daffy     Duck             1          1         56  N
         2  Daisy     Duck             2          2         54  N
         3  Donnie    Duck             3          2         32  N
         4  Darla     Duck             4          1         30  N
         5  Mickey    Mouse            1          1         57  D
         6  Minnie    Mouse            2          2         54  D
         7  Mary      Mouse            4          2         28  N
         8  Marla     Mouse            4          2         26  N
         9  Mannie    Mouse            3          1         24  N
        10  Marvin    Mouse            3          1         22  D
        11  Mike      Mouse            3          1         20  D
        12  Buggs     Bunny            1          1         49  N
        13  Beverly   Bunny            2          2         47  N
        14  Gary      Goat             1          1         60  N
        15  Joannie   Goat             2          2         54  D
        16  Jerry     Goat             3          1         23  D
        17  Jimmy     Goat             3          1         21  D
        18  Bill      Beaver           1          1         47  N
        19  Barbara   Beaver           2          2         45  N
        20  Terry     Beaver           3          1         22  D
```

.

Always check the log. Notice that the run statement produced a warning that it had no effect. You do not need a run statement when using PROC SQL. You do however need a *quit;* statement. This is another example of two of the differences cited in the section on differences between PROC SQL and the data step.



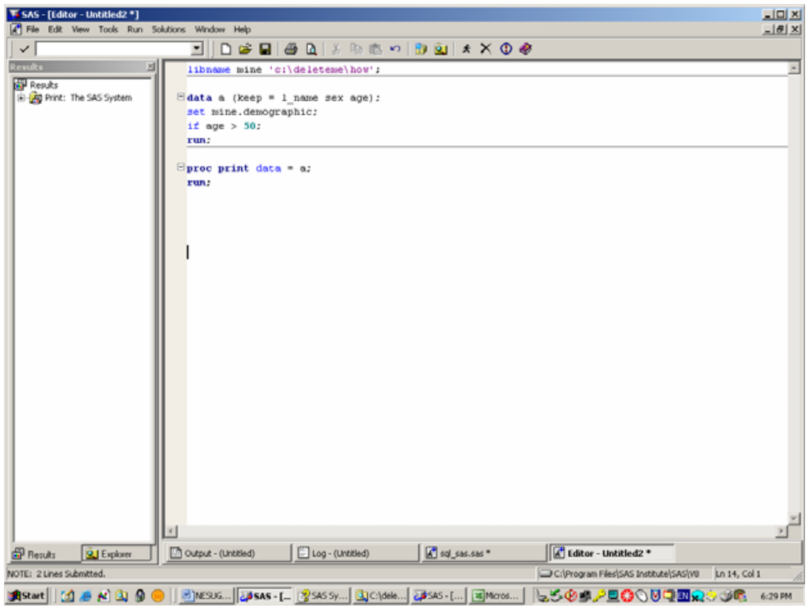
```
381   libname mine 'C:\deleteme\how';
NOTE: Libref MINE was successfully assigned as follows:
      Engine:        V8
      Physical Name: C:\deleteme\how
382
383   PROC SQL;
384       Select *
385       From  mine.demographic;
386       run;
NOTE: PROC SQL statements are executed immediately; The RUN statement has no effect.
387       quit;
NOTE: PROCEDURE SQL used:
      real time           0.01 seconds
      cpu time            0.01 seconds
```

SUBSETTING THE DATA

Now let's subset the data by defining just those columns that we want using a subsetting WHERE clause. In this instance, let's extract the sleep participants "WHERE age > 50".

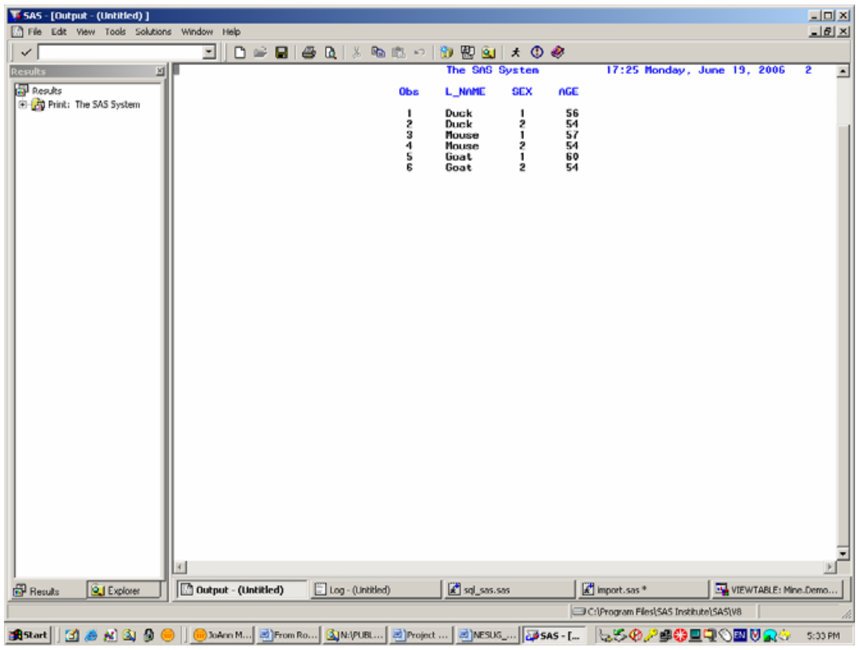
Using a data step, we would set the table and then do a proc print naming just those variables that we wanted to see printed. Go into the program editor, clear the log and output window, and type the following code, then execute the code by using the submit icon on your toolbar.

```
Data a  (keep = l_name sex age) ;
Set mine.demographic;
If age > 50;
Run;

Proc print data=a;
Run;
```



```
libname mine 'c:\deleteme\how';

data a (keep = l_name sex age);
set mine.demographic;
if age > 50;
run;

proc print data = a;
run;
```

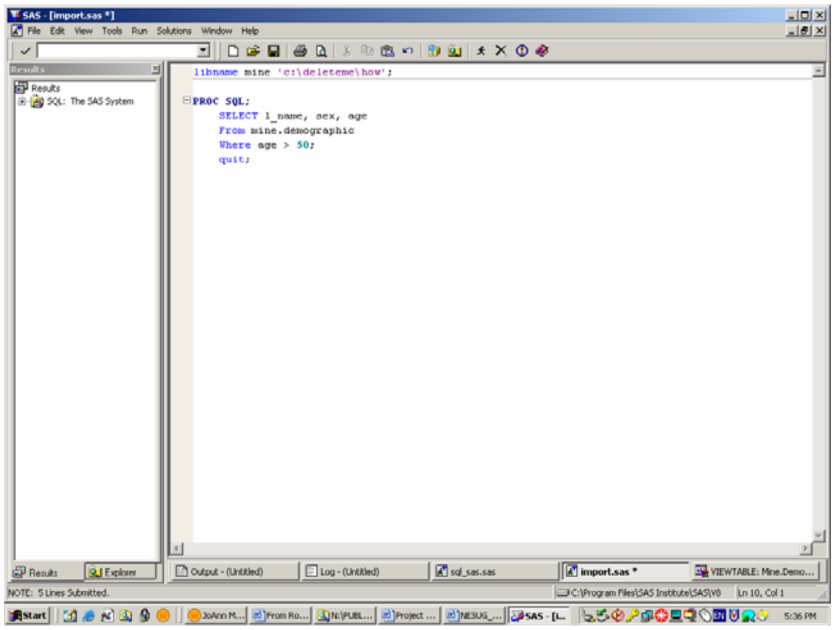
You should see the following output.



```
          The SAS System          17:25 Monday, June 19, 2006   2

   Obs    L_NAME    SEX    AGE

    1     Duck       1      56
    2     Duck       2      54
    3     Mouse      1      57
    4     Mouse      2      54
    5     Goat       1      60
    6     Goat       2      54
```

Now let's subset this data using PROC SQL. Go to the Program Editor window, clear the log and type the following code, then execute the code by using the submit icon on your toolbar.

```
PROC SQL;
      SELECT l_name, sex, age
      From mine.demographic
      Where age > 50;
      quit;
```

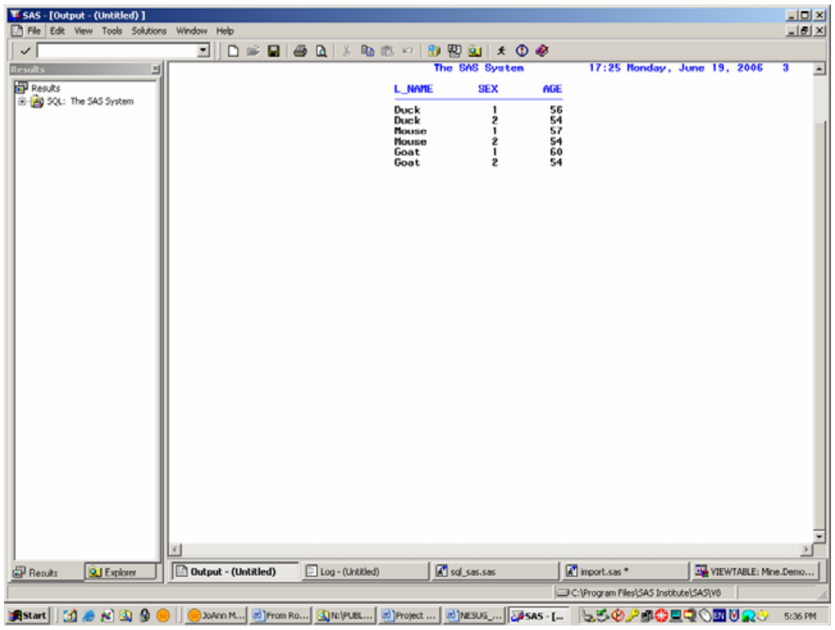


```
libname mine 'c:\deleteme\how';

PROC SQL;
     SELECT l_name, sex, age
     From mine.demographic
     Where age > 50;
     quit;
```

NOTE: 5 Lines Submitted.

Notice that variables in the SELECT statement in PROC SQL require a comma between them but not at the end of the select statement. This is another difference between PROC SQL and the data step. Again, unlike the data step, the PROC SQL statement does not require a run statement. It will produce a message into the log that says the run statement has no effect if you do include it.

You should see the following output.



```
              The SAS System          17:25 Monday, June 19, 2006   3

     L_NAME          SEX        AGE
     ---------------------------------
     Duck              1         56
     Duck              2         54
     Mouse             1         57
     Mouse             2         54
     Goat              1         60
     Goat              2         54
```

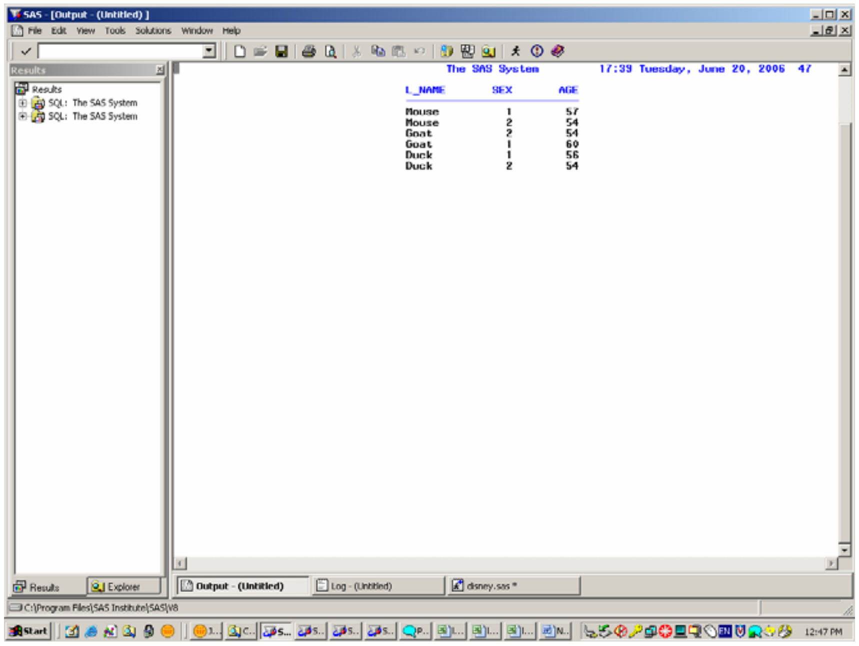
So far, you will notice that the PROC SQL code does use a few less keystrokes, but the output is similar to the data step.

ORDER BY AND GROUP BY

Let's take the WHERE clause and build upon it. Using the same logic as above, let's put some order to our data. ORDER BY returns rows in ascending order unless you specify DESC, for descending. Let's order persons over 50 in descending order. Notice that instead of using the actual variable name, we can use an alias. The number 1 says give me the variable that is in position 1 on the SELECT statement. This is an example of a relative column number that is used in substitute for a variable name. A relative column number can be used in the ORDER BY and GROUP BY statements. Go to the program editor, clear the log and output window, and type the following code, then execute the code by using the submit icon on your toolbar.

```
PROC SQL;
      SELECT l_name, sex, age
      From mine.demographic
      Where age > 50
      Order by 1 DESC;
      quit;
```

Your output should look like this.



```
             The SAS System         17:39 Tuesday, June 20, 2006  47

     L_NAME         SEX        AGE
     ----------------------------
     Mouse            1         57
     Mouse            2         54
     Goat             2         54
     Goat             1         60
     Duck             1         56
     Duck             2         54
```

You can group the data by the same variable with the GROUP BY statement. The syntax for this would look thus:

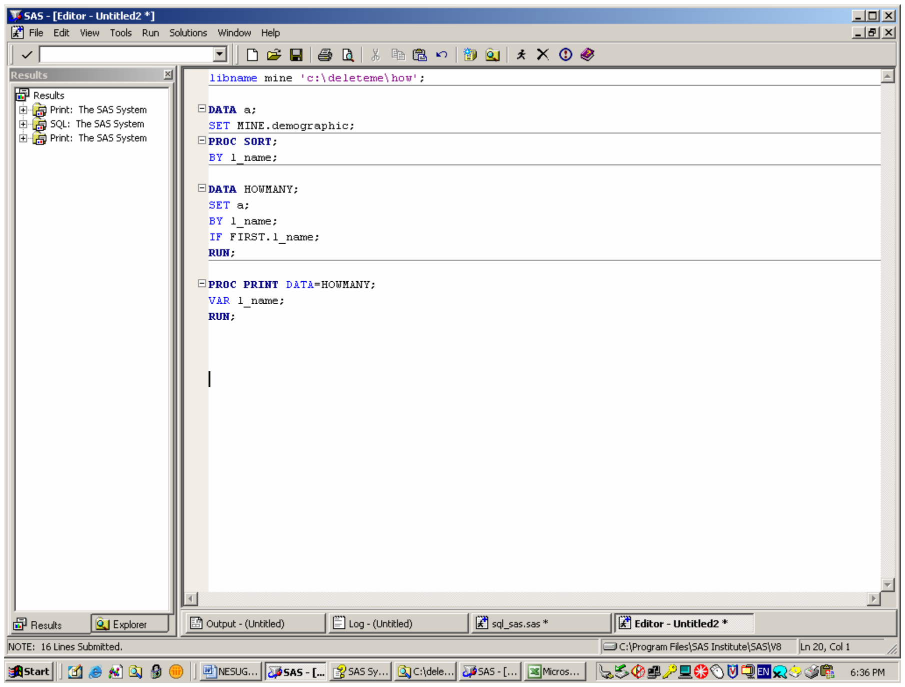
```
PROC SQL;
      SELECT l_name, sex, age
      From mine.demographic
      Where age > 50
      Order by 1;
      Group by 1;
      quit;
```

One word of caution: Do not try to GROUP BY using summarized values. An error message pertaining to non-aggregate values on the GROUP BY statement can be confusing but says that you have tried to do a group by with summarized values. This is a common error when first using the GROUP BY.

DISTINCT FUNCTION

The DISTINCT function in the SELECT statement is very powerful. If you have more than one row for a value and you only want one value returned from your query, use DISTINCT. Let's bring in all the variables in our table to see how many distinct last names there are in this dataset. In a traditional data step we would have to set the data, sort the data, then subset the data by the sort field, using an **IF FIRST**. statement, and finally print the data. Go into the program editor, clear the log and output window, and type the following code, then execute the code by using the submit icon on your toolbar.

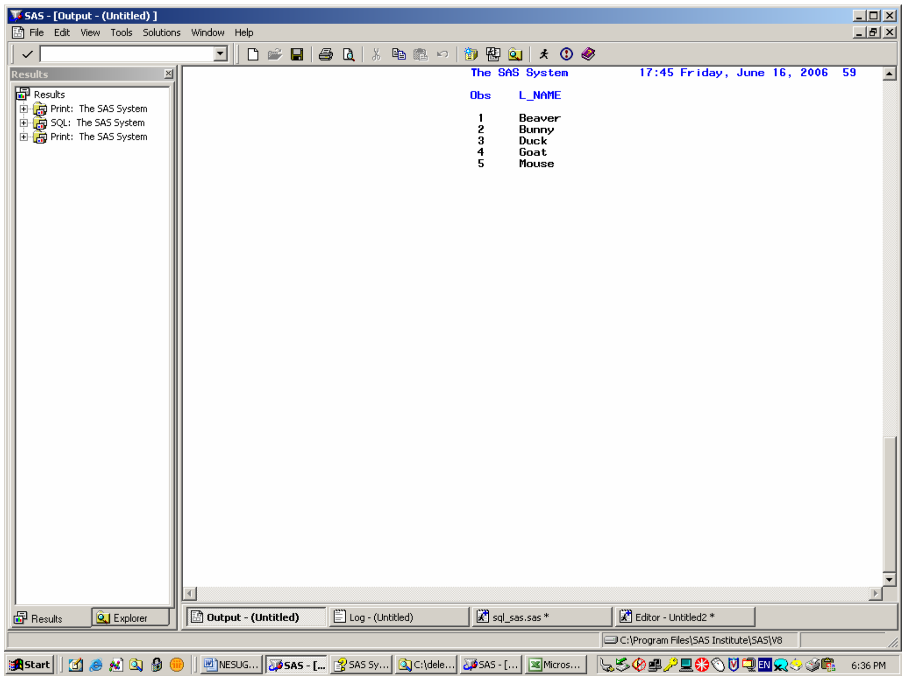
```
DATA a;
SET MINE.demographic;

