ABSTRACT
There is more than one way of creating a dataset, looking at the data and doing basic statistics - sometimes SQL can be more useful. This paper presents different ways of manipulating data, one of which is with the use of SQL.

INTRODUCTION
One of the major challenges, concerns, and issues that the pharmaceutical/clinical trial industry encounter when conducting a clinical trial are timelines. As programmers in the clinical trial industry we are often faced with producing multiple SAS® programs code at record speed and often time program inefficiencies are produced in our code. Often time, we are required to handle data to support thousands of patients which if not programmed efficiently, can cost a lot of time when processing large amounts of data. The Structured Query Language (SQL) is one programming tool that can be used to save time and increase programming efficiencies.

SQL is a very powerful tool that was initially designed as a query tool for relational databases, but is now being used in many different software products. It is a commonly used language for retrieving and manipulating data in tables and views of those tables.

SQL was developed by IBM in the 1970’s and included in SAS® Release 6.06 in 1989, the SQL procedure. PROC SQL is part of the BASE SAS® software and allows you to use SQL within the SAS® system. PROC SQL has many capabilities, benefits, and advantages within the SAS® system. It can reduce the amount of processing time and amount of code, in some cases, when compared to non-SQL base SAS® code, data step and proc steps.

Listed below are some of the capabilities of PROC SQL within the SAS® system, which have many similarities and functionalities as the traditional SAS® data step. The objective of this paper is to show ways that PROC SQL can be used in replace of data step and procedures which can ultimately reduce processing time and code.

- SQL and SAS® terminology
- SQL capabilities
  - Retrieve data
  - Retrieve ordered data
  - Create permanent and temporary datasets
  - Create new variables
  - Create macro variables
  - Subset data
  - Proc Print or Select
  - Proc SQL compared to Proc Univariate
  - Proc SQL compared to Proc Freq
  - Coalesce function
  - Advantages of SQL join when merging datasets
- Additional topics

SQL AND SAS® TERMINOLOGY
Displayed below is a list of terms used in both SAS® and SQL. When processing and storing data a file in SAS® is referred to as a dataset, whereas in SQL it is referred to as a table. Records within SAS® are variables in a dataset and in SQL it is referred to as a column. Fields in a SAS® dataset are variables and columns in SQL.
SQL CAPABILITIES
The following DEMOG and TREAT datasets will be referenced throughout the examples in this paper. Both datasets consists of ten observations.

<table>
<thead>
<tr>
<th>PTNO</th>
<th>SEX</th>
<th>RACE</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F</td>
<td></td>
<td>21</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>B</td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>M</td>
<td>C</td>
<td>33</td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td>C</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>M</td>
<td>C</td>
<td>45</td>
</tr>
<tr>
<td>6</td>
<td>M</td>
<td>C</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>M</td>
<td>B</td>
<td>33</td>
</tr>
<tr>
<td>8</td>
<td>F</td>
<td>B</td>
<td>38</td>
</tr>
<tr>
<td>9</td>
<td>M</td>
<td>H</td>
<td>40</td>
</tr>
<tr>
<td>10</td>
<td>M</td>
<td>B</td>
<td>41</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PTNO</th>
<th>TREAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
</tr>
<tr>
<td>7</td>
<td>B</td>
</tr>
<tr>
<td>8</td>
<td>A</td>
</tr>
<tr>
<td>9</td>
<td>B</td>
</tr>
<tr>
<td>10</td>
<td>B</td>
</tr>
</tbody>
</table>

GENERAL STRUCTURE
SQL is widely used to retrieve and manipulate data. When selecting specific variables or table names variables should always be separated by a comma, however, SAS® data step does not require the variables to be separated by a comma. An asterisk “*” can be used in replace of listing variables to indicated to select all variables currently in the SAS® dataset. A semicolon should be placed at the end of the last clause in the SQL step.

RETRIEVE DATA
The general code structure of a PROC SQL step to retrieve data is displayed below. Please note, that it is not necessary to repeat the PROC SQL statement with every query. The QUIT statement is used to terminate the SQL procedure.

