ABSTRACT
SQL is a language that is used in several relational database software systems. The basics of SQL extend to most of these packages and are known as ANSI (American National Standards Institute). **SAS® PROC SQL** has the ANSI standards in addition to several of the SAS statements and functions that are particular to the **SAS/BASE**.

Some of these statements and functions are not widely used maybe because there are not enough publications showing advantages for programming, computer optimization and correct implementation.

This paper assumes basic knowledge of **SAS PROC SQL**, and describes some features of the procedure that make this procedure particularly useful and can improve plain vanilla **SAS PROC SQL**.

INTRODUCTION

**SQL** is a procedure that is invoked in **SAS**, and like most procedures it has options. **SAS PROC SQL** has the additional advantages of the **SAS** functions, options and statements. These advantages can be used to 'spice' up your programs and make them more efficient in the same way they are used in Data Step and other procedures.

Data that will be used in this paper

In **SAS PROC SQL**, a **DATA SET** is known as a **TABLE**, an **OBSERVATION** as a **ROW** and a **VARIABLE** as a **COLUMN**. Because of this, the options that are available in data step are available when referring to a table, also, the attributes that can be assigned to a variable can be assigned to a column.

**Data DOGS**

```sas
data DOGS;
  length NAME BREED $20
    DOB        $10
  ;
  input NAME BREED DOB;
  cards;
  Nala Airdale 11jan1990
  Nina AMBR  22apr1997
  Jadzia Pointer 15feb1999
  Rocco MMBNR  08may1980
  Bruja Rottweiler 15aug2000
  Chacal Rottweiler 29jan1990
  Chata Boxer UNK
  Tyson Rottweiler 15mar2000
; run;
```

**Data PARENTS**

```sas
Data PARENTS;
  length PARENT NAME $20;
  input  PARENT NAME;
  cards;
  Ina  Nala
  Ina  Chata
  Ina  Bruja
  Ina  Fidonga
  Chavo  Roco
  Cecilia Nina
  Cecilia Jadzia
; run;
```
SAS data step options that can be used in PROC SQL:

The following data set Program Data Vector (PDV) options can be used in the when referring to a table, (which SAS is a data set):

- `KEEP=`
- `DROP=`
- `RENAME=`
- `WHERE=`

The general structure for their use within SAS PROC SQL is:

```sas
PROC SQL SQL_OPTIONS;
  CREATE TABLE EXAMPLE (KEEP= .....)
    RENAME=( .....)
  DROP=
    WHERE=( NEWVAR1 ....)
) AS SELECT *
FROM ORIGINAL_TABLE (KEEP=VAR1 VAR2....)
  RENAME=(VAR1 = NEWVAR1
    VAR2 = NEWVAR2
    ....)
  DROP=
    WHERE=( NEWVAR1 ....)
) ;
QUIT;
```

Notice, there are no semicolons after the SAS statements, and the order of the statements is the same as in a data step or PROC statement.

The advantage of using these options increases with the number of variables and observations in a data set. In this presentation, the data sets are not big.

If you have a data set with 50 variables and you only need 45 of those variables, you could type 45 variables:

```
SELECT VAR1, VAR2..., VAR45
FROM DATA SET;
```

or you could

```
SELECT *
FROM DATA SET (DROP=VAR46  ...  VAR50) ;
```

In the second method, you only need to type 5 variables, and your program is more efficient, because you did not bring in 5 variables you did not need in your PDV.

In our example you can do:

```
select * from
  DOGS (DROP=dob
    RENAME=(NAME=FirstName))
;  
FirstName   BREED
Nala         Airdale
Nina         AMBR
Jadzia       Pointer
Rocco         MMBNR
Bruja         Rottweiler
Chacal       Rottweiler
Chata         Boxer
Tyson         Rottweiler
```
The previous example has only 3 variables, and by removing one variable from the dataset DOGS not much is gained with the use of the DROP statement. However, usually there are more variables in a dataset. Suppose that there are 20 variables in a dataset and you need 17 of those variables. Do you want to include those 3 to be able to use the "*" to refer to your variables? or do you want to type 17 variable names to be able to avoid carrying 3 extra variables? This problem is especially true when the variable names are not easy to type.