PROC SORT;
BY l_name;

DATA HOWMANY;
SET a;
BY l_name;
IF FIRST.l_name;
RUN;

PROC PRINT DATA=HOWMANY;
VAR l_name;
RUN;
```



```
libname mine 'c:\deleteme\how';

DATA a;
SET MINE.demographic;
PROC SORT;
BY l_name;

DATA HOWMANY;
SET a;
BY l_name;
IF FIRST.l_name;
RUN;

PROC PRINT DATA=HOWMANY;
VAR l_name;
RUN;
```

NOTE: 16 Lines Submitted. C:\Program Files\SAS Institute\SAS\V8 Ln 20, Col 1

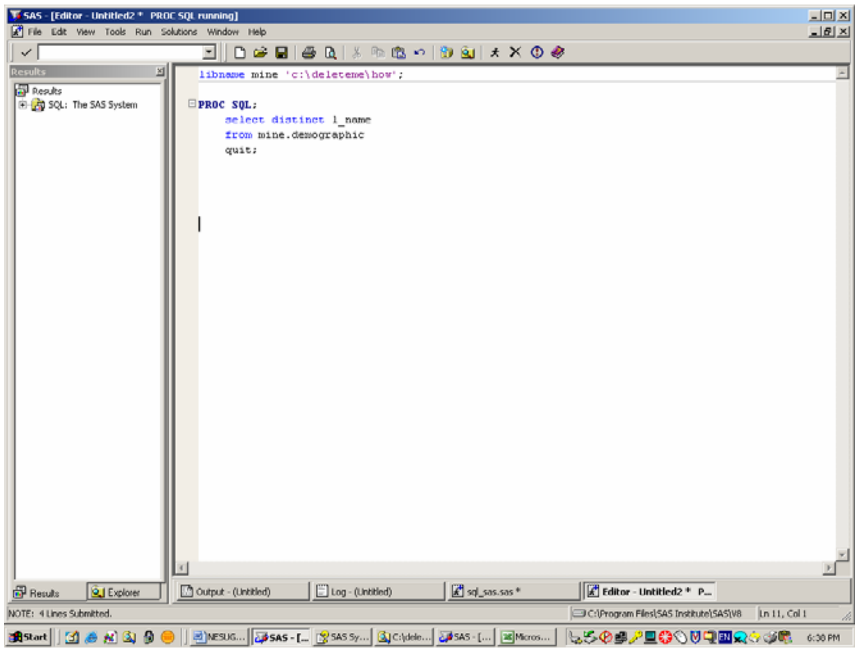
Your output should look like this:



```
                The SAS System          17:45 Friday, June 16, 2006  59

                Obs    L_NAME

                  1    Beaver
                  2    Bunny
                  3    Duck
                  4    Goat
                  5    Mouse
```

With PROC SQL the distinct operator passes the statement directly to the DBMS, not through SAS, so that the operator checks for duplicate rows. Distinct is a very powerful operator in the PROC SQL statement. Go into the program editor, clear the log and output window, and type the following code, then execute the code by using the submit icon on your toolbar.

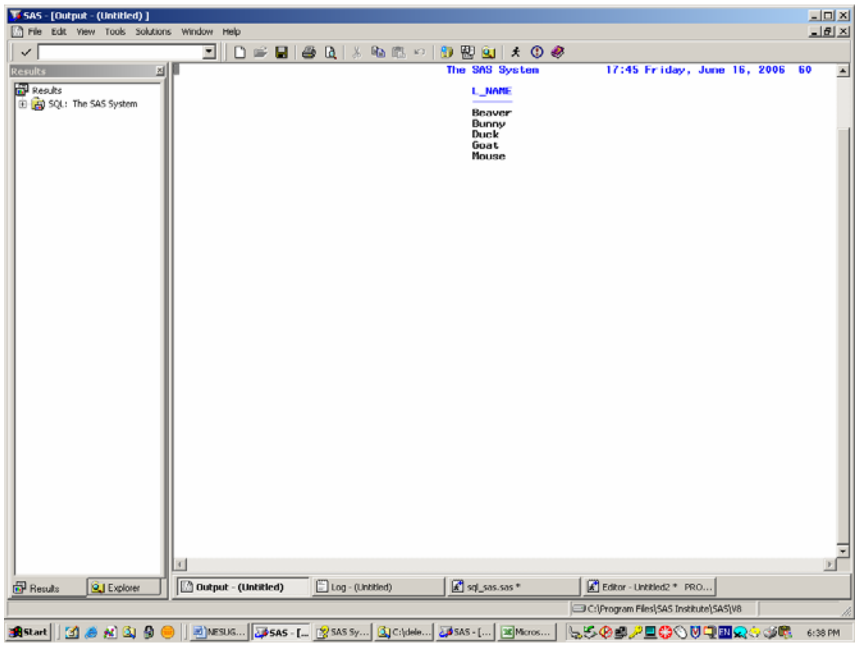
```
PROC SQL;
       select distinct l_name
       from mine.demographic
       quit;
```



```
libname mine 'c:\deleteme\how';

PROC SQL;
    select distinct l_name
    from mine.demographic
    quit;
```

```
NOTE: 4 Lines Submitted.
```

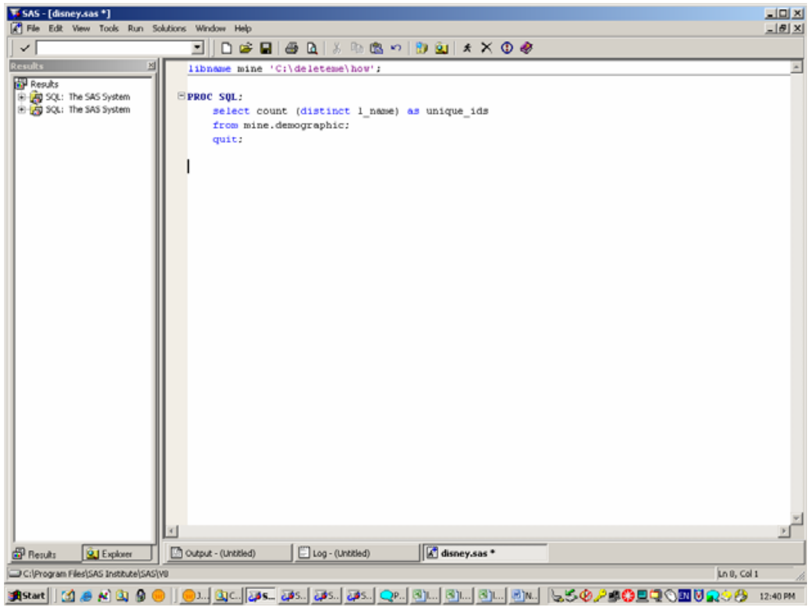
Notice that the output appears immediately. Notice also how much less code this query required as compared to a traditional data step. The beauty of SQL is impressive when the table contains millions of observations, and lots of distinct values.



```
The SAS System          17:45 Friday, June 16, 2006  60

   L_NAME
   ______
   Beaver
   Bunny
   Duck
   Goat
   Mouse
```

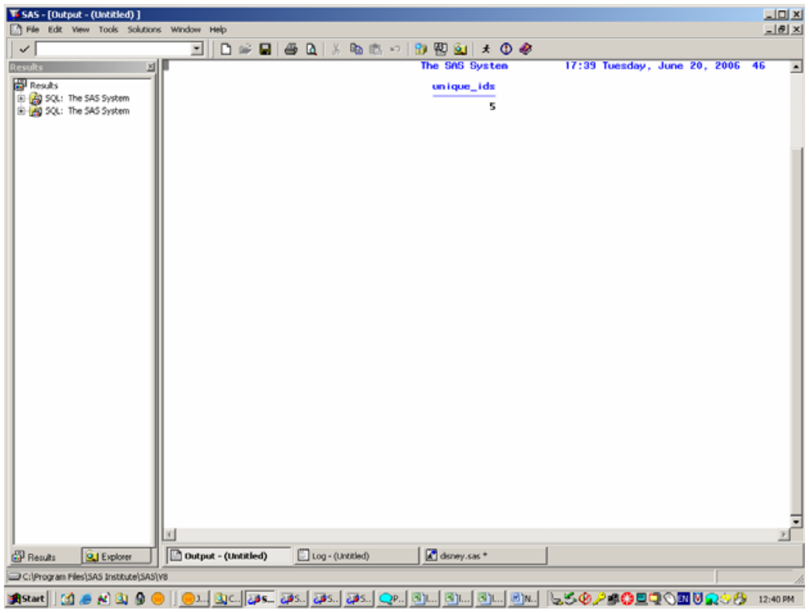
PROQ SQL using the SELECT DISTINCT can be used for one or more variables at the same time, and the table does not have to be presorted. This truly is a powerful difference between traditional data steps and PROC SQL. In addition, you can use DISTINCT with COUNT to nest the query and count the number of DISTINCT values. Go into the program editor, clear the log and output window, and type the following code, then execute the code by using the submit icon on your toolbar.

```
PROC SQL;
      select count (distinct l_name)
      from mine.demographic
      quit;
```



```
libname mine 'C:\deleteme\how';

PROC SQL;
    select count (distinct l_name) as unique_ids
    from mine.demographic;
    quit;
```

The count function counted the number of distinct last names in the data. Your output should look like this:

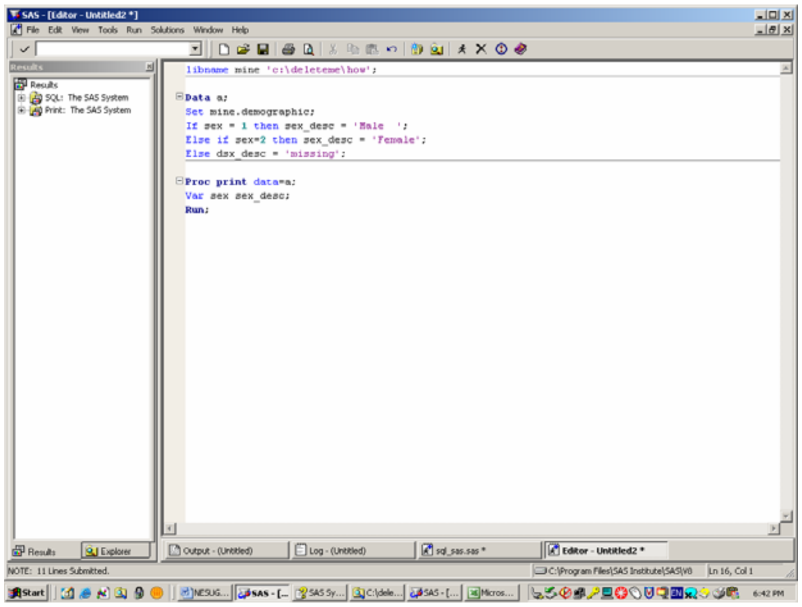


```
The SAS System          17:39 Tuesday, June 20, 2006  46

  unique_ids
  ----------
           5
```

CASE STATEMENT

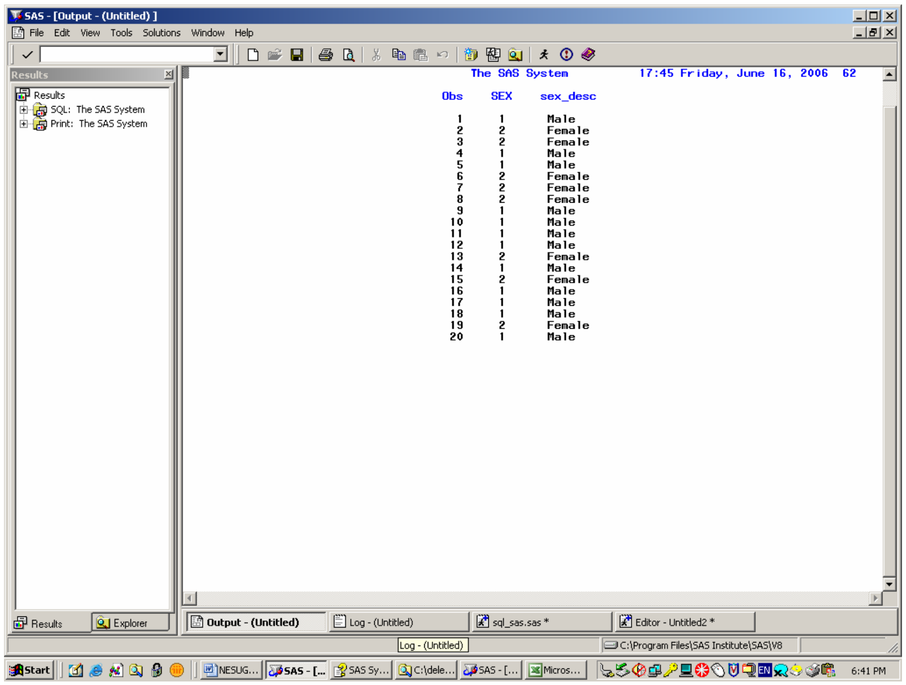
The CASE statement allows you to give meaningful names to variables or created fields in your program. This would compare to a data step using **if then else** logic. Note that the value 'missing' will be truncated (cut off in this example) unless you leave spaces for six characters. Let's create meaningful values for the sex variable using traditional data steps. Go into the program editor, clear the log and output window, and type the following code, then execute the code by using the submit icon on your toolbar.

