Paper 59

How and When to Use WHERE

J. Meimei Ma, Quintiles, Research Triangle Park, NC
Sandra Schlotzhauer, Schlotzhauer Consulting, Chapel Hill, NC

INTRODUCTION

This tutorial explores the various types of WHERE as methods for creating or operating on subsets. The discussion covers DATA steps, Procedures, and basic use of WHERE clauses in the SQL procedure. Topics range from basic syntax to efficiencies when accessing large indexed data sets. The intent is to start answering the questions:

- What is WHERE?
- How do you use WHERE?
- Why is WHERE sometimes more efficient?
- When is WHERE appropriate?

The primary audience for this presentation includes programmers, statisticians, data managers, and other people who often need to process subsets. Only basic knowledge of the SAS system is assumed, in particular no experience beyond Base SAS is expected. However, people with more experience may still discover new reasons for using WHERE as well as potential pitfalls. The simple examples provided to show syntax and potential complications apply to all operating systems running Release 6.08 or later.

TERMINOLOGY

Efficiency Elements

There are two primary categories of elements to consider when evaluating efficiency: machine and human. Machine efficiency elements include computer processing time, often called CPU, and processing time for reading or writing computer data, called I/O for Input/Output. Efficiency for humans is measured by considering programmer time, level of expertise required, or clarity of final code. Major components of programmer time include planning programming strategy, writing new code, testing programs, running production programs, and revising existing code.

Always consider both machine and human elements when choosing between options for efficiency reasons. The choice may be difficult, or at least ambiguous, since the more machine efficient option can require additional human effort or vice-versa.

Large File Environment

In large file environments choosing an efficient programming strategy tends to be important. A large file can be defined as a file for which processing all records is a significant event. This may apply to files that are short and wide, with relatively few observations but a large number of variables, or long and narrow, with only a few variables for many observations. The exact size that qualifies as large depends on the computing environment. In a mainframe environment a file may need to contain millions of records before being considered large. For a microcomputer, the threshold will always be lower even as processing power increases on the desktop. Batch processing is used more frequently for large file processing. Data warehouses typically involve very large files.

Types of WHERE

Several types of “WHERE” exist in Release 6.08 or later of SAS software. The most common are:

- WHERE statement (DATA step or Procedure)
- WHERE data set option
- WHERE clause in PROC SQL

The original source of WHERE probably stems from SQL (Structured Query Language). Other areas that support WHERE commands or clauses include SAS/FSP, SAS/ASSIST, SAS/CONNECT. Issues related to these products are not addressed in this paper.

THE WHERE STATEMENT

Syntax

The WHERE statement is used for selecting observations from a SAS data set by specifying a simple or complex conditional clause. In addition to standard comparison and logical operators such as EQ or AND, special operators are available. In general, SAS functions can be included. The syntax is identical whether the statement is used in a DATA step or a Procedure:

```
WHERE where-expression ;
```

For example, to create a subset of a permanent SAS data set that includes all observations with a certain value of a categorical variable:

```
DATA subset;
  SET libref.indata;

    WHERE catvar = value;

    /* other SAS statements */

    OUTPUT;      /* OPTIONAL */
    RETURN;      /* OPTIONAL */
RUN;    /* OPTIONAL */
```

If no DATA step is required, then you should simply process a subset directly, as in the following:

```
PROC FREQ  DATA=libref.indata;
  WHERE catvar = value;
  TABLES var1 * var2;
RUN;
```

Special Operators

Several special operators are available for where-expressions. The examples here provide a taste of the possibilities.

```
/* Select drug names starting with A */
  WHERE drugname CONTAINS 'A';
```

"Asprin" and "Advil" would be selected, but so would "VOLMAX." However, the WHERE statement would not select "Zithromax" because the CONTAINS operator is case-sensitive.

```
/* Select for missing values */
  WHERE treatmnt IS MISSING;
  WHERE treatmnt IS NULL;
```

All missing values for the variable TREATMNT would be selected. The advantage of using either IS MISSING or IS NULL is that you do not need to know whether a variable is numeric or character.

```
/* Select states with capital C in name */
   WHERE state LIKE '%C%' ;
```

The states "North Carolina", "South Carolina", "Connecticut", "California" and "DC" would be found. However, the WHERE statement would not select "Kentucky", "Wisconsin", and so on because the LIKE operator is case-sensitive. The % sign substitutes for any combination of characters, and as in the example above, can be used more than once in an expression.

```
/* Select body system starting with Gastro */
  WHERE bodysys LIKE 'Gastro%' ;
```

"Gastrointestinal" and "GastroIntestinal" would be selected but not "GASTROINTESTINAL" because LIKE is case-sensitive.

```
/* Select values not equal to 65  */
  WHERE spdlimit <> 65 ;
  WHERE spdlimit NE 65;
```

In this case,the two WHERE statements are equivalent. Version 6 documentation incorrectly states that the <> operator is interpreted as “maximum” when it is actually interpreted as “not equals.”

```
/* Select age >= 18 AND age <= 34 */
  WHERE age BETWEEN 18 AND 34 ;
```

The BETWEEN operator can be easier for a programmer to understand. In ACCESS view descriptors, using BETWEEN is more machine efficient than the traditional operators.