The RENAME option is useful when you want to change the name of some of the variables, but you don't want to type all of the variables. I can think of two different ways of doing this.

The first example is just a select statement, and no new data sets or variables are created, the RENAME option is used just as a label (as you would in PROC PRINT or other reporting procedures.) In the second example, a data set needs to be created to be able to drop the variable that is not needed any longer. Also, there is a need for a SELECT statement to see the new variable(s).

Advantages of using data set options when merging two data sets:

While merging two data sets, you can use the SAS options to your 'typing' advantage. If you want to create a data set from two data sets with 50 variables each and they have 3 variables in common, you want to keep all the variables, but you don't want to type 97 variables. You can not create a table select "*" from both tables, because there are 3 variables that exist in both data sets, you have two alternatives, type the 97 unique variables or use a RENAME option:

```
CREATE TABLE FINAL(DROP=VAR1B VAR2B VAR3B)
AS SELECT A.*, B.*
FROM DATA SET1 A, DATA SET2 (RENAME = (VAR1=VAR1B VAR2=VAR2B VAR3=VAR3B)) B
WHERE A.VAR1=B.VAR1B
AND   A.VAR2=B.VAR2B
AND   A.VAR3=B.VAR3B
;
```

Using the data that exists in our example:

```
proc sql;
create table new(DROP=nickname) as
select a.*, B.*
from DOGS a,  PARENTs(RENAME=(name=nickname)) b
WHERE a.name=b.nickname
;
select * from new;
quit;
```

```
name       breed        DOB        Parent
Nala       Airdale      11JAN1990  Ina
Nina       AMBR         22APR2001  Cecilia
Jadzia     Pointer      15FEB1999  Cecilia
Rocco      MMBNR        08MAY2001  Chavo
Bruja      Rottweiler   15AUG2000  Ina
Chata      Boxer                .  Ina
```

```
select * from DOGS(RENAME=(Name=FirstName));
create table change_name(DROP=name) as select *, name as FirstName from DOGS;
select * from change_name;
```
Notice that we lost two records/dog names, Tyson and Chacal did not have a record in the PARENTS data set. If we want to get the list of Rottweilers and their parent, in **SAS PROC SQL** we do not need to have both data sets sorted by Name before we put the data sets together; in fact, we can see the list without creating a data set. We need to do a **LEFT JOIN** to be able to get all the observations in DOGS.

```sas
proc sql;
  select a.*, B.*
  from DOGS a left join parents b
    on a.name=b.name
    and a.breed='Rottweiler'
    and a.breed='Rottweiler'
  ;
quit;
```

Also notice that we got all the records in DOGS, no matter what breed they are.

The way to limit the records that come into the **LEFT JOIN** is limiting the records from the beginning.

```
CREATE TABLE FINAL AS SELECT
  A.*, B.*
FROM DATA SET1 (WHERE=( VAR1='keep')) A LEFT JOIN DATA SET2 B
  ON A.VAR1=B.VAR1B
  AND A.VAR2=B.VAR2B
  AND A.VAR3=B.VAR3B
;
```

All the observations and only the observations **WHERE VAR1='keep'** will be kept in the data set **FINAL**. This last statement is going to make your program more efficient because only the observations that meet the requirement **VAR1='keep'** will be included.

To get only the records and all the records for Rottweilers, we can use a **WHERE** statement in the data set we want to limit.

```
select a.*, b.*
from DOGS
  (WHERE=(breed='Rottweiler'))a
  left join parents b
    on a.name=b.name
;
```

We get all the Rottweilers and the PARENT when there is one.