**PROC SQL**

```
options;
SELECT var1, var2, ..., varn
FROM TABLE
WHERE EXPRESSION
ORDER BY var1, var2, ..., varn;
QUIT;
```

Options statement is optional and can be used to control processing.

Where statement is optional and is necessary if you wish to subset the data based on a certain conditions
RETRIEVE ORDERED DATA
Data can be retrieved in an order different than what the dataset is sorted. The ORDER statement is used to retrieve the data in either ascending or descending order and must appear at the end of the query. The general structure follows:

PROC SQL:
CREATE TABLE X AS
SELECT PTNO, SEX, RACE
FROM DEMOG
ORDER BY SEX;
QUIT;

CREATE PERMANENT AND TEMPORARY DATASETS
The general code structure of a PROC SQL step to create a table is listed below and similar to the code above to retrieve data, with the inclusion of the CREATE TABLE statement.

PROC SQL options;
CREATE TABLE tablename AS
SELECT var1, var2, ..., varn
FROM TABLE
WHERE EXPRESSION
ORDER BY COLUMNS;
QUIT;

CREATING NEW VARIABLES
New variables can easily be created using SQL. Many of the functions that can be used in the data step can be used. The following code creates a new variable new_ptno which contains the character version of patient number, ptno. Please note that label, length, and format statements specified below are optional and not required to create a new variable.

SELECT PTNO, PUT(PTNO, 2.) LABEL 'NEW PTNO' FORMAT $2. LENGTH 5 AS NEW_PTNO
FROM DEMOG;

CREATE MACRO VARIABLES
PROC SQL can create and/or update macro variables using the INTO keyword and can be accessed, once created, in other sections of the program. The following syntax is the general structure for creating a macro variable using SQL:

SELECT COUNT(DISTINCT PTNO)
INTO: GROUP1::GROUP2
FROM DEMOG
GROUP BY SEX;
QUIT;

Based on the data above from the DEMOG dataset, macro variables GROUP1 and GROUP2 are created with values of 4 and 6, respectively.

SUBSET DATA
Below is SQL and equivalent print procedure code to subset data based on certain conditions. For this example, we are going to subset the DEMOG data set to obtain all patients who are female, SEX='F' over the age of 25.
Both the index and like functions both search the character string, sex, for any occurrence of the letter F. The index function will return the location of the specified character string. A value of 0 indicates that the string was not found. Be careful when using quotes, the following warning will be displayed in the log:

**WARNING: Apparent invocation of macro F not resolved.**

**PROC PRINT OR SELECT?**

Often times we need to view subsets that have values that contain data within a variable whose length is very large. If using PROC PRINT, the following message will be displayed in the SAS® log:

**WARNING: Data too long for column "varname"; truncated to 122 characters to fit.**

To avoid warnings such as the one listed above; PROC SQL can be used with the FLOW option which will allow you to control the appearance of wide columns. It allows the text to be flowed in its column instead of wrapping the text to the next row.

**PROC SQL COMPARED TO PROC UNIVARIATE**

Simple statistics can be used within the SQL procedure, such as mean, count, standard deviation, maximum and minimum, etc... It produces the same results as the PROC UNIVARIATE procedure. Quite often we are faced with large datasets with thousands of records to compute summary statistics on. To reduce the amount of processing time, SQL can be used to reduce that time when summarizing data with by groups. For example, if obtaining the above statistics on the variable age categorized by sex, the data would have to be sorted by the sex variable if using PROC UNIVARIATE. If using SQL, the sort step can be eliminated. See example below:

**SQL:**

```sql
PROC SQL;
CREATE TABLE X AS
SELECT SEX, MEAN(AGE) AS MEAN, COUNT(*) AS COUNT, STD(AGE) AS STD, MAX(AGE) AS MAX, MIN(AGE) AS MIN
FROM DEMOG GROUP BY SEX;
QUIT;
```

**PROC UNIVARIATE:**

```sas
PROC UNIVARIATE DATA=DEMOG;
BY SEX;
VAR AGE;
OUTPUT OUT=STATS MEAN=MEAN N=COUNT STD=STD MAX=MAX MIN=MIN;
RUN;
```
PROC SQL COMPARED TO PROC FREQ
The DISTINCT clause is used to get the occurrences of data within the data set. The frequency procedure can also be used, however, when handling large data sets SQL is reduces processing time. As indicated above, if grouping data by certain variables, the data do not have to be pre-sorted.