```
Data a;
Set mine.demographic;

If sex = 1 then sex_desc = 'Male  ';  /* leave two spaces*/
Else if sex=2 then sex_desc = 'Female';
Else sex_desc = 'missing';

Proc print data = a;
Var sex sex_desc;

Run;
```



```
libname mine 'c:\deleteme\how';

Data a;
Set mine.demographic;
If sex = 1 then sex_desc = 'Male  ';
Else if sex=2 then sex_desc = 'Female';
Else dsx_desc = 'missing';

Proc print data=a;
Var sex sex_desc;
Run;
```

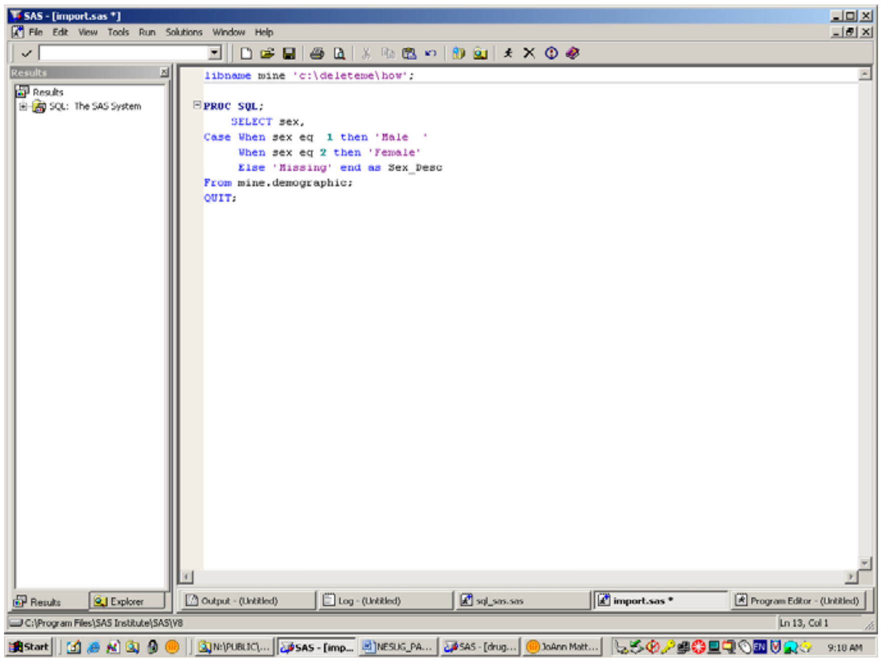
Your output should look like this:



```
          The SAS System        17:45 Friday, June 16, 2006  62

    Obs    SEX    sex_desc

      1     1      Male
      2     2      Female
      3     2      Female
      4     1      Male
      5     1      Male
      6     2      Female
      7     2      Female
      8     2      Female
      9     1      Male
     10     1      Male
     11     1      Male
     12     1      Male
     13     2      Female
     14     1      Male
     15     2      Female
     16     1      Male
     17     1      Male
     18     1      Male
     19     2      Female
     20     1      Male
```

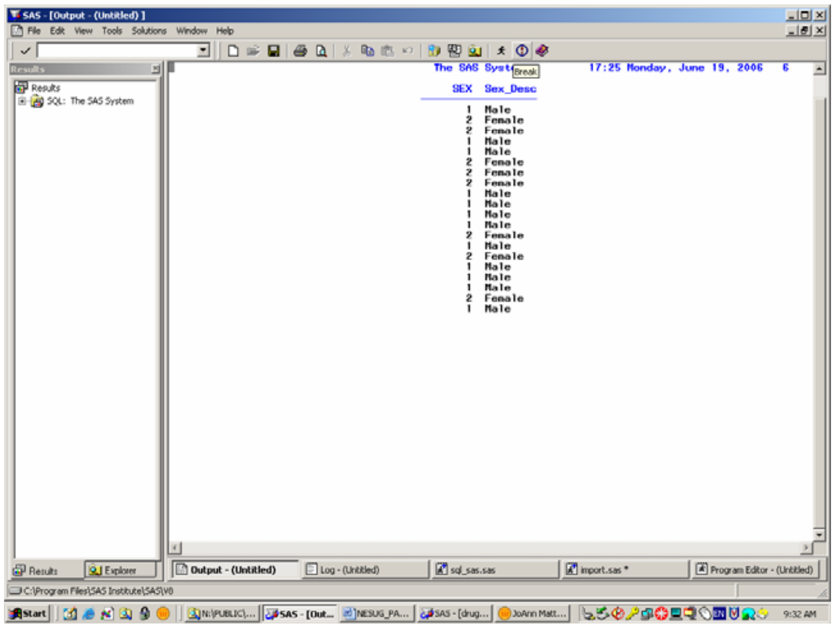
Now let's compare the CASE statement using PROC SQL. The following is a simple case statement for our Disney sleep research data. Notice that the CASE statement requires an END statement. Go into the program editor, clear the log and output window, and type the following code, then execute the code by using the submit icon on your toolbar.

```
PROC SQL;
      SELECT sex,
Case When sex eq  1 then 'Male   '
     When sex eq 2 then 'Female'
     Else 'Missing' end as Sex_Desc
From mine.demographic;
QUIT;
```



```
libname mine 'c:\deleteme\how';

PROC SQL;
    SELECT sex,
Case When sex eq  1 then 'Male  '
     When sex eq 2 then 'Female'
     Else 'Missing' end as Sex_Desc
From mine.demographic;
QUIT;
```

You should see the following output:



```
              The SAS System        17:25 Monday, June 19, 2006   6

                 SEX  Sex_Desc
                 ---------------
                   1  Male
                   2  Female
                   2  Female
                   1  Male
                   1  Male
                   2  Female
                   2  Female
                   2  Female
                   1  Male
                   1  Male
                   1  Male
                   1  Male
                   2  Female
                   1  Male
                   2  Female
                   1  Male
                   1  Male
                   1  Male
                   2  Female
                   1  Male
```

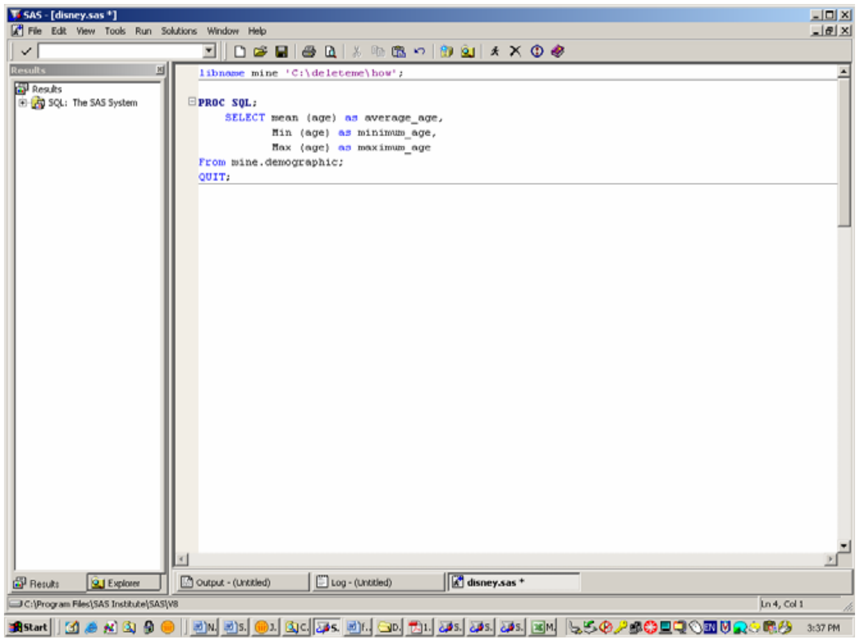
You can create many new variables by combining case statements, and the syntax would look thus:

```
PROC SQL;
      SELECT sex, rel, employ,
Case When sex eq 1 then 'Male   '
     When sex eq 2 then 'Female'
     Else 'Missing' end as Sex_Desc,
Case When rel eq 1 then 'Father   '
     When rel eq 2 then 'Mother   '
     When rel eq 3 then 'Daughter'
     When rel eq 4 then 'Son'
     Else 'Missing' end as Rel_Desc,
Case When employ eq 1 then 'Employee  '
     When employ eq 2 then 'Dependent'
     Else 'Missing' end as Emp_Desc,
from mine.demographic;
QUIT;
```

SUMMARY FUNCTIONS

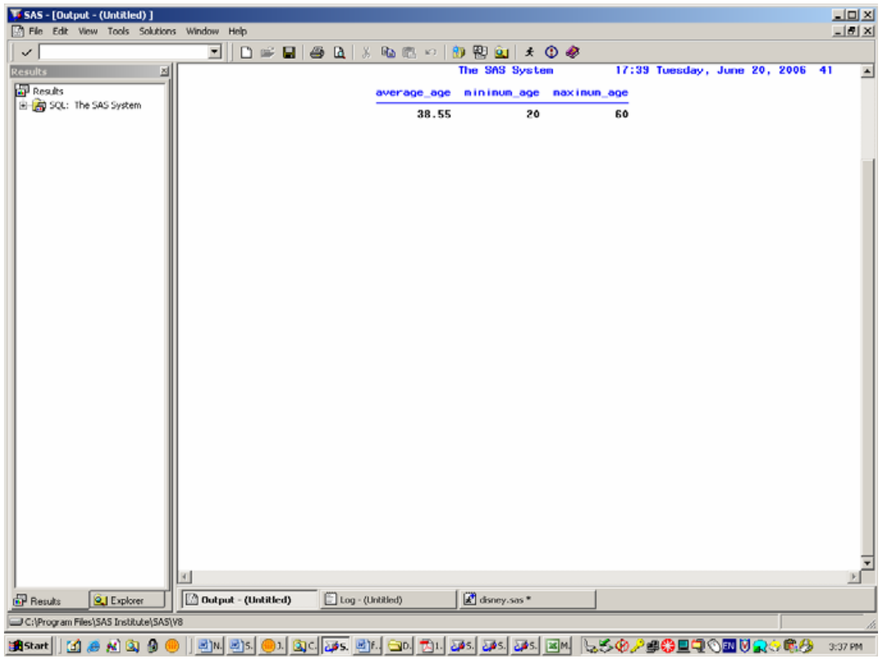
Let's take that code we just created and remove the CASE statement and modify it to calculate the mean age with a summary function. Notice we are giving the summary statistic an alias so that it has a meaningful label. We have generated the mean, minimum and maximum but there are lots of other summary functions, including AVG, MEAN, COUNT, FREQ, MIN, MAX, NMISS, to name a few. Revise the code from our last PROC SQL, then execute the code by using the submit icon on your toolbar.

```
PROC SQL;
      SELECT mean (age) as average_age,
             Min (age) as minimum_age,
             Max (age) as maximum_age
From mine.demographic;
QUIT;
```



```
libname mine 'C:\deleteme\how';

PROC SQL;
    SELECT mean (age) as average_age,
           Min (age) as minimum_age,
           Max (age) as maximum_age
From mine.demographic;
QUIT;
```

Your output should look like this:


SAS - [Output - (Untitled)]
File Edit View Tools Solutions Window Help
Results
Results
SQL: The SAS System
Results
Explorer
Output - (Untitled)
Log - (Untitled)
disney.sas *
C:\Program Files\SAS Institute\SAS\V8
Start
3:37 PM

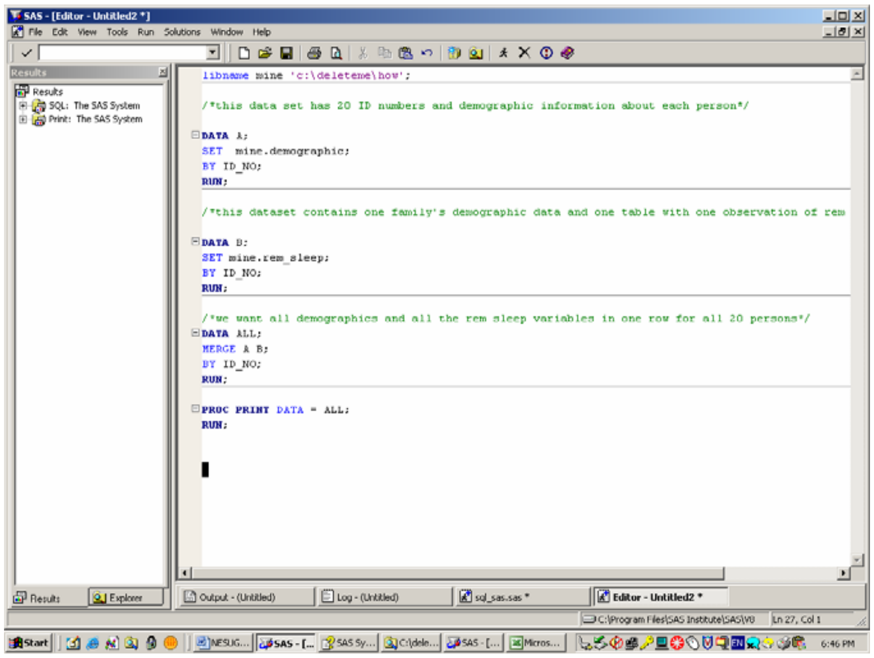
```
                          The SAS System          17:39 Tuesday, June 20, 2006  41

                average_age  minimum_age  maximum_age
                ---------------------------------------
                      38.55           20           60
```

MERGING VERSUS JOINS – The true power of PROC SQL
The true power of the PROC SQL procedure will become apparent when you are merging very large datasets or combining two datasets with some common variable. Because PROC SQL does not require presorting of the tables you are joining, computer resources are saved when using joins. There are ways to think about which is more advantageous, to merge using a data step or join tables using PROC SQL.

ONE TO ONE MERGES
For one to one matching, both the data step and PROC SQL are acceptable and use about the same resources. Begin by considering a simple SAS program that takes data from two tables and merges both into one final table: The demographics data set has 20 ID numbers and demographic information about each person. The rem_sleep dataset also has 20 rem sleep values for the same id's. We want all demographic data and all the rem sleep data to be combined into one table with 20 observations. Go into the program editor, clear the log and output window, and type the following code, then execute the code by using the submit icon on your toolbar.

```
DATA A;
SET  mine.demographic;
Proc sort;
BY ID_NO;
RUN;

/*this dataset contains one family's demographic data and one table with one
observation of rem sleep data per person.  */

DATA B;
SET mine.rem_sleep;
Proc sort;
BY ID_NO;
RUN;

/*we want all demographics and all the rem sleep variables in one row for all 20
persons*/

DATA ALL;
MERGE A B;
```

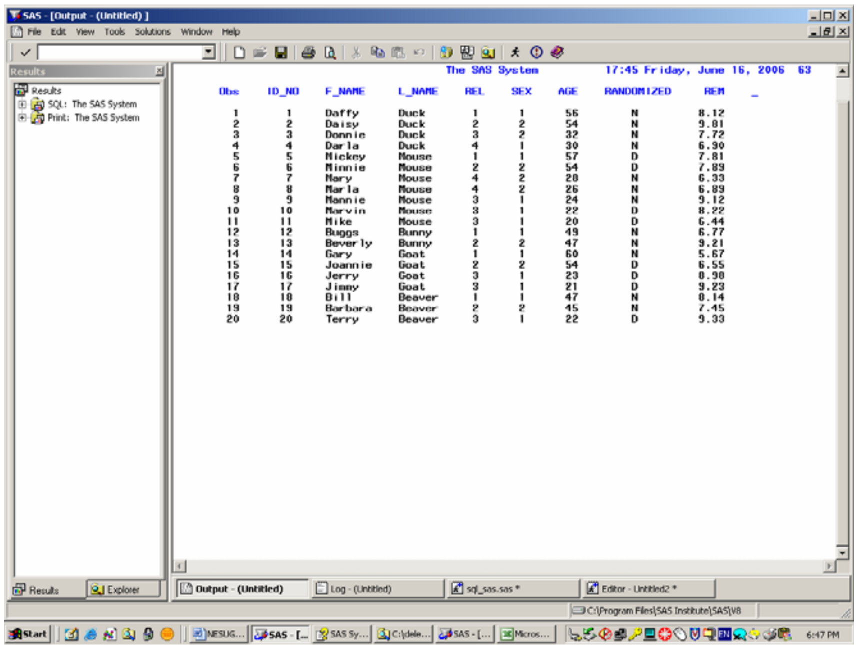
```
BY ID_NO;
RUN;
PROC PRINT DATA = ALL;
RUN;
```



```
libname mine 'c:\deleteme\how';

/*this data set has 20 ID numbers and demographic information about each person*/

DATA A;
SET  mine.demographic;
BY ID_NO;
RUN;

/*this dataset contains one family's demographic data and one table with one observation of rem

DATA B;
SET mine.rem_sleep;
BY ID_NO;
RUN;

/*we want all demographics and all the rem sleep variables in one row for all 20 persons*/
DATA ALL;
MERGE A B;
BY ID_NO;
RUN;

PROC PRINT DATA = ALL;
RUN;
```

In this example we are doing a one to one merge. That is to say we have one match from each table. We can merge the two datasets and the resulting final dataset will contain one observation with id_no and the variables from each dataset.

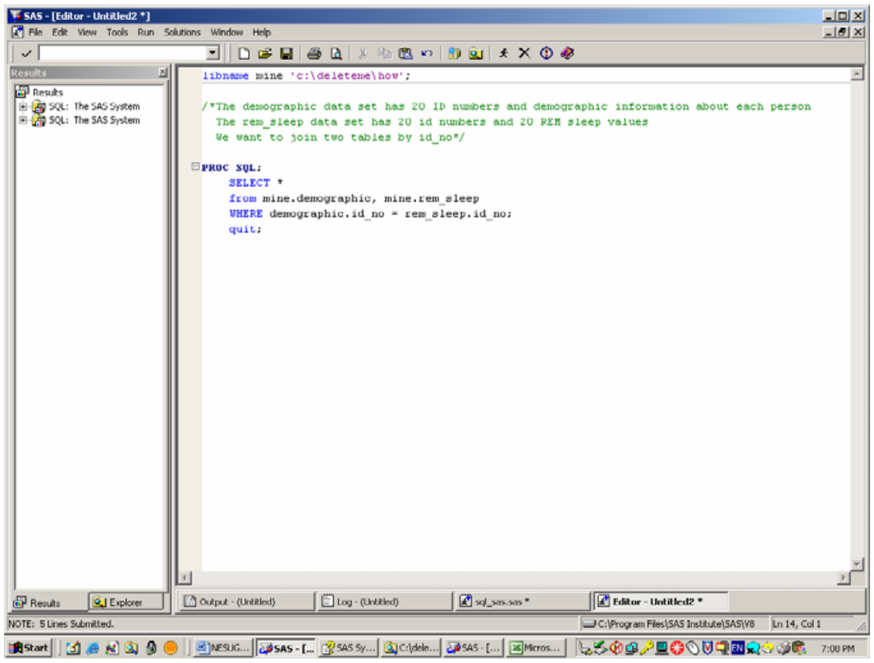
Submit the code and out output should look like this.