In the previous examples, we were only doing **SELECT** statements, we were not creating data sets, if we wanted to create a new data set, there would be a warning in the log, because there is a column in the data set DOGS and a column in the data set PARENTS with the same name, **NAME**.
Differences between Data Step and SAS PROC SQL

The word CASE

The word **CASE** is used in data step and in SAS PROC SQL in the same manner. The general statement for the use of the word **CASE** is:

```sql
SELECT VAR1, VAR2,
    CASE [varK] WHEN ("VALUE1") THEN ["Constant value" | function]
    WHEN ("VALUE2") THEN ["other constant" | function]
    ELSE ["last value" | function]
END AS NEWVAR
FROM ORIGINAL_TABLE
;
```

In data step, instead of using **END**, we would use **OTHERWISE**.

**Example of the use of CASE 1:**

```sql
select
    name,
    case NAME
    when ('Nala') then 'None'
    when ('Nina') then 'Novice'
    else 'There is hope'
end as education
from parent;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>education</th>
<th>parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nala</td>
<td>None</td>
<td>Ina</td>
</tr>
<tr>
<td>Chata</td>
<td>There is hope</td>
<td>Ina</td>
</tr>
<tr>
<td>Bruja</td>
<td>There is hope</td>
<td>Ina</td>
</tr>
<tr>
<td>Fidonga</td>
<td>There is hope</td>
<td>Ina</td>
</tr>
<tr>
<td>Rocco</td>
<td>There is hope</td>
<td>Chavo</td>
</tr>
<tr>
<td>Nina</td>
<td>Novice</td>
<td>Cecilia</td>
</tr>
<tr>
<td>Jadzia</td>
<td>There is hope</td>
<td>Cecilia</td>
</tr>
</tbody>
</table>

**Example of the use of CASE 2:**

```sql
select name,
    case
    when index(breed,'MB') then put(breed,$UKC.)
    else breed
end as lbreed
label "Long Breed"
from dogs;
```

<table>
<thead>
<tr>
<th>name</th>
<th>Long Breed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nala</td>
<td>Airdale</td>
</tr>
<tr>
<td>Nina</td>
<td>American Mixed Breed Registry</td>
</tr>
<tr>
<td>Jadzia</td>
<td>Pointer</td>
</tr>
<tr>
<td>Rocco</td>
<td>Mexican Mixed Breed Non Registered</td>
</tr>
<tr>
<td>Bruja</td>
<td>Rottweiler</td>
</tr>
<tr>
<td>Chacal</td>
<td>Rottweiler</td>
</tr>
<tr>
<td>Chata</td>
<td>Boxer</td>
</tr>
<tr>
<td>Tyson</td>
<td>Rottweiler</td>
</tr>
</tbody>
</table>
In **SAS PROC SQL**, the word **CASE** is a reserved word. A variable named CASE can exist in but it cannot be used while inside it. The example below tries to give an example of this paragraph.

```sas
data new;
    length case $20;
    set parents;
        if parent = 'Ina' then case = 'None';
        else if parent = 'Chavo' then case = 'Spoiled';
        else case = 'Work';
    run;

proc print data=new;
run;