<table>
<thead>
<tr>
<th>SQL:</th>
<th>PROC FREQ:</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE TABLE X AS SELECT DISTINCT SEX FROM DEMOG; QUIT;</td>
<td>PROC FREQ DATA=DEMOG; TABLES SEX; RUN;</td>
</tr>
</tbody>
</table>

COALESCE FUNCTION
The coalesce function is a SQL specific function which returns the first value that is non-missing value among its arguments. Please note that all arguments in the list must have the same type. It reduces the amount of code by eliminating the use of several if statements within the data step or in some cases when using SQL. It will allow you to replace missing values with a non-missing value. The following example replaces missing values of race with UNK for unknown.

SELECT PTNO, COALESCE(RACE, 'UNK') AS RACE FROM DEMOG; QUIT;

ADVANTAGES OF SQL JOIN WHEN MERGING DATASETS
Many times we are required to combine two or more datasets. In the data step, the merge statement is used and it requires the data to be sorted. In SQL this task can be done very simply by using the JOIN clause to combine datasets and is often used in the SQL procedure. It does not require the data to be sorted and the variables do not have to have the same name in the join clause. The following example gives an overview of how two data sets can be combined to obtain the treatment.

PROC SQL:
CREATE TABLE X AS SELECT DISTINCT  A.PTNO, A.AGE, A.SEX, A.RACE B.PTNO, B.TREAT FROM DEMOG A, TREAT B WHERE A.PTNO=B.PTNO; QUIT;

ADDITIONAL TOPICS
Additional topics not covered in this paper, but not limited to are listed below:
- Merging: Left join, full join
- Functions that is specific to SQL, monotonic, etc..
- Functions that are unknown to SQL: FIRST., LAST., and LAG
- Case statements such as if and else statements
- Dictionary Tables
- Different options in SQL, such as noprint, obs, etc..

CONCLUSION
Clearly there are more than one alternative when accessing, creating, and manipulating data within the SAS® system. This paper’s intent was to introduce some of the basic concepts of PROC SQL, a powerful language in the BASE SAS® software, and how it can reduce program code and processing time when handling large data sets when compared to some of the non-SQL techniques in SAS®, such as data step and proc steps. Although this paper only focuses on some of the basics of using PROC SQL, the language has many more capabilities outside the scope of this paper. By now, you should know some of the similarities and differences between PROC SQL and Data Step programming and be able to create some basic PROC SQL code. Ultimately, it is up to the programmer to decide which technique to use and make the best decision for whatever programming task you are required to perform to maintain high quality and efficient programming code.
REFERENCES
PROC SQL Beyond the Basics Using SAS®, Kirk Paul Lafler
SQL Processing with the SAS® System, SAS® Institute
SAS® Programming Course Notes, SAS® Institute
Top Ten Reasons to use PROC SQL, Weiming Hu, SUGI Paper 29
Tips for Using SQL: When to Use and How?, Ying Feng, NESUG Paper 18

ACKNOWLEDGMENTS
I would like to thank Kim Sturgen, Bonnie Duncan, and Cecilia Mauldin for their encouragement, review, and comments.

CONTACT INFORMATION
Your comments and questions are valued and encouraged. Contact the author at:

LaTonya Murphy
PPD Inc.
3900 Paramount Parkway
Morrisville, NC  27560
Work Phone:  (919) 462-4146
Fax: (919) 654-0427
E-mail: Latonya.Murphy@rtp.ppdi.com

Lan Tran
PPD Inc.
3900 Paramount Parkway
Morrisville, NC  27560
Work Phone:  (919) 462-5663
Fax: (919) 462-4128
E-mail: Lan.Tran@rtp.ppdi.com

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