```
                                   The SAS System         17:45 Friday, June 16, 2006  63

  Obs    ID_NO   F_NAME    L_NAME    REL    SEX    AGE   RANDOMIZED    REM   _

    1      1     Daffy     Duck       1      1     56        N        8.12
    2      2     Daisy     Duck       2      2     54        N        9.81
    3      3     Donnie    Duck       3      2     32        N        7.72
    4      4     Darla     Duck       4      1     30        N        6.90
    5      5     Mickey    Mouse      1      1     57        D        7.81
    6      6     Minnie    Mouse      2      2     54        D        7.89
    7      7     Mary      Mouse      4      2     20        N        6.33
    8      8     Marla     Mouse      4      2     26        N        6.89
    9      9     Mannie    Mouse      3      1     24        N        9.12
   10     10     Marvin    Mouse      3      1     22        D        8.22
   11     11     Mike      Mouse      3      1     20        D        6.44
   12     12     Buggs     Bunny      1      1     49        N        6.77
   13     13     Beverly   Bunny      2      2     47        N        9.21
   14     14     Gary      Goat       1      1     60        N        5.67
   15     15     Joannie   Goat       2      2     54        D        6.55
   16     16     Jerry     Goat       3      1     23        D        8.98
   17     17     Jimmy     Goat       3      1     21        D        9.23
   18     18     Bill      Beaver     1      1     47        N        8.14
   19     19     Barbara   Beaver     2      2     45        N        7.45
   20     20     Terry     Beaver     3      1     22        D        9.33
```

We can accomplish the same merging of these two datasets with a simple PROC SQL. In this instance we want to join both tables on a common variable, id_no. An inner join will discard any rows that do not have a corresponding row in both source tables. In this sample, however there is an equal match for each row, i.e. a ONE to ONE merge. Go into the program editor, clear the log and output window, and type the following code, then execute the code by using the submit icon on your toolbar.

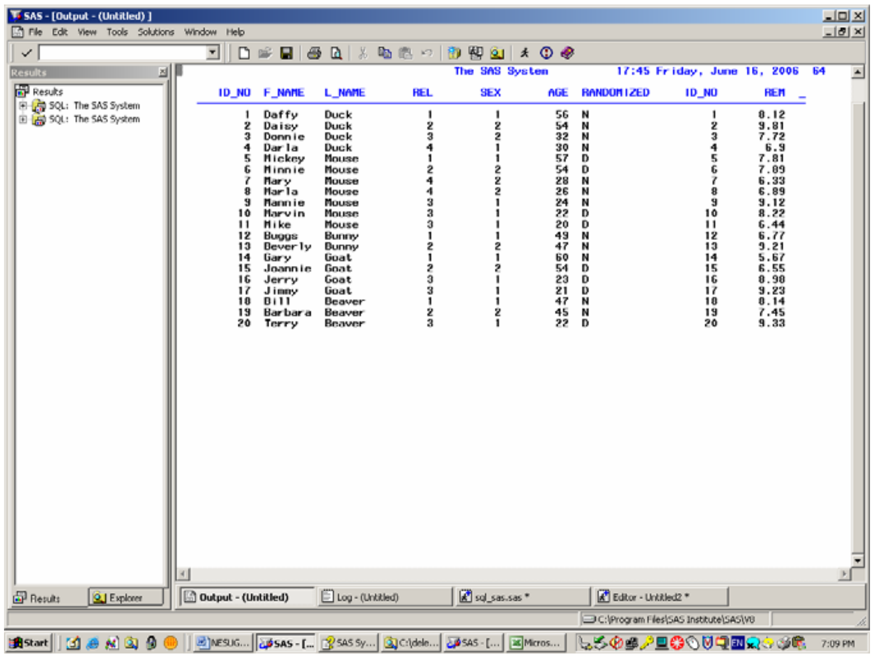
```
PROC SQL;
SELECT *
from mine.demographic, mine.rem_sleep
WHERE demographic.id_no = rem_sleep.id_no;
      quit;
```



```
libname mine 'c:\deleteme\how';

/*The demographic data set has 20 ID numbers and demographic information about each person
  The rem_sleep data set has 20 id numbers and 20 REM sleep values
  We want to join two tables by id_no*/

PROC SQL;
    SELECT *
    from mine.demographic, mine.rem_sleep
    WHERE demographic.id_no = rem_sleep.id_no;
    quit;
```

NOTE: 5 Lines Submitted.

You should see the following in your output window:

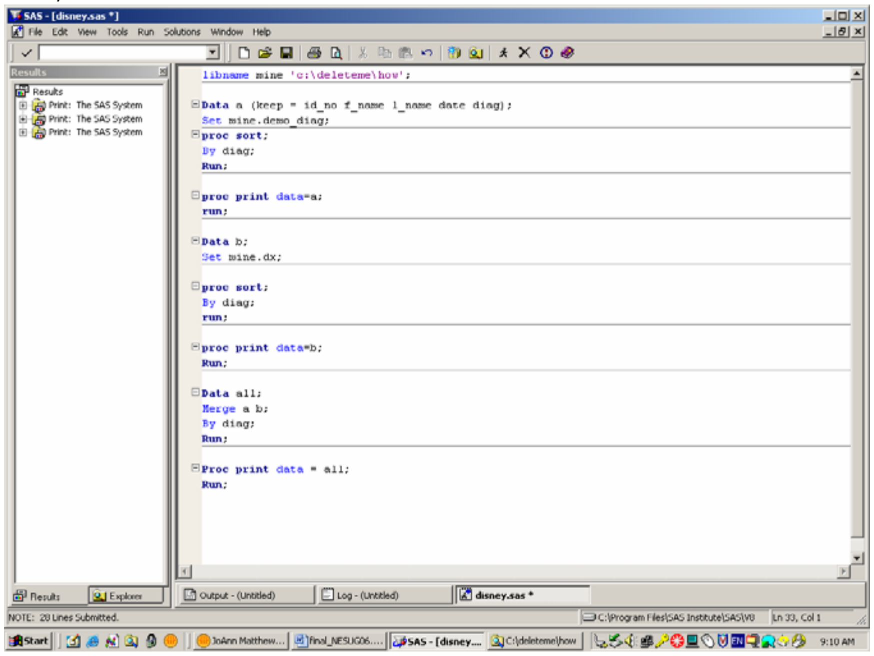


```
                                        The SAS System          17:45 Friday, June 16, 2006   64

       ID_NO  F_NAME    L_NAME        REL        SEX         AGE  RANDOMIZED     ID_NO       REM
       ------------------------------------------------------------------------------------------
           1  Daffy     Duck            1          1          56  N                  1      8.12
           2  Daisy     Duck            2          2          54  N                  2      9.81
           3  Donnie    Duck            3          2          32  N                  3      7.72
           4  Darla     Duck            4          1          30  N                  4       6.9
           5  Mickey    Mouse           1          1          57  D                  5      7.81
           6  Minnie    Mouse           2          2          54  D                  6      7.89
           7  Mary      Mouse           4          2          28  N                  7      6.33
           8  Marla     Mouse           4          2          26  N                  8      6.89
           9  Mannie    Mouse           3          1          24  N                  9      9.12
          10  Marvin    Mouse           3          1          22  D                 10      8.22
          11  Mike      Mouse           3          1          20  D                 11      6.44
          12  Buggs     Bunny           1          1          49  N                 12      6.77
          13  Beverly   Bunny           2          2          47  N                 13      9.21
          14  Gary      Goat            1          1          60  N                 14      5.67
          15  Joannie   Goat            2          2          54  D                 15      6.55
          16  Jerry     Goat            3          1          23  D                 16      8.98
          17  Jimmy     Goat            3          1          21  D                 17      9.23
          18  Bill      Beaver          1          1          47  N                 18      8.14
          19  Barbara   Beaver          2          2          45  N                 19      7.45
          20  Terry     Beaver          3          1          22  D                 20      9.33
```

ONE TO MANY MERGES

Like one to one merges, one to many merges can be accomplished either with data steps or an PROC SQL procedure. So far we have worked with very small datasets. Let's look at a larger dataset that contains multiple evaluation times for Disney characters who have participated in a sleep study for depression. The dataset is called mine.demo_diag. It contains all the Disney characters evaluation times and diagnoses. The mine.dx file contains one value for each of 33 psychiatric diagnoses. What we want to see are only those diagnoses for the Disney dataset (n=32). Go into the program editor, clear the log and output window, and type the following code, then execute the code by using the submit icon on your toolbar.

```
libname mine 'c:\deleteme\how';
Data a (keep = id_no f_name l_name date diag);
Set mine.demo_diag;

proc sort data=a;
By diag;
Run;

proc print data=a;
run;

Data b;
Set mine.dx;

proc sort data=b;
By diag;
run;

proc print data=b;
Run;

Data all;
Merge a b;
By diag;
Run;

Proc print data = all;
Run;
```

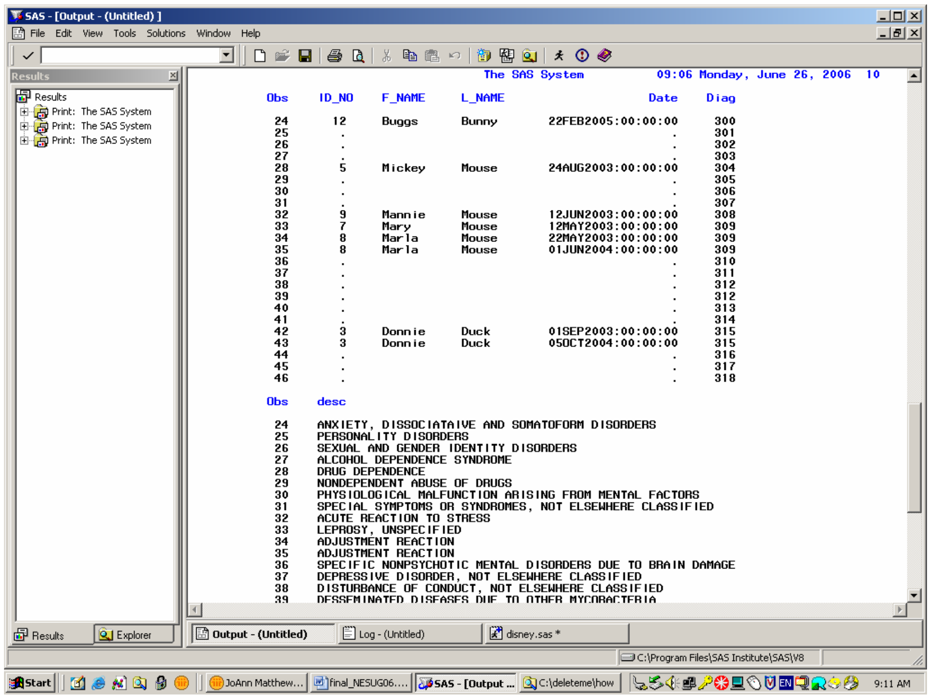


```
libname mine 'c:\deleteme\how';

Data a (keep = id_no f_name l_name date diag);
Set mine.demo_diag;
proc sort;
By diag;
Run;

proc print data=a;
run;

Data b;
Set mine.dx;

proc sort;
By diag;
run;

proc print data=b;
Run;

Data all;
Merge a b;
By diag;
Run;

Proc print data = all;
Run;
```

```
NOTE: 20 Lines Submitted.          C:\Program Files\SAS Institute\SAS\V8     Ln 33, Col 1
```

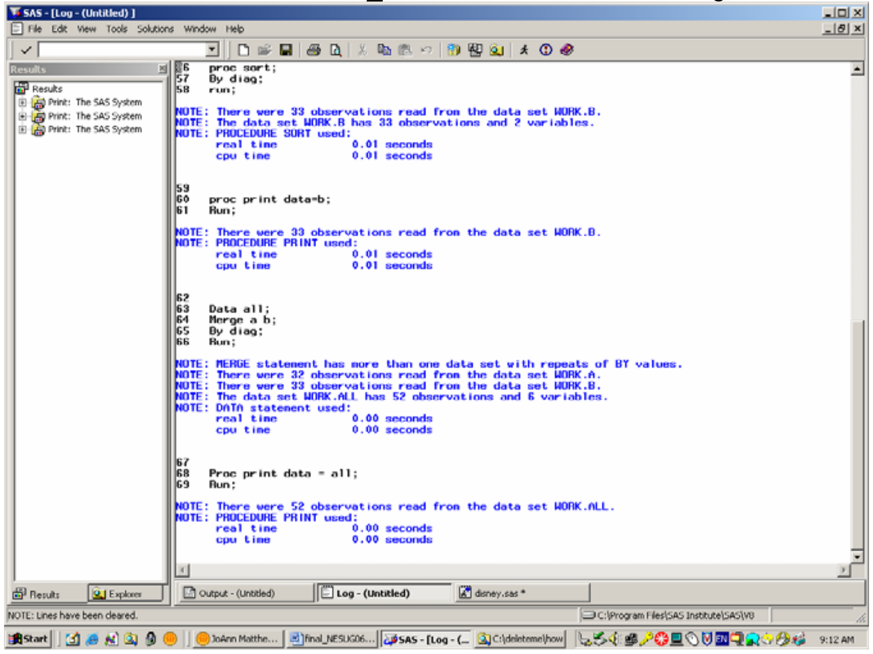
In this example we are doing a one to many merge, with many id_no's in the first dataset but only one value for each of the diagnosis numbers on the second dataset. We want to merge the datasets and have one diagnosis description for each row of the demo_diag table or n = 32.

Look at the output. Displayed is the second page of output, and it is messy and incorrect.