proc sql;
    select * from new WHERE parent='Ina';
    select * from new WHERE case='None';
quit;
```

**Output from Proc Print:**

<table>
<thead>
<tr>
<th>Obs</th>
<th>case</th>
<th>parent</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>None</td>
<td>Ina</td>
<td>Nala</td>
</tr>
<tr>
<td>2</td>
<td>None</td>
<td>Ina</td>
<td>Chata</td>
</tr>
<tr>
<td>3</td>
<td>None</td>
<td>Ina</td>
<td>Bruja</td>
</tr>
<tr>
<td>4</td>
<td>None</td>
<td>Ina</td>
<td>Fidonga</td>
</tr>
<tr>
<td>5</td>
<td>Spoiled</td>
<td>Chavo</td>
<td>Rocco</td>
</tr>
<tr>
<td>6</td>
<td>Work</td>
<td>Cecilia</td>
<td>Nina</td>
</tr>
<tr>
<td>7</td>
<td>Work</td>
<td>Cecilia</td>
<td>Jadzia</td>
</tr>
</tbody>
</table>

**Output from first select statement:**

<table>
<thead>
<tr>
<th>case</th>
<th>parent</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Ina</td>
<td>Nala</td>
</tr>
<tr>
<td>None</td>
<td>Ina</td>
<td>Chata</td>
</tr>
<tr>
<td>None</td>
<td>Ina</td>
<td>Bruja</td>
</tr>
<tr>
<td>None</td>
<td>Ina</td>
<td>Fidonga</td>
</tr>
</tbody>
</table>

**Output from Log from the second select statement**

```sas
select * from new WHERE case='None';
+ 22
+ 76
+ERROR 22-322: Syntax error, expecting one of the following:
+ a name, a quoted string, a numeric constant,
+ a datetime constant, a missing value, (, +, -, 
+ BTRIM, CALCULATED, CASE, EXISTS, INPUT, LOWER,
+ NOT, PUT, SELECT, SUBSTRING, TRANSLATE, UPPER,
+ USER, WHEN, ^, ~.
+ERROR 76-322: Syntax error, statement will be ignored.
```

To avoid this problem, you can try to avoid the calling your variables CASE, or if you have no options, **rename** the variable:

```sas
select * from new(rename=(case=education)) WHERE education='None';
```
Equivalences in format modifiers between data step and SQL;

The format modification is prevalent for programmers to use question marks when creating a formatted variable to avoid errors in the log from invalid values. I don't like this practice, but a way to prevent errors in the log while using SAS PROC SQL is:

```
select name, dob, input(dob, ? date9.) as idate format mmddyy8. from DOGS;
```

Produces a clean log versus

```
select name, dob, input(dob, date9.) as idate format mmddyy8. from DOGS;
```

Produces this line in the log

```
+ERROR: Invalid day value
```

The order in the execution in SAS PROC SQL

In SQL, the order of the statements is important; the order of the statements is the order the requirements are executed. A general statement for SQL IS:

```
SELECT VAR1, VAR2, ...
FROM ORIGINAL_TABLE  [ FULL | LEFT | RIGHT | INNER ] JOIN TABLE2
ON   condition 1
AND  condition 2
WHERE   condition k
AND  condition 1
GROUP BY VAR1, VAR...
HAVING  condition p
AND  condition q
ORDER BY var1, var2...
;
```

A FULL, LEFT, RIGHT, or INNER JOIN is executed before a WHERE statement.

ON is executed before a WHERE statement, and a GROUP is executed before a HAVING statement and an ORDER is always the last statement to be done.

```
select a.breed,
       input(a.dob,? date9.) as ndob format date9.,
       coalesce(a.name, b.name) as name,
       b.parent
from DOGS a full join parents b
on a.name=b.name
;
All the records in DOGS were brought in.

<table>
<thead>
<tr>
<th>breed</th>
<th>ndob</th>
<th>name</th>
<th>parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rottweiler</td>
<td>15AUG2000</td>
<td>Bruja</td>
<td>Ina</td>
</tr>
<tr>
<td>Rottweiler</td>
<td>29JAN1990</td>
<td>Chacal</td>
<td></td>
</tr>
<tr>
<td>Boxer</td>
<td>7JUN1990</td>
<td>Chata</td>
<td>Ina</td>
</tr>
<tr>
<td>Pointer</td>
<td>15FEB1999</td>
<td>Jadzia</td>
<td>Cecilia</td>
</tr>
<tr>
<td>Airdale</td>
<td>11JAN1990</td>
<td>Nala</td>
<td>Ina</td>
</tr>
<tr>
<td>AMBR</td>
<td>22APR1997</td>
<td>Nina</td>
<td>Cecilia</td>
</tr>
<tr>
<td>MMBNR</td>
<td>08MAY1980</td>
<td>Rocco</td>
<td>Chavo</td>
</tr>
</tbody>
</table>

Only Rottweilers were brought in, because the join/merge was done before the WHERE statement and after the join, the where eliminated the records where breed is not Rottweiler.

