ABSTRACT
This paper describes various techniques using PROC SQL to improve efficiency, performance and clarity of SAS programs. Programmers working on post marketing and sales data are often presented with the challenge of summarizing datasets containing thousands of variables and millions of records. This can be accomplished by using arrays in conjunction with OF operator on some functions like SUM and NMISS in a DATA step. However, arrays and OF operators are not supported in select statement of PROC SQL which means all the variables in the dataset has to be listed in the summarizing function which is tedious and impractical when thousands of variables are involved. This paper starts with description of a technique which makes use of efficiency of PROC SQL without having to list all the variables. Then, a technique to convert correlated sub queries using PROC SQL into a left join and non correlated sub query using PROC SQL is presented. This improves clarity of programming and makes it easier to understand for beginners without compromising on performance. Some more techniques are discussed including the one which reproduces the results of PROC SORT with NODUP-KEY/NODUPLCATES options using PROC SQL with DISTINCT or UNIQUE and other undocumented PROC SQL options. All the techniques described are illustrated with different real time scenarios.

INTRODUCTION
Most of the large corporations use Relational Database Management Systems (RDBMS) like Oracle, MS SQL Server, Access, DB2 and etc. as data repository. Most of the database applications support Structured Query Language (SQL) for information retrieval and storage. Over the years, SQL has evolved as a standard for data retrieval, storage and manipulation. 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).

DBMS programs run on various platforms and different operating systems. SQL is a set of easily understandable statements and clauses. The query is written in English like syntax, making it easy to comprehend and is platform independent. Even though SQL is a standard, its implementation is different in different database packages. PROC SQL is the SAS implementation of SQL and can be used in place of traditional data steps. The goal of this paper is to demonstrate the use of PROC SQL procedure in place of traditional data steps to enhance efficiency and simplicity.

ILLUSTRATIONS
This section presents some of the real time scenarios and illustrates the use of PROC SQL.

SCENARIO 1: FUNCTION (OF…) USING PROC SQL
Working with post marketing data in pharmaceutical industry or retail industry which involves millions of observations and thousands of variables is always challenging. To help perform various operations especially involving large number of variables, OF operator can be used with some of the functions like SUM and NMISS in a DATA step. The advantage of this method is that it makes the syntax much less laborious to type, since it avoids listing of variables individually. However, OF operator cannot be used with PROC SQL, as SQL Engine does not support this feature. Many a times, use of PROC SQL cannot be avoided especially if operations like many to many matched merging needs to be performed. The same can be achieved by using macro variables with PROC SQL. This is illustrated here. The technique presented below can be applied whenever number of variables with a prefix is involved. PROC SQL does not support arrays and the following technique can be used to overcome this shortcoming.

Data Preparation:
SAS code for simulating a dataset with large number of variables is presented in Figure 1. This code snippet first creates dataset x with one variable "id". Then it creates 24 variables with names prefixed by "nrx" and 24 variables with names prefixed by "trx". Resultant dataset will have 10 observations and 49 variables: nrx1 - nrx24, trx1 - trx24 and id.
Figure 2 presents the code using DATA step and Figure 3 presents the PROC SQL implementation of the same.

**Figure 1: Data preparation for Scenario 1**

*Create dataset x with id variable*

```plaintext
data x;
  input id @@;
cards;
  1 2 3 4 5 6 7 8 9 10;
run;
```

*Create a dataset containing 24 variables whose names are prefixed by nrx and 24 variables whose names are prefixed by trx.*

```plaintext
data x1;
  set x;
  array nrx(24) nrx1 - nrx24;
  array trx(24) trx1 - trx24;
  do i = 1 to 24;
    nrx(i) = 1;
    trx(i) = 10;
  end;
drop i;
run;
```

**Figure 2: Data step implementation for Scenario 1**

```plaintext
data z1;
  set x1;
  sumtrx = sum(of trx1-trx24, 0);
  sumnrx = sum(of nrx:, 0);
run;
```

**Figure 3: PROC SQL implementation for Scenario 1**

*Store the names of variables prefixed by "nrx" in dataset names1;*

```plaintext
proc contents data = x1;
  out = names1(keep= name
  where=(name like nrx%)); run;
```

*Assign all variable names of names1 to a macro variable separated by comma;*

```plaintext
proc sql;
  select name into : name1 separated by ,
  from names1;
quit;
```

*Store the names of variables prefixed by "trx" in dataset names2;*

```plaintext
proc contents data = x1;
  out = names2(keep= name
  where=(name like 'trx%')); run;
```

*Assign all variable names of names2 to a macro variable separated by comma;*

```plaintext
proc sql;
  select name into : name2 separated by ,
  from names2;
quit;
```