```
                                    The SAS System        09:06 Monday, June 26, 2006  10

         Obs    ID_NO    F_NAME    L_NAME                   Date    Diag

          24      12     Buggs     Bunny      22FEB2005:00:00:00     300
          25       .                                           .     301
          26       .                                           .     302
          27       .                                           .     303
          28       5     Mickey    Mouse      24AUG2003:00:00:00     304
          29       .                                           .     305
          30       .                                           .     306
          31       .                                           .     307
          32       9     Mannie    Mouse      12JUN2003:00:00:00     308
          33       7     Mary      Mouse      12MAY2003:00:00:00     309
          34       8     Marla     Mouse      22MAY2003:00:00:00     309
          35       8     Marla     Mouse      01JUN2004:00:00:00     309
          36       .                                           .     310
          37       .                                           .     311
          38       .                                           .     312
          39       .                                           .     312
          40       .                                           .     313
          41       .                                           .     314
          42       3     Donnie    Duck       01SEP2003:00:00:00     315
          43       3     Donnie    Duck       05OCT2004:00:00:00     315
          44       .                                           .     316
          45       .                                           .     317
          46       .                                           .     318

         Obs    desc

          24    ANXIETY, DISSOCIATAIVE AND SOMATOFORM DISORDERS
          25    PERSONALITY DISORDERS
          26    SEXUAL AND GENDER IDENTITY DISORDERS
          27    ALCOHOL DEPENDENCE SYNDROME
          28    DRUG DEPENDENCE
          29    NONDEPENDENT ABUSE OF DRUGS
          30    PHYSIOLOGICAL MALFUNCTION ARISING FROM MENTAL FACTORS
          31    SPECIAL SYMPTOMS OR SYNDROMES, NOT ELSEWHERE CLASSIFIED
          32    ACUTE REACTION TO STRESS
          33    LEPROSY, UNSPECIFIED
          34    ADJUSTMENT REACTION
          35    ADJUSTMENT REACTION
          36    SPECIFIC NONPSYCHOTIC MENTAL DISORDERS DUE TO BRAIN DAMAGE
          37    DEPRESSIVE DISORDER, NOT ELSEWHERE CLASSIFIED
          38    DISTURBANCE OF CONDUCT, NOT ELSEWHERE CLASSIFIED
          39    DESSEMINATED DISEASES DUE TO OTHER MYCOBACTERIA
```

Check the log. We want to see 32 rows of data in our final merge. Unfortunately, what we have is the total of the two tables. 32 rows from demo_dx and 33 rows from the diagnosis table. The final table contains 52 rows, not 32.



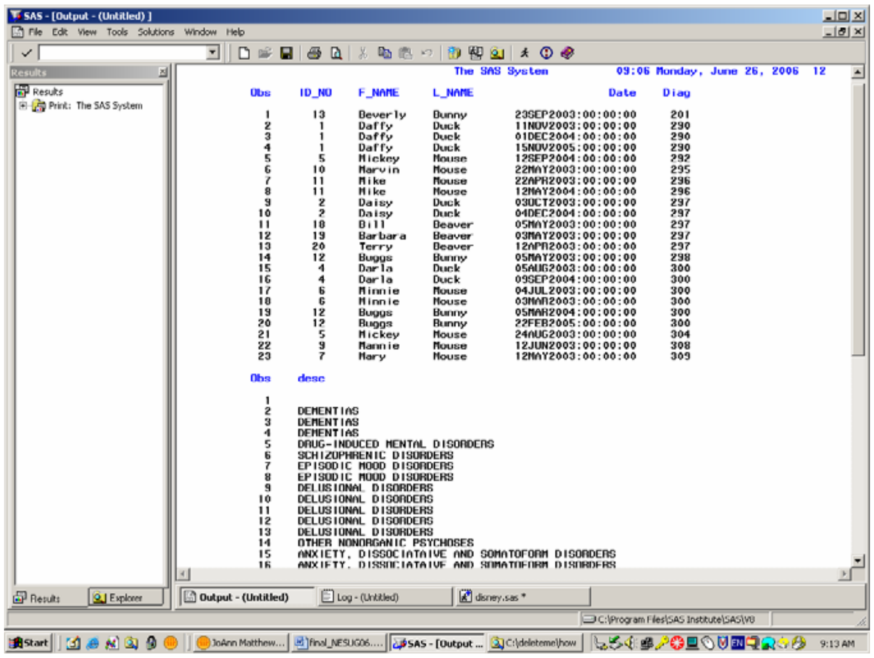
```
      proc sort;
      By diag;
      run;

NOTE: There were 33 observations read from the data set WORK.B.
NOTE: The data set WORK.B has 33 observations and 2 variables.
NOTE: PROCEDURE SORT used:
      real time           0.01 seconds
      cpu time            0.01 seconds



      proc print data=b;
      Run;

NOTE: There were 33 observations read from the data set WORK.B.
NOTE: PROCEDURE PRINT used:
      real time           0.01 seconds
      cpu time            0.01 seconds



      Data all;
      Merge a b;
      By diag;
      Run;

NOTE: MERGE statement has more than one data set with repeats of BY values.
NOTE: There were 32 observations read from the data set WORK.A.
NOTE: There were 33 observations read from the data set WORK.B.
NOTE: The data set WORK.ALL has 52 observations and 6 variables.
NOTE: DATA statement used:
      real time           0.00 seconds
      cpu time            0.00 seconds



      Proc print data = all;
      Run;

NOTE: There were 52 observations read from the data set WORK.ALL.
NOTE: PROCEDURE PRINT used:
      real time           0.00 seconds
      cpu time            0.00 seconds
```

NOTE: Lines have been cleared.

In order to correct the data step, we need to add the IN = option to our merged data. The IN = option creates a new temporary variable that allows you to select only those observations that are on the data set A. Go back to the program editor window, and add two more lines of code (in red), then execute.

```
Data all;
Merge a (in=x) b (in=y);
By diag;
if x;
Run
```

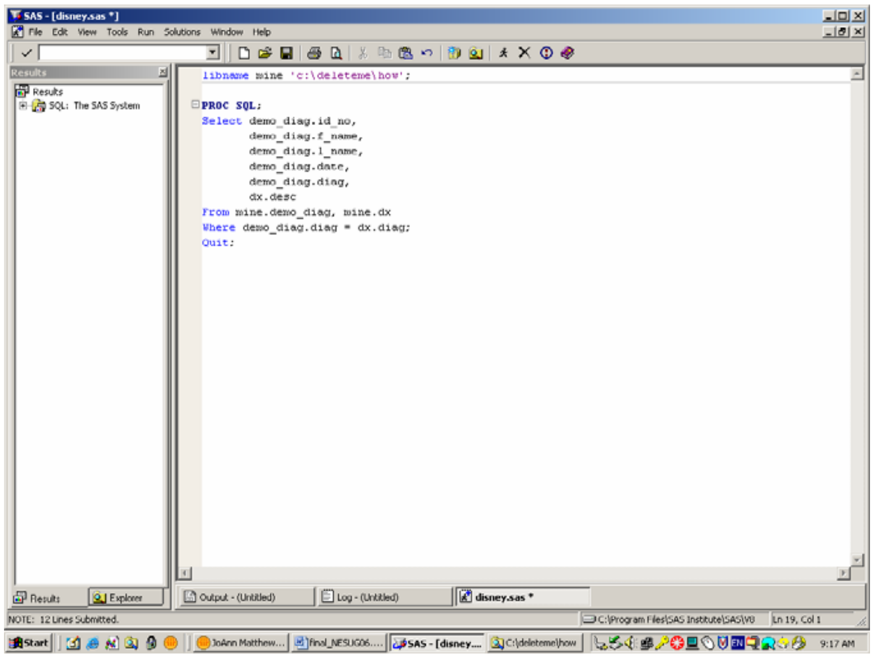
Check the log and confirm that you have only 32 observations. Now look at the output. The data is clean and the results are the correct number of rows (n=32).



```
                                The SAS System        09:06 Monday, June 26, 2006  12

     Obs    ID_NO    F_NAME     L_NAME                     Date    Diag

       1      13     Beverly    Bunny      23SEP2003:00:00:00     201
       2       1     Daffy      Duck       11NOV2003:00:00:00     290
       3       1     Daffy      Duck       01DEC2004:00:00:00     290
       4       1     Daffy      Duck       15NOV2005:00:00:00     290
       5       5     Mickey     Mouse      12SEP2004:00:00:00     292
       6      10     Marvin     Mouse      22MAY2003:00:00:00     295
       7      11     Mike       Mouse      22APR2003:00:00:00     296
       8      11     Mike       Mouse      12MAY2004:00:00:00     296
       9       2     Daisy      Duck       03OCT2003:00:00:00     297
      10       2     Daisy      Duck       04DEC2004:00:00:00     297
      11      18     Bill       Beaver     05MAY2003:00:00:00     297
      12      19     Barbara    Beaver     03MAY2003:00:00:00     297
      13      20     Terry      Beaver     12APR2003:00:00:00     297
      14      12     Buggs      Bunny      05MAY2003:00:00:00     298
      15       4     Darla      Duck       05AUG2003:00:00:00     300
      16       4     Darla      Duck       09SEP2004:00:00:00     300
      17       6     Minnie     Mouse      04JUL2003:00:00:00     300
      18       6     Minnie     Mouse      03MAR2003:00:00:00     300
      19      12     Buggs      Bunny      05MAR2004:00:00:00     300
      20      12     Buggs      Bunny      22FEB2005:00:00:00     300
      21       5     Mickey     Mouse      24AUG2003:00:00:00     304
      22       9     Mannie     Mouse      12JUN2003:00:00:00     308
      23       7     Mary       Mouse      12MAY2003:00:00:00     309

     Obs    desc

       1
       2    DEMENTIAS
       3    DEMENTIAS
       4    DEMENTIAS
       5    DRUG-INDUCED MENTAL DISORDERS
       6    SCHIZOPHRENIC DISORDERS
       7    EPISODIC MOOD DISORDERS
       8    EPISODIC MOOD DISORDERS
       9    DELUSIONAL DISORDERS
      10    DELUSIONAL DISORDERS
      11    DELUSIONAL DISORDERS
      12    DELUSIONAL DISORDERS
      13    DELUSIONAL DISORDERS
      14    OTHER NONORGANIC PSYCHOSES
      15    ANXIETY, DISSOCIATAIVE AND SOMATOFORM DISORDERS
      16    ANXIETY, DISSOCIATAIVE AND SOMATOFORM DISORDERS
```

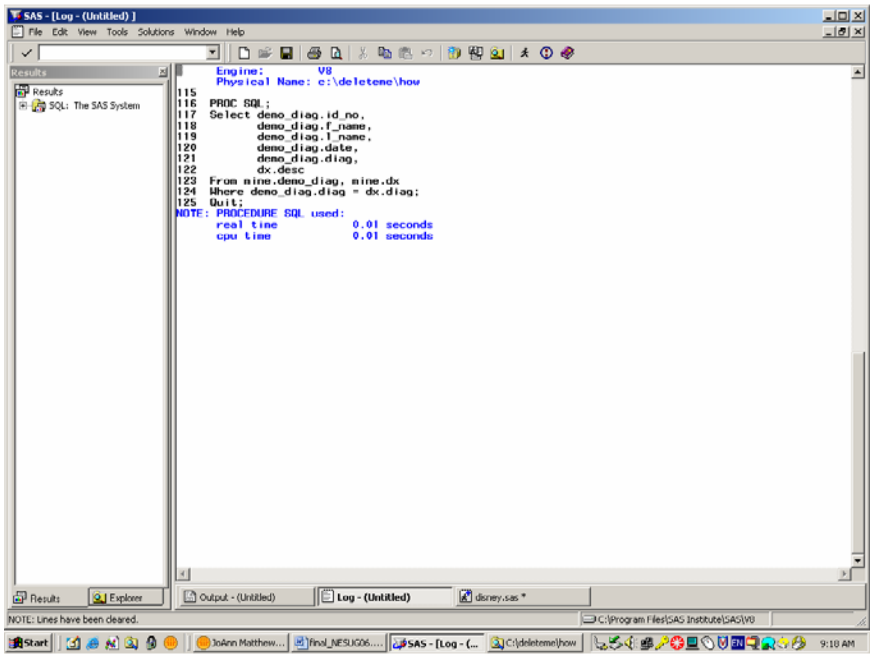
We can accomplish this same result with a simple PROC SQL and the code is significantly shorter. Go to the Program Editor, clear the log and output and type the following code: Notice how we are specifically naming the variables depending on what table they are being queried from – i.e. demo_diag.id_no. *(Give me id_no from the mine.demo_diag table, please).*

```
PROC SQL;
Select demo_diag.id_no,
       demo_diag.f_name,
       demo_diag.l_name,
         demo_diag.date,
         demo_diag.diag,
         dx.desc
From mine.demo_diag, mine.dx
Where demo_diag.diag = dx.diag;
Quit;
```



```
libname mine 'c:\deleteme\how';

PROC SQL;
Select demo_diag.id_no,
       demo_diag.f_name,
       demo_diag.l_name,
       demo_diag.date,
       demo_diag.diag,
       dx.desc
From mine.demo_diag, mine.dx
Where demo_diag.diag = dx.diag;
Quit;
```

Notice the log. It does not display the number of observations. You need to know your data and what number of observations you would expect, when using PROC SQL.

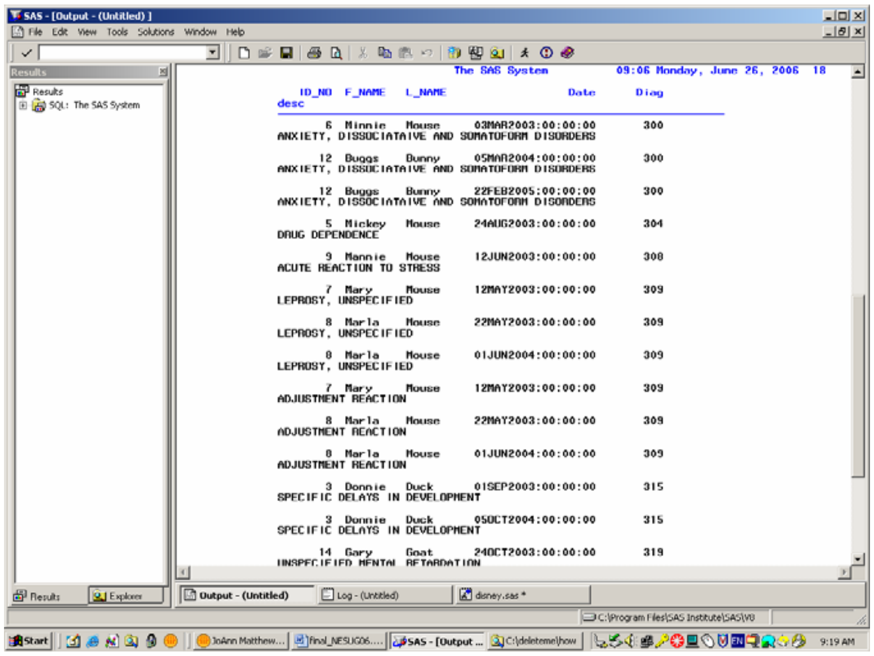


```
      Engine:        V8
      Physical Name: c:\deleteme\how
115
116   PROC SQL;
117   Select demo_diag.id_no,
118          demo_diag.f_name,
119          demo_diag.l_name,
120          demo_diag.date,
121          demo_diag.diag,
122          dx.desc
123   From mine.demo_diag, mine.dx
124   Where demo_diag.diag = dx.diag;
125   Quit;
NOTE: PROCEDURE SQL used:
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Note also that this is an example of an inner join. An inner join is the result of variables that are in one table that have a match in a second table. These are also called equijoins. An equijoin combines tables based on a common variable in both tables, eliminating the redundant columns.

i.e.   Where a.id = b.id

And your output should look like this:



```
                      The SAS System          09:06 Monday, June 26, 2006  18

   ID_NO  F_NAME   L_NAME                  Date       Diag
desc
--------------------------------------------------------------
       6  Minnie   Mouse     03MAR2003:00:00:00        300
ANXIETY, DISSOCIATAIVE AND SOMATOFORM DISORDERS

      12  Buggs    Bunny     05MAR2004:00:00:00        300
ANXIETY, DISSOCIATAIVE AND SOMATOFORM DISORDERS

      12  Buggs    Bunny     22FEB2005:00:00:00        300
ANXIETY, DISSOCIATAIVE AND SOMATOFORM DISORDERS

       5  Mickey   Mouse     24AUG2003:00:00:00        304
DRUG DEPENDENCE

       9  Mannie   Mouse     12JUN2003:00:00:00        308
ACUTE REACTION TO STRESS

       7  Mary     Mouse     12MAY2003:00:00:00        309
LEPROSY, UNSPECIFIED

       8  Marla    Mouse     22MAY2003:00:00:00        309
LEPROSY, UNSPECIFIED

       8  Marla    Mouse     01JUN2004:00:00:00        309
LEPROSY, UNSPECIFIED

       7  Mary     Mouse     12MAY2003:00:00:00        309
ADJUSTMENT REACTION

       8  Marla    Mouse     22MAY2003:00:00:00        309
ADJUSTMENT REACTION

       8  Marla    Mouse     01JUN2004:00:00:00        309
ADJUSTMENT REACTION

       3  Donnie   Duck      01SEP2003:00:00:00        315
SPECIFIC DELAYS IN DEVELOPMENT

       3  Donnie   Duck      05OCT2004:00:00:00        315
SPECIFIC DELAYS IN DEVELOPMENT

      14  Gary     Goat      24OCT2003:00:00:00        319
UNSPECIFIED MENTAL RETARDATION
```