But if we inverse the conditions for WHERE and ON we would get a different output.

<table>
<thead>
<tr>
<th>breed</th>
<th>ndob</th>
<th>name</th>
<th>parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rottweiler</td>
<td>15AUG2000</td>
<td>Bruja</td>
<td>Ina</td>
</tr>
<tr>
<td>Rottweiler</td>
<td>29JAN1990</td>
<td>Chacal</td>
<td></td>
</tr>
<tr>
<td>Rider</td>
<td>7JUN1990</td>
<td>Chata</td>
<td>Ina</td>
</tr>
<tr>
<td>Pointer</td>
<td>15FEB1999</td>
<td>Jadzia</td>
<td>Cecilia</td>
</tr>
<tr>
<td>Airdale</td>
<td>11JAN1990</td>
<td>Nala</td>
<td>Ina</td>
</tr>
<tr>
<td>AMBR</td>
<td>22APR1997</td>
<td>Nina</td>
<td>Cecilia</td>
</tr>
<tr>
<td>MMBNR</td>
<td>08MAY1980</td>
<td>Rocco</td>
<td>Chavo</td>
</tr>
<tr>
<td>Rottweiler</td>
<td>15MAR2000</td>
<td>Tyson</td>
<td></td>
</tr>
</tbody>
</table>
Care needs to be put in the order of the statements. The results of the query vary with the order of the statements.

But if we want to limit the records from DOGS to the value of a variable that we are creating, we need to use the option HAVING. The variable NDOB is being created in the select statement and we can not use it before a WHERE statement.

```sql
select a.breed,
       input(a.dob, ? date9.) as ndob format date9.,
       coalesce(a.name, b.name) as name,
       b.parent
from DOGS a full join parents b
on a.name=b.name
WHERE breed='Rottweiler'
having ndob gt '01jan2000'd;
```

<table>
<thead>
<tr>
<th>breed</th>
<th>ndob</th>
<th>name</th>
<th>parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rottweiler</td>
<td>15AUG2000</td>
<td>Bruja</td>
<td>Ina</td>
</tr>
<tr>
<td>Rottweiler</td>
<td>15MAR2000</td>
<td>Tyson</td>
<td></td>
</tr>
</tbody>
</table>

Conclusions

SQL is a very powerful language. SAS PROC SQL has additional advantages because SAS PROC SQL tables are data sets, they can take advantage of the options of the PDV. This paper showed examples on how to use the KEEP, RENAME, DROP, and WHERE data set options are commonly used in the data step options and other procedures in SAS PROC SQL. These options are not ANSI, so can not be used in other SQL environments.

You can have plain vanilla SAS PROC SQL or you can SASsy it up with DPV options.

Limitations

This paper did not cover the difference in use in SQL ANSI functions and SAS PROC SQL functions, like MEAN, COUNT, MIN, MAX. It did not cover options, like flow, number that can be invoked with SAS PROC SQL.

Acknowledgements

I want to thank PPD management for continuing to support me in participating in PharmaSUG and to continue to present papers, especially Kim Sturgen and Bonnie Duncan. I want to thank specially Andy Barnett for reviewing my paper many times I also want to thank Neil Howard, Larry Sleeper and Craig Mauldin for their time, although if there are mistakes, they are all mine. I made modifications and additions after they read the paper.

Contact Information.

Angelina Cecilia Mauldin
PPD Inc.
3900 Paramount Parkway
Morrisville, NC 27560
(919) 462-4199
Cecilia.Mauldin@rtp.ppdi.com
www.ppdi.com