*Use the macro variables with all the variable names with sum function;*

```plaintext
proc sql;
  create table z2 as
  select * , sum(&name1.) as sumnrx ,
  sum(&name2.) as sumtrx
  from x1;
quit;
```

**SCENARIO 2: PROC SORT NODUPKEY USING PROC SQL**

PROC SORT with NODUPKEY option eliminates all the duplicate combinations of BY variables. The output of "DISTINCT *" used in SELECT statement is analogous to options NODUPLICATES or NODUPREC of PROC SORT. The results obtained with PROC SORT with NODUPKEY option can also be achieved using PROC SQL. This method is useful during some scenarios when PROC SQL needs to be used. In that case it makes more sense to implement all the steps using PROC SQL rather than introducing another PROC SORT step to remove duplicate combinations of BY variable. Two methods are presented for this. Method 1 (Figure 5) is more efficient than method 2 (Figure 6). Figure 4 shows the data preparation. The function monotonic() produces the observation number for each of the datasets.
SCENARIO 3: CORRELATED AND NON-CORRELATED SUBQUERIES

Suppose dataset x has observations corresponding to all subjects in a study and dataset y has information of subjects who experienced adverse events during the study. If one needs to get all the subjects who did not experience any adverse event, then the dataset x needs to be searched. In a correlated subquery (Figure 8), value of correlated reference (x.a in this case) for observation in dataset x is passed into the subquery "SELECT * FROM Y WHERE y.a = x.a". Since the condition is "NOT EXISTS", the observation is written to the output dataset only if there are no matching values for x.a in dataset y. This process is repeated for every observation of x (i.e. 10 times in this case). Correlated subqueries are difficult to comprehend and sometimes inefficient. A method is presented in Figure 9 in which the correlated subquery presented in Figure 8 is divided into left join and non correlated subquery. This makes the code readable and easier to comprehend.

```plaintext
Figure 4: Data preparation for Scenario 2:
*Create a simple test dataset t1 with five variables;
data t1;
  input a b c d e;
cards;
  1 2 3 4 5
  1 2 3 5 4
  2 2 2 4 5
  2 2 2 5 4
  3 3 3 4 5
  3 3 3 5 4;
run;

Figure 5: Implementation of Scenario2 using Method1
proc sql;
create table final(drop = num1) as
  select t1.*, monotonic() as num1
  from t1
  group by a,b,c
  having min(num1) = num1;
quit;

Figure 6: Implementation of Scenario2 using Method2
*Create variable num1 whose value is same as the observation number;
proc sql;
create table fin1 as
  select t1.*, monotonic() as num1
  from t1;
quit;
*Create variable num2 which is the minimum of num1 grouped by a,b,c;
proc sql;
create table fin0 as
  select a,b,c,min(num1) as num2
  from fin1
  group by a,b,c;
quit;
*Merge fin1 and fin0 by using num1,num2 as respective keys;
proc sql;
create table final1(drop = num1) as
  select fin1.*
  from fin1 f, fin0 p
  where f.a=p.a and f.b=p.b and f.c=p.c and f.num1=p.num2;
quit;

Figure 7: Data preparation for Scenario 3:
data x;
  input a b c @@;
cards;
  1 1 1 1 1 2 2 2 2 2 3 3 3 3
  3 3 4 4 4 5 5 5 6 6 6 7 7 7;
run;
data y;
  input a x y @@;
cards;
  1 1 1 2 2 3 3 3
run;

Figure 8: Merging using Correlated subquery
proc sql;
create table u1 as
  select * from x
  where not exists ( select * from y
    where y.a = x.a ) ;
quit;
```
CONCLUSION
This paper presents PROC SQL alternatives to traditional data step programming techniques for some real time scenarios. This makes the code readable and easily comprehendible to programmers coming from different backgrounds.

REFERENCES:

ACKNOWLEDGMENTS
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 registered trademarks or trademarks of their respective companies.

We would like to thank management of MaxisIT Inc. for their kind support and cooperation.

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

Author Name: Sreekanth Middela  Author Name: Suhas R. Sanjee
Company     : MaxisIT Inc.                                    Company     : MaxisIT Inc.
Address      : 1199 Amboy Ave                                   Address      : 1199 Amboy Ave
City: Edison  State: NJ ZIP: 08830                               City: Edison  State: NJ ZIP: 08830
Email:sreekanth@maxisit.com                                Email: sanjee@maxisit.com

Figure 9: Merging using Non-Correlated subquery

```sql
proc sql;
create table u2 as
  select x.* from x
  left join y
  on x.a = y.a;

create table u3 as
  select x.a from x , y
  where x.a = y.a;

create table u4 as
  select * from u2
  where a not in
  ( select distinct a from u3 )
quit;
```