Keep in mind that you can join more than one table at a time. In fact PROC SQL can join up to 32 tables. The syntax for two tables would look like this:

```
      PROC SQL;
      SELECT *
      from mine.demographic, mine.rem_sleep
      WHERE demographic.id_no = rem_sleep.id_no
        and demographic.place = rem_sleep.place;
      quit;
```

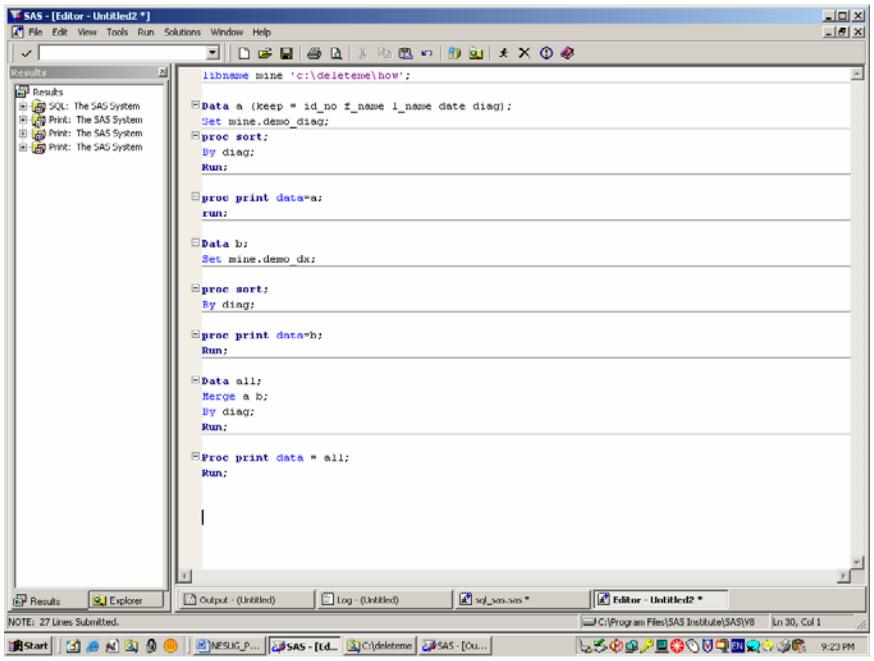
While both the data step and PROC SQL can be used for one to many merges, where PROC SQL becomes very powerful is when accessing huge datasets. When using small datasets, the data step may be just as efficient as PROC SQL but when accessing huge datasets or large tables with many rows and many columns, PROC SQL is more powerful and generally uses less resources.

MANY TO MANY MATCHES

Let's use a new rem sleep dataset that contains multiple sleep evaluations for each of our Disney families. The study is now several years in funding, and every participant was asked to complete a second and third sleep time to establish average REM times. Some families were able and some were not. We need to merge the two tables of multiple sleep REM values. This is an example of many to many merge. Use PROC SQL when the merge is a many to many merge and especially when some data is missing. Go into the program editor, clear the log and output window, and type the following code, then execute the code by using the submit icon on your toolbar.

```
libname mine 'c:\deleteme\how';

Data a (keep = id_no f_name l_name date diag);
Set mine.demo_diag;
proc sort;
By id_no;
Run;

proc print data=a;
run;

Data b;
Set mine.rem_two;
```

```
proc sort;
By id_no;

proc print data=b;
Run;

Data all;
Merge a (in=x) b (in=y);
By id_no;
Run;

Proc print data = all;
Run;
```


SAS - [Editor - Untitled2 *]
File Edit View Tools Run Solutions Window Help
Results
SQL: The SAS System
Print: The SAS System
Print: The SAS System
Print: The SAS System
Results Explorer
Output - (Untitled) Log - (Untitled) sql_sas.sas * Editor - Untitled2 *
C:\Program Files\SAS Institute\SAS\V8 Ln 30, Col 1
9:23 PM

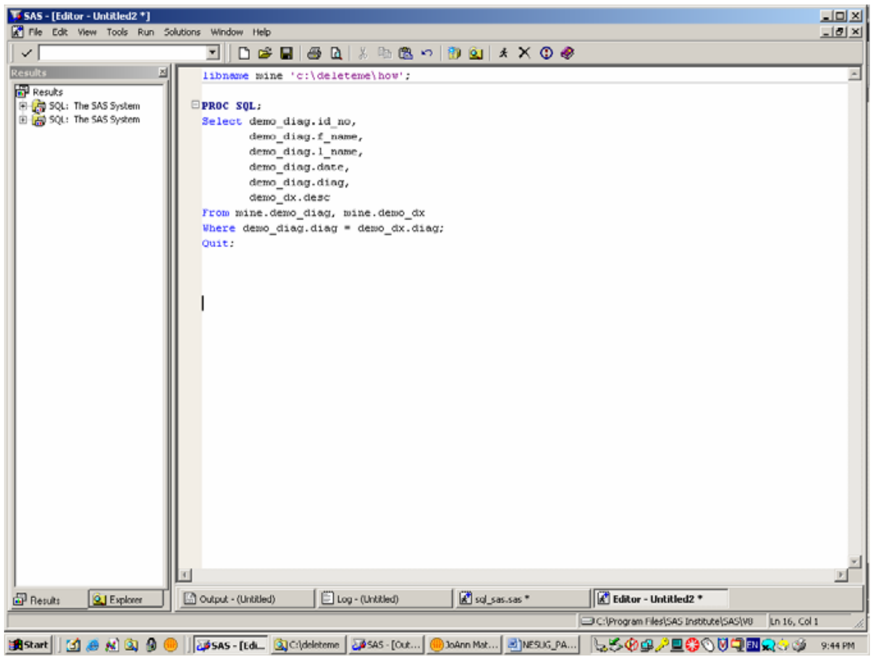
```
libname mine 'c:\deleteme\how';

Data a (keep = id_no f_name l_name date diag);
Set mine.demo_diag;
proc sort;
By diag;
Run;

proc print data=a;
run;

Data b;
Set mine.demo_dx;

proc sort;
By diag;

proc print data=b;
Run;

Data all;
Merge a b;
By diag;
Run;

Proc print data = all;
Run;
```

NOTE: 27 Lines Submitted.

What you get is messy and incorrect. The best way to do a MANY to MANY MATCH is with PROC SQL. This is particularly true if some of the values are missing, as PROC SQL will adjust for this condition.

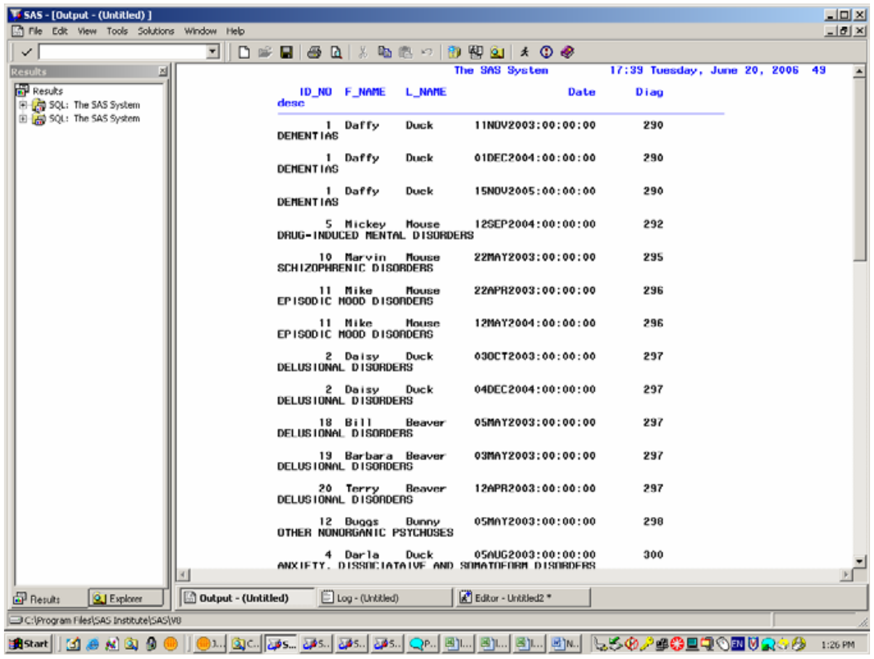
Let's do a many to many merge with PROC SQL using the WHERE diag on demo_diag matches diag on demo_dx. Go into the program editor, clear the log and output window, and type the following code, then execute the code by using the submit icon on your toolbar.

```
PROC SQL;
      SELECT      demo_diag.id_no,
                  demo_diag.f_name,
                  demo_diag.l_name,
                  demo_diag.date,
                  demo_diag.diag,
                  demo_dx.desc
      From mine.demo_diag, mine.demo_dx
      Where demo_diag.diag = demo_dx.diag;
      quit;
```



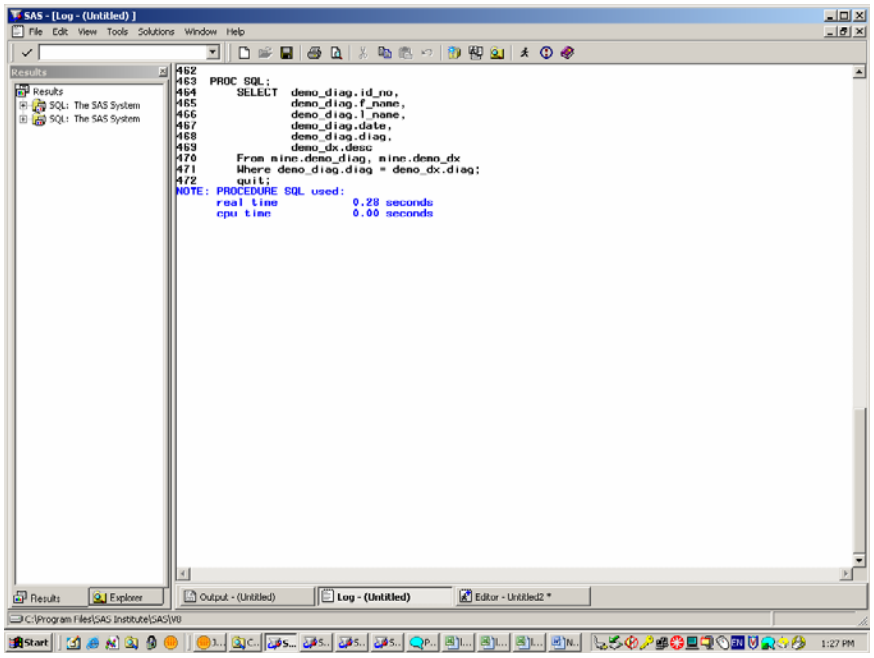
```
libname mine 'c:\deleteme\how';

PROC SQL;
Select demo_diag.id_no,
       demo_diag.f_name,
       demo_diag.l_name,
       demo_diag.date,
       demo_diag.diag,
       demo_dx.desc
From mine.demo_diag, mine.demo_dx
Where demo_diag.diag = demo_dx.diag;
Quit;
```

This output is correct with correct diagnoses for each member of the study.



```
                          The SAS System          17:39 Tuesday, June 20, 2006  49

    ID_NO  F_NAME   L_NAME                   Date       Diag
desc
------------------------------------------------------------
        1  Daffy    Duck      11NOV2003:00:00:00        290
DEMENTIAS

        1  Daffy    Duck      01DEC2004:00:00:00        290
DEMENTIAS

        1  Daffy    Duck      15NOV2005:00:00:00        290
DEMENTIAS

        5  Mickey   Mouse     12SEP2004:00:00:00        292
DRUG-INDUCED MENTAL DISORDERS

       10  Marvin   Mouse     22MAY2003:00:00:00        295
SCHIZOPHRENIC DISORDERS

       11  Mike     Mouse     22APR2003:00:00:00        296
EPISODIC MOOD DISORDERS

       11  Mike     Mouse     12MAY2004:00:00:00        296
EPISODIC MOOD DISORDERS

        2  Daisy    Duck      03OCT2003:00:00:00        297
DELUSIONAL DISORDERS

        2  Daisy    Duck      04DEC2004:00:00:00        297
DELUSIONAL DISORDERS

       18  Bill     Beaver    05MAY2003:00:00:00        297
DELUSIONAL DISORDERS

       19  Barbara  Beaver    03MAY2003:00:00:00        297
DELUSIONAL DISORDERS

       20  Terry    Beaver    12APR2003:00:00:00        297
DELUSIONAL DISORDERS

       12  Buggs    Bunny     05MAY2003:00:00:00        298
OTHER NONORGANIC PSYCHOSES

        4  Darla    Duck      05AUG2003:00:00:00        300
ANXIETY, DISSOCIATAIVE AND SOMATOFORM DISORDERS
```

Let's look at the log.

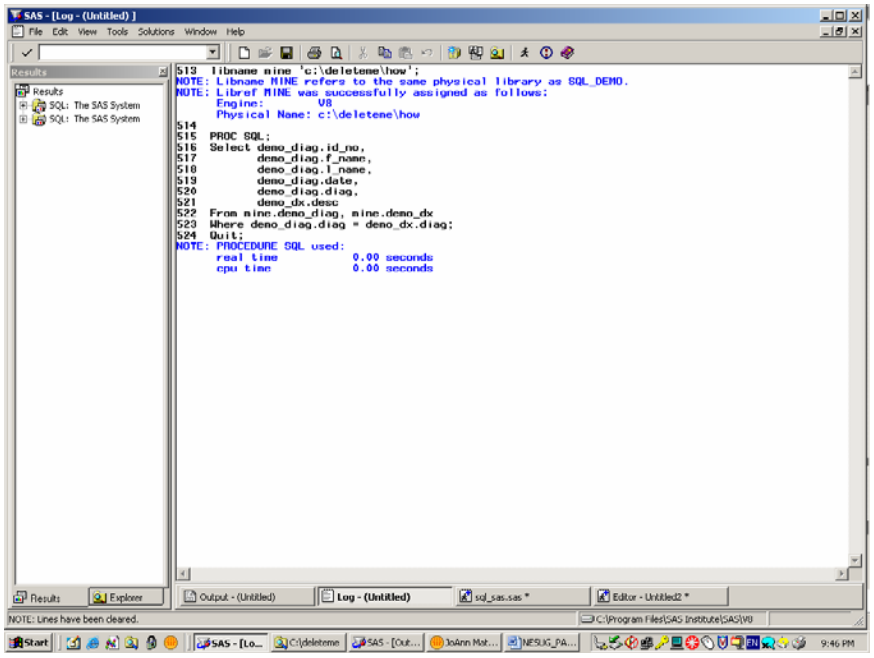


```
462
463  PROC SQL;
464      SELECT  demo_diag.id_no,
465              demo_diag.f_name,
466              demo_diag.l_name,
467              demo_diag.date,
468              demo_diag.diag,
469              demo_dx.desc
470      From mine.demo_diag, mine.demo_dx
471      Where demo_diag.diag = demo_dx.diag;
472      quit;
NOTE: PROCEDURE SQL used:
      real time           0.28 seconds
      cpu time            0.00 seconds
```

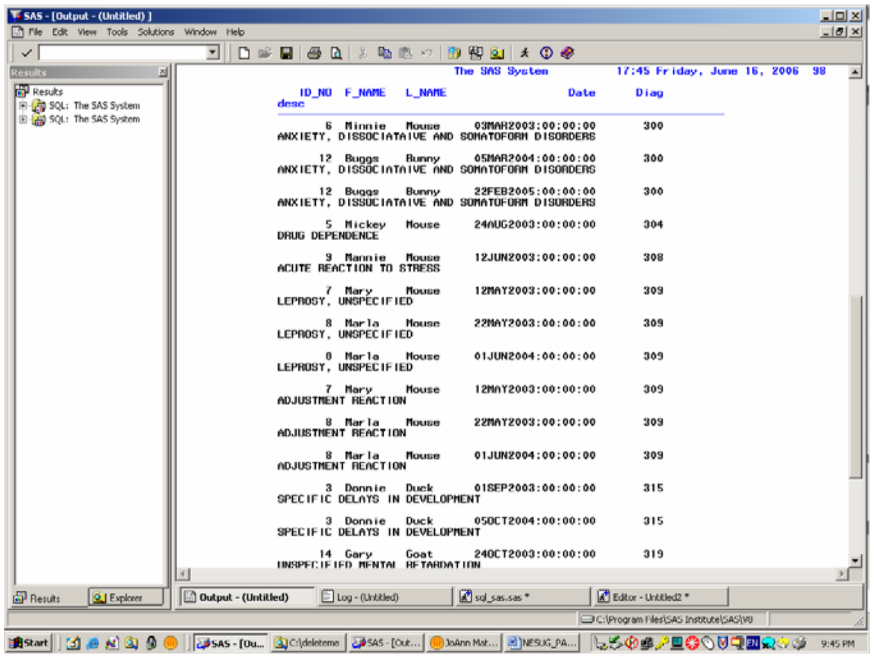
Notice the log for PROC SQL does not confirm the number of observations, as the log does when using the data step! This can be a real disadvantage of SQL. With the data step you see exactly how many observations were read in for each data step. Checking the log can help you to be assured that the data you are getting is the data you really want. With SQL, YOU MUST KNOW YOUR DATA! If not, use only a few observations, using the INOBS = n option on the PROC SQL statement,

i.e. **PROC SQL inobs = 10;**

The INOBS = option restricts the number of rows (observations) that PROC SQL will process.
Check the log:



```
513  libname mine 'c:\deleteme\how';
NOTE: Libname MINE refers to the same physical library as SQL_DEMO.
NOTE: Libref MINE was successfully assigned as follows:
      Engine:        V8
      Physical Name: c:\deleteme\how
514
515  PROC SQL;
516  Select demo_diag.id_no,
517         demo_diag.f_name,
518         demo_diag.l_name,
519         demo_diag.date,
520         demo_diag.diag,
521         demo_dx.desc
522  From mine.demo_diag, mine.demo_dx
523  Where demo_diag.diag = demo_dx.diag;
524  Quit;
NOTE: PROCEDURE SQL used:
      real time           0.00 seconds
      cpu time            0.00 seconds
```



```
                          The SAS System           17:45 Friday, June 16, 2006  98

   ID_NO  F_NAME   L_NAME                      Date       Diag
desc
-----------------------------------------------------------------------

       6  Minnie   Mouse     03MAR2003:00:00:00        300
ANXIETY, DISSOCIATAIVE AND SOMATOFORM DISORDERS

      12  Buggs    Bunny     05MAR2004:00:00:00        300
ANXIETY, DISSOCIATAIVE AND SOMATOFORM DISORDERS

      12  Buggs    Bunny     22FEB2005:00:00:00        300
ANXIETY, DISSOCIATAIVE AND SOMATOFORM DISORDERS

       5  Mickey   Mouse     24AUG2003:00:00:00        304
DRUG DEPENDENCE

       9  Mannie   Mouse     12JUN2003:00:00:00        308
ACUTE REACTION TO STRESS

       7  Mary     Mouse     12MAY2003:00:00:00        309
LEPROSY, UNSPECIFIED

       8  Marla    Mouse     22MAY2003:00:00:00        309
LEPROSY, UNSPECIFIED

       8  Marla    Mouse     01JUN2004:00:00:00        309
LEPROSY, UNSPECIFIED

       7  Mary     Mouse     12MAY2003:00:00:00        309
ADJUSTMENT REACTION

       8  Marla    Mouse     22MAY2003:00:00:00        309
ADJUSTMENT REACTION

       8  Marla    Mouse     01JUN2004:00:00:00        309
ADJUSTMENT REACTION

       3  Donnie   Duck      01SEP2003:00:00:00        315
SPECIFIC DELAYS IN DEVELOPMENT

       3  Donnie   Duck      05OCT2004:00:00:00        315
SPECIFIC DELAYS IN DEVELOPMENT

      14  Gary     Goat      24OCT2003:00:00:00        319
UNSPECIFIED MENTAL RETARDATION
```

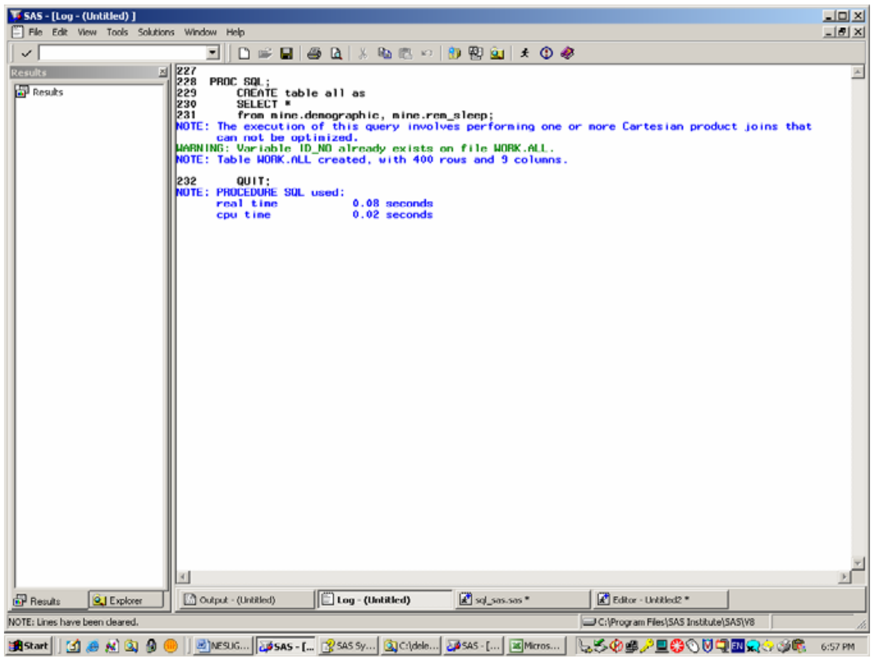
CARTESIAN PRODUCT
If you join two tables without using the WHERE clause, you will get the product of the two numbers (a Cartesian product). 20 x 20 = 100. Generally, a Cartesian join is accidental. Usually if you want to join two tables you are intending to do a concatenation, using SET. An accidental Cartesian, when done on a large table can bring your system to it's knees. 10,000 rows x 10,000 rows may be more than your system resource allocation will allow.

If your intent is to accomplish a Cartesian product ,this is difficult to do using traditional data step processing. For example a table with 20 observations in one table, and 20 observations in the other table will result in 400 observations in total. Cartesian products can best be accomplished by the use of PROC SQL. Here is an example of a Cartesian product but PROC SQL was smart enough to know this isn't what we really want to do, because we should be using a where statement.

Let's make an intentional programming error. Go to the SAS Editor window, clear the log and type the following code, then execute the code by using the submit icon on your toolbar.

```
PROC SQL;
      CREATE table all as
      SELECT *
      from mine.demographic, mine.rem_sleep;
      QUIT;
```

Since id_no appears on both tables, you will see an error message as a result of trying to do a Cartesian product (a join without a where clause) on two datasets, a many to many merge. Most Cartesian products are accidental and are resource intensive.



```
227
228  PROC SQL;
229      CREATE table all as
230      SELECT *
231      from mine.demographic, mine.rem_sleep;
NOTE: The execution of this query involves performing one or more Cartesian product joins that
      can not be optimized.
WARNING: Variable ID_NO already exists on file WORK.ALL.
NOTE: Table WORK.ALL created, with 400 rows and 9 columns.

232      QUIT;
NOTE: PROCEDURE SQL used:
      real time           0.08 seconds
      cpu time            0.02 seconds
```

NOTE: Lines have been cleared.

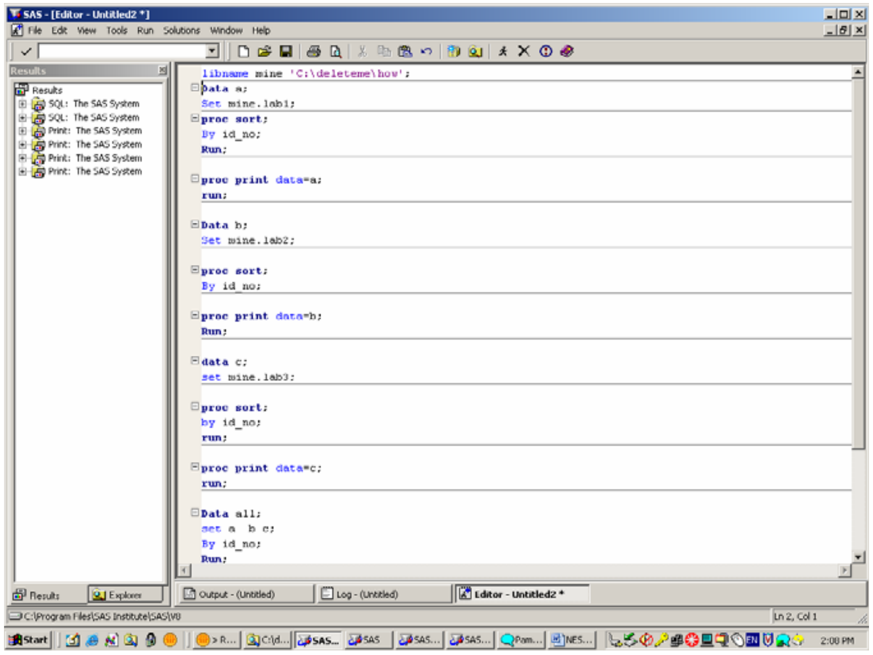
OUTER JOINS
In contract to an inner join, an outer join retrieves records from both tables but only returns those observations that do not match the first table. When you join two tables, the first one may have rows that don't have matching counterparts in the second table. Conversely, the table on the right may have rows that don't have a match in the table on the left. If you do an inner join on those tables all the unmatched rows are excluded. There are three types of outer join, left outer, right outer and full outer join. The syntax is similar to the inner join, except that you substitute outer for inner. The left outer join preserves unmatched rows from the left table but discards unmatched rows from the right table. Right outer joins preserves unmatched rows from the right table but discards unmatched rows from the left table. You can use this on the same tables and get the same result by reversing the order in which you present the tables in the join: Full outer joins combine the functions of the left outer join and the right outer join. It retains the unmatched rows from both the left and the right tables.

UNION ALL
Suppose you have three tables that you wish to concatenate. You can use the SET statement in SAS® to concatenate the tables. Or you can use the UNION ALL, as long as all the variables in all the tables are input in the same position. This type of union creates another table that has everything, all the columns in all the source tables.

The Disney participants used three separate sleep labs. The data for those three labs needs to be combined into one table. LAB1 was housed in Magic Kingdom and contains 11 sleep evaluations (rows of data) for the Mouse family. LAB2 was in Epcot Center and contains 9 sleep evaluations (9 rows) for the Duck family. The third sleep lab was in Animal Kingdom and contains 12 sleep evaluations for the Bunny, Goat and Beaver families. Using a traditional data step to concatenate we would use the SET, instead of MERGE. Go into the program editor, clear the log and output window, and type the following code, then execute the code by using the submit icon on your toolbar.

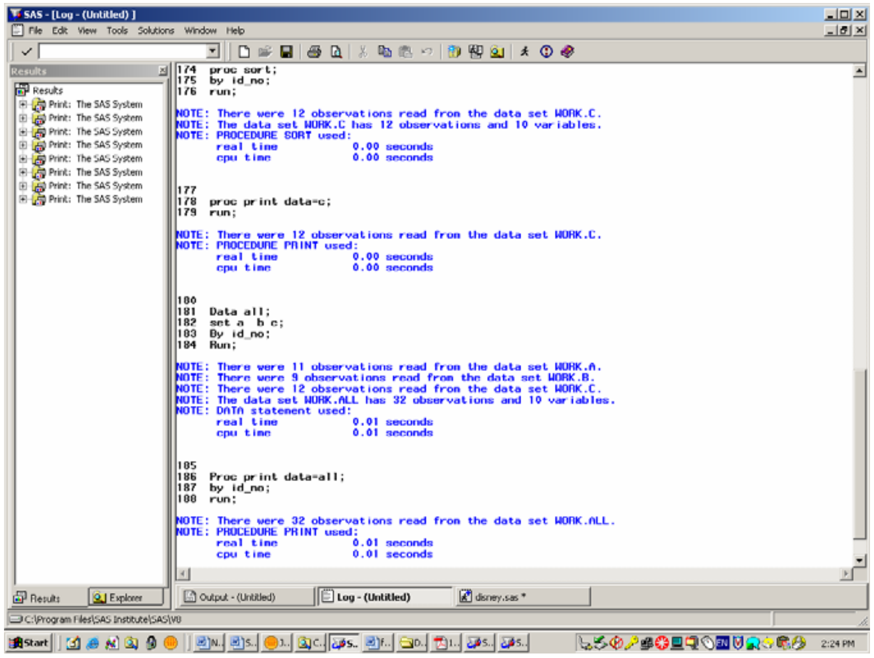
```
libname mine 'c:\deleteme\how';

Data a;
Set mine.lab1;
proc sort;
By id_no;
Run;
```

```
proc print data=a;
run;

Data b;
Set mine.lab2;

proc sort;
By id_no;

proc print data=b;
Run;

data c;
set mine.lab3;

proc sort;
by id_no;
run;

proc print data=c;
run;

Data all;
set a  b c;
By id_no;
Run;

Proc print data=all;
by id_no;
run;
```



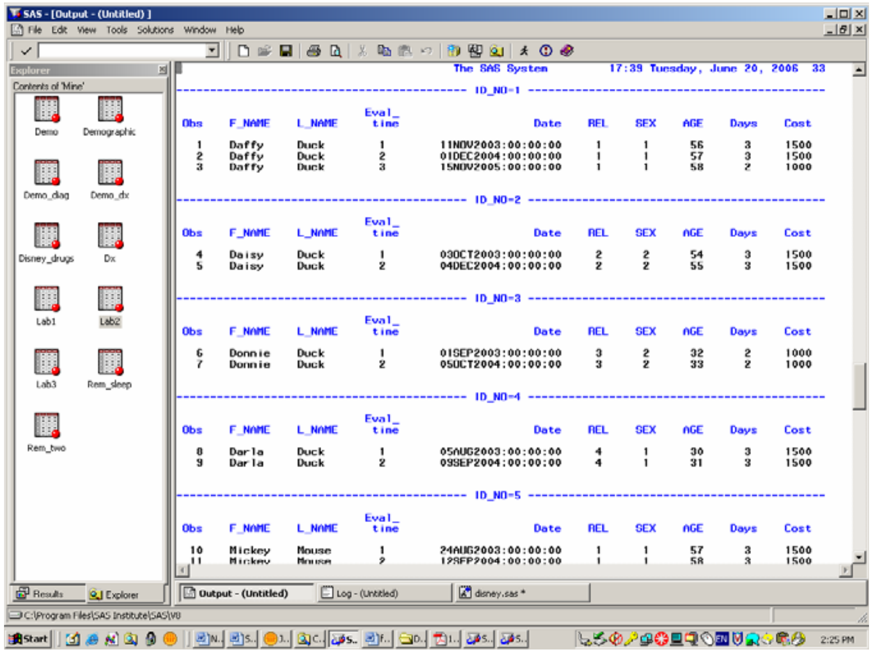
```
libname mine 'C:\deleteme\how';
Data a;
Set mine.lab1;
proc sort;
By id_no;
Run;

proc print data=a;
run;

Data b;
Set mine.lab2;

proc sort;
By id_no;

proc print data=b;
Run;

data c;
set mine.lab3;

proc sort;
by id_no;
run;

proc print data=c;
run;

Data all;
set a  b c;
By id_no;
Run;
```

Always check the log when running data steps. Let’s examine the log.



```
proc sort;
by id_no;
run;

NOTE: There were 12 observations read from the data set WORK.C.
NOTE: The data set WORK.C has 12 observations and 10 variables.
NOTE: PROCEDURE SORT used:
      real time           0.00 seconds
      cpu time            0.00 seconds



proc print data=c;
run;

NOTE: There were 12 observations read from the data set WORK.C.
NOTE: PROCEDURE PRINT used:
      real time           0.00 seconds
      cpu time            0.00 seconds



Data all;
set a  b c;
By id_no;
Run;

NOTE: There were 11 observations read from the data set WORK.A.
NOTE: There were 9 observations read from the data set WORK.B.
NOTE: There were 12 observations read from the data set WORK.C.
NOTE: The data set WORK.ALL has 32 observations and 10 variables.
NOTE: DATA statement used:
      real time           0.01 seconds
      cpu time            0.01 seconds



Proc print data=all;
by id_no;
run;

NOTE: There were 32 observations read from the data set WORK.ALL.
NOTE: PROCEDURE PRINT used:
      real time           0.01 seconds
      cpu time            0.01 seconds
```

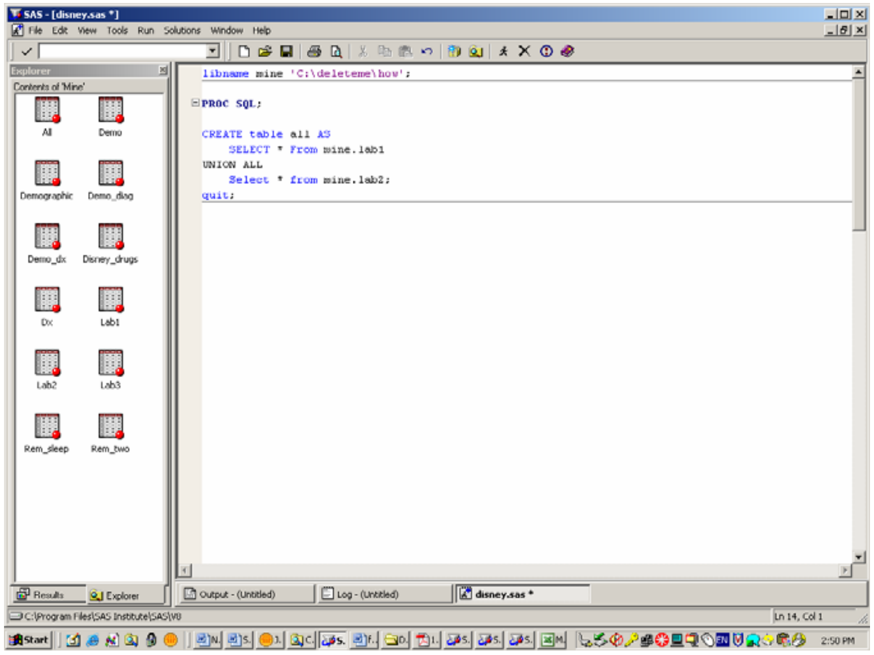
Observe the number of observations in the final table ALL. We would expect to see 32 observations, 11 from A, 9 from B, and 12 from C. Look at the output window and observe that the final dataset appears clean and concatenated.



```
                                      The SAS System        17:39 Tuesday, June 20, 2006  33

----------------------------------------- ID_NO=1 ------------------------------------------

                            Eval_
Obs   F_NAME   L_NAME       time                  Date   REL   SEX   AGE   Days   Cost

  1   Daffy    Duck           1     11NOV2003:00:00:00    1     1    56     3     1500
  2   Daffy    Duck           2     01DEC2004:00:00:00    1     1    57     3     1500
  3   Daffy    Duck           3     15NOV2005:00:00:00    1     1    58     2     1000

----------------------------------------- ID_NO=2 ------------------------------------------

                            Eval_
Obs   F_NAME   L_NAME       time                  Date   REL   SEX   AGE   Days   Cost

  4   Daisy    Duck           1     03OCT2003:00:00:00    2     2    54     3     1500
  5   Daisy    Duck           2     04DEC2004:00:00:00    2     2    55     3     1500

----------------------------------------- ID_NO=3 ------------------------------------------

                            Eval_
Obs   F_NAME   L_NAME       time                  Date   REL   SEX   AGE   Days   Cost

  6   Donnie   Duck           1     01SEP2003:00:00:00    3     2    32     2     1000
  7   Donnie   Duck           2     05OCT2004:00:00:00    3     2    33     2     1000

----------------------------------------- ID_NO=4 ------------------------------------------

                            Eval_
Obs   F_NAME   L_NAME       time                  Date   REL   SEX   AGE   Days   Cost

  8   Darla    Duck           1     05AUG2003:00:00:00    4     1    30     3     1500
  9   Darla    Duck           2     09SEP2004:00:00:00    4     1    31     3     1500

----------------------------------------- ID_NO=5 ------------------------------------------

                            Eval_
Obs   F_NAME   L_NAME       time                  Date   REL   SEX   AGE   Days   Cost

 10   Mickey   Mouse          1     24AUG2003:00:00:00    1     1    57     3     1500
 11   Mickey   Mouse          2     12SEP2004:00:00:00    1     1    58     3     1500
```

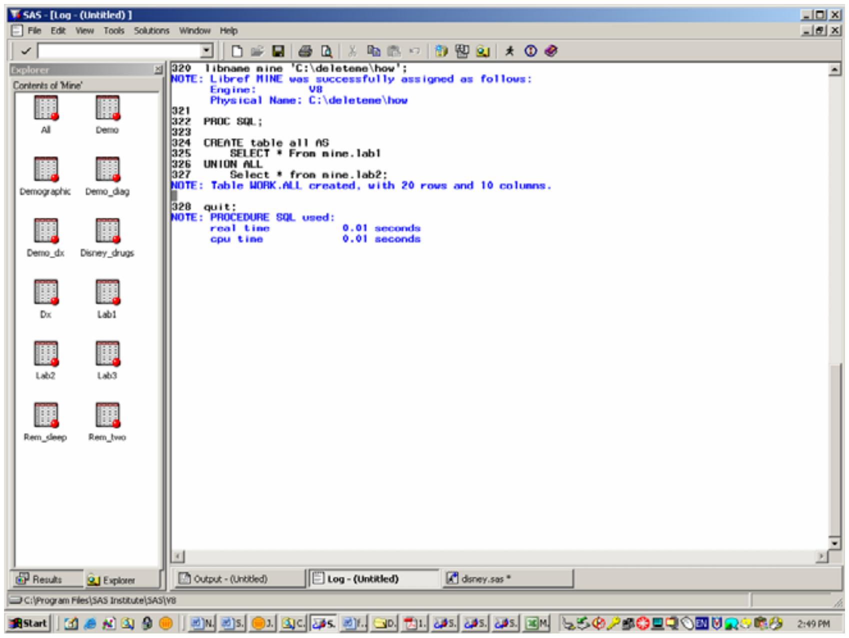
Now let’s use a UNION ALL in PROC SQL. One quick and dirty way to do a UNION ALL is to simply use the asterisk function that says “*Give me all the variables*”. This works beautifully if both tables have the exact same format. If the tables are union compatible, i.e share the same format, the resultant table would be all the rows in the first table, plus all the rows from the second table. Go into the program editor, clear the log and output window, and type the following code, then execute the code by using the submit icon on your toolbar.

```
libname mine 'C:\deleteme\how';

PROC SQL;

CREATE table all AS
      SELECT * From mine.lab1

UNION ALL
      Select * from mine.lab2;

UNION ALL
      Select * from mine.lab3;
quit;
```



```
libname mine 'C:\deleteme\how';

PROC SQL;

CREATE table all AS
    SELECT * From mine.lab1
UNION ALL
    Select * from mine.lab2;
quit;
```

Always check your log. Make sure that the row counts are what you expect.

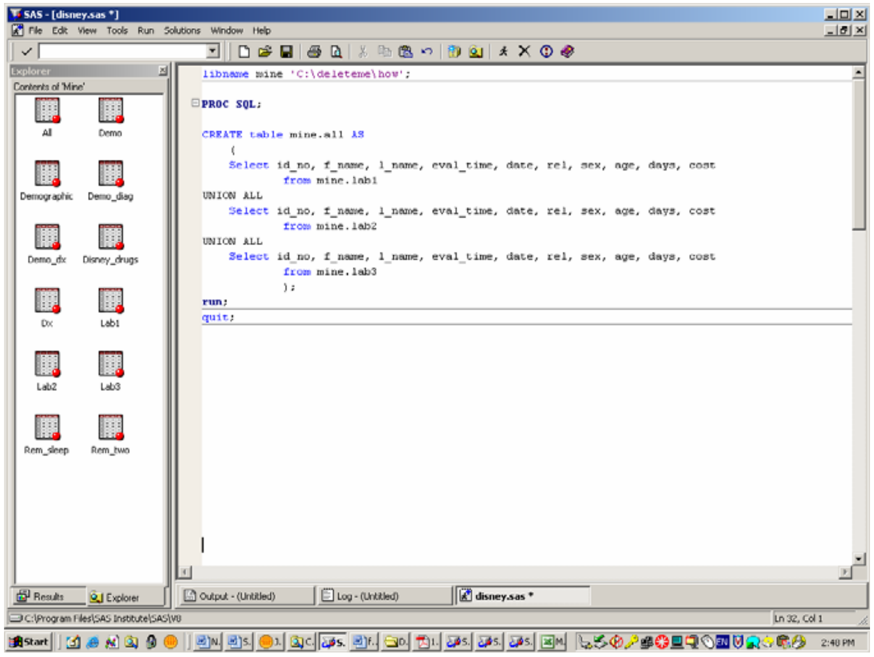


```
320  libname mine 'C:\deleteme\how';
NOTE: Libref MINE was successfully assigned as follows:
      Engine:        V8
      Physical Name: C:\deleteme\how
321
322  PROC SQL;
323
324  CREATE table all AS
325      SELECT * From mine.lab1
326  UNION ALL
327      Select * from mine.lab2;
NOTE: Table WORK.ALL created, with 20 rows and 10 columns.

328  quit;
NOTE: PROCEDURE SQL used:
      real time           0.01 seconds
      cpu time            0.01 seconds
```

If on the other hand, the tables are not union compatible, you can use the UNION ALL and explicitly define the variables that you wish to concatenate. Explicitly listing the columns that you want rather than relying on the * shorthand is usually a good idea. It is quite possible that even though the tables were union compatible when you first ran the query, when running the query later, one of the tables could have been modified. The resultant tables are no longer union compatible. Explicit definition is always safer.

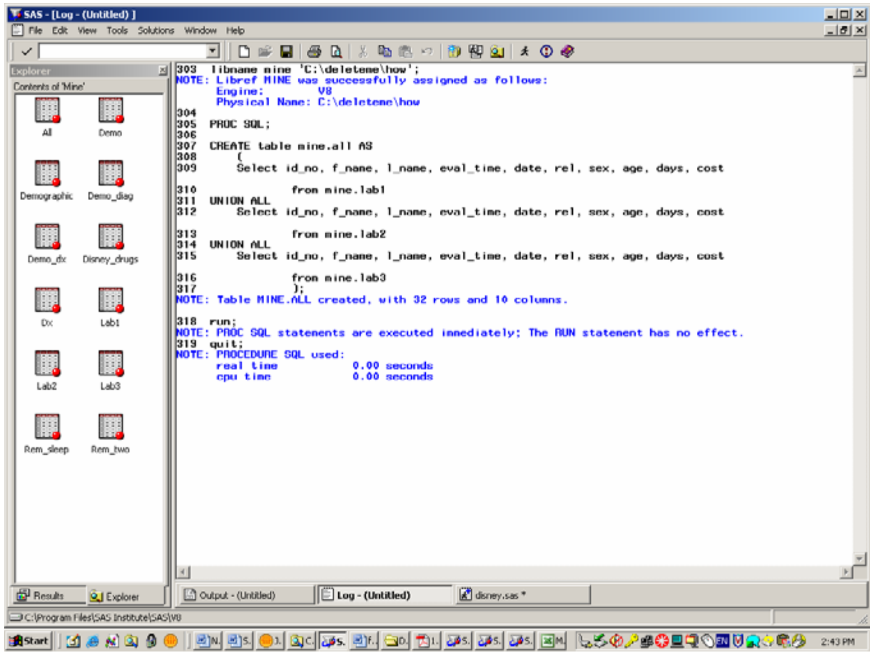
Let's do a join with explicitly defining the variables. Go into the program editor, clear the log and output window, and type the following code, then execute the code by using the submit icon on your toolbar.

```
libname mine 'C:\deleteme\how';

PROC SQL;

CREATE table mine.all AS
(
Select id_no, f_name, l_name, eval_time, date, rel, sex, age, days, cost
           from mine.lab1

UNION ALL
Select id_no, f_name, l_name, eval_time, date, rel, sex, age, days, cost
          from mine.lab2

UNION ALL
Select id_no, f_name, l_name, eval_time, date, rel, sex, age, days, cost
           from mine.lab3
                  );
quit;
```



```
libname mine 'C:\deleteme\how';

PROC SQL;

CREATE table mine.all AS
    (
    Select id_no, f_name, l_name, eval_time, date, rel, sex, age, days, cost
            from mine.lab1
UNION ALL
    Select id_no, f_name, l_name, eval_time, date, rel, sex, age, days, cost
            from mine.lab2
UNION ALL
    Select id_no, f_name, l_name, eval_time, date, rel, sex, age, days, cost
            from mine.lab3
            );
run;
quit;
```

Check the log. With UNION ALL, you will see how many total rows and columns were created.



```
303  libname mine 'C:\deleteme\how';
NOTE: Libref MINE was successfully assigned as follows:
      Engine:        V8
      Physical Name: C:\deleteme\how
304
305  PROC SQL;
306
307  CREATE table mine.all AS
308      (
309      Select id_no, f_name, l_name, eval_time, date, rel, sex, age, days, cost

310                 from mine.lab1
311  UNION ALL
312      Select id_no, f_name, l_name, eval_time, date, rel, sex, age, days, cost

313                 from mine.lab2
314  UNION ALL
315      Select id_no, f_name, l_name, eval_time, date, rel, sex, age, days, cost

316                 from mine.lab3
317                 );
NOTE: Table MINE.ALL created, with 32 rows and 10 columns.

318  run;
NOTE: PROC SQL statements are executed immediately; The RUN statement has no effect.
319  quit;
NOTE: PROCEDURE SQL used:
      real time           0.00 seconds
      cpu time            0.00 seconds
```

UNION DISTINCT
If you want duplicate rows eliminated from the final table use the UNION DISTINCT function. It behaves like the UNION ALL, but returns only non duplicate rows.

```
libname mine 'C:\deleteme\how';

PROC SQL;

CREATE table mine.all AS
(
Select id_no, f_name, l_name, eval_time, date, rel, sex, age, days, cost
           from mine.lab1

UNION DISTINCT
Select id_no, f_name, l_name, eval_time, date, rel, sex, age, days, cost
          from mine.lab2

UNION DISTINCT
Select id_no, f_name, l_name, eval_time, date, rel, sex, age, days, cost
           from mine.lab3
                  );
quit;
```

CONCLUSION:

PROC SQL is a very powerful addition to your SAS bag of tools. When working with huge datasets, the CPU and resource savings can be dramatic. The beauty of PROC SQL is that the data does not need to be sorted like with the traditional data step. This Hands-On Workshop is intended to get you started exploring the SQL procedure in SAS. Each of the PROC SQL statements that we have discussed has many additional capabilities which are beyond the scope of this paper. And while we used the Display Manager in SAS to submit the code, check the log and list file, the PROC SQL syntax will work on any platform, whether you are running on the mainframe, UNIX, or SAS Version 9's Enterprise Guide.

REFERENCES:

Allen G. Taylor, *SQL for Dummies 5th Edition*, Wiley Publishing, Inc. 2003
Michael J. Larkins and Thomas L. Coffing, Jr. , *Teradata SQL, Unleash the Power*, Coffing Publishing, 2002
SAS Institute Inc. (1990) SAS *Language and Procedure Guide, Version 6*, Third Edition, Cary, NC: SAS Institute, Inc.
Lora D. Delwiche and Susan J. Slaughter, *The Little SAS Book: A Primer, Third Edition,* Cary, NC. SAS Institute, Inc., 2003

ACKNOWLEDMENTS

SAS ® and all 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 registered trademarks of their respective companies.

CONTACT INFORMATION

Your comments or questions are encouraged. Please feel free to contact the author at:

JoAnn Matthews
Highmark Inc
120 Fifth Avenue
Pittsburgh PA 15222-3099
e-mail: joann.matthews@highmark.com