Execution Logic of WHERE Statement

In a DATA step, the WHERE statement is not considered a standard executable statement. Observations are selected before data values are moved into the program data vector (PDV). The selection occurs immediately after input data set options are applied and before other DATA step statements are executed. As a result, certain automatic variables cannot be used in where-expressions and any variables created in the DATA step cannot be included. Other implications of WHERE processing logic become clearer when compared with Subsetting IF logic.

The WHERE statement can only be used in DATA steps that use existing SAS data set(s) as input, i.e., a SET, MERGE, or UPDATE statement must exist. If you are using an INPUT statement to read in “raw” files, then you cannot use WHERE. A single WHERE statement can apply to multiple data sets. Of course, the assumption is that all the data sets include the variables referenced in the where-expression. In a complex DATA step with more than one SET, MERGE, or UPDATE statement, you should combine the selection criteria into one WHERE statement. If more than one WHERE exists

then only the last statement is applied and no error message appears.

Using WHERE is incompatible with the OBS= data set option and the POINT= option of the SET statement. These statements select observations by observation number. Also, FIRSTOBS= may not have a value other than one. For instance, you cannot use a testing strategy based on using OBS=100 when your program includes WHERE.

If you use the MODIFY statement with BY processing, then dynamic WHERE processing occurs. There may be an efficiency concern for large files that are not sorted and have not been indexed as a result.

Comparison to Subsetting IF

The "traditional" method of creating subsets in a DATA step relies on the subsetting IF statement (SubIF). In many cases, you can produce the same subset using either WHERE or SubIF. However, situations exist in which different results will be obtained. The values of FIRST. and LAST. may be different since WHERE selections are made before BY groups are defined. With a MERGE or UPDATE based on a BY statement, WHERE selects observations before the merge while SubIF selects after the merge (see Example 2).

Advantages of WHERE Statement

- Uses index if appropriate
- Efficient (before transfer to PDV)
- Special operators available
- Can apply directly to Procedures

Advantages of Subsetting IF

- Executable conditionally (IF-THEN)
- Any input file type
- Use any automatic variables
- Compatible with OBS=, POINT=, FIRSTOBS=
- Same in all versions/releases

Because of the overhead required for WHERE processing, using SubIF can be more machine efficient when a data set is relatively narrow (short record lengths). In any case, SubIF should appear as close to the beginning of a DATA step to avoid unnecessary processing of records that will ultimately be deleted.

THE WHERE DATA SET OPTION

Syntax

The syntax of the WHERE data set option, called WHERE=, is a combination of standard data set option parentheses and a where-expression. The selection criteria only apply to the last data set if more than one is listed in the same statement. The syntax is the same for either DATA steps or Procedures:

```
datasetname(WHERE=(where-expression));
```

For example:

```
DATA subset;
  MERGE   libref.indata(WHERE=(age > 35))
          libref.trtment(WHERE=(trt='T')) ;
    BY idvar;
     /* SAS statements */
RUN;

PROC FREQ
  DATA=libref.indata(WHERE=(catvar=value));
    TABLES var1 * var2;
RUN;
```

Comparison to WHERE Statement

The WHERE data set option has the same incompatibilities as the WHERE statement with the OBS= data set option and the POINT= option of the SET statement. The limitation that FIRSTOBS=1 must be true also applies.

If both a WHERE statement and WHERE= are used together in the same DATA step or Procedure, then the data set option takes precedence. In this situation, the WHERE statement is ignored.

In many situations, the differences in using a WHERE statement or the WHERE= data set option are minimal. However, known issues in some Version 6 releases are:

- With the CNTLIN option in PROC FORMAT, you need to use WHERE=.
- PROC COPY and PROC CPORT do not support the WHERE statement.
- In PROC GANNO, you should use WHERE= if you want the program code to be portable across all operating systems (fixed in Release 6.10 for Windows but not for VMS).

BEYOND BASICS

ACCESS View Descriptors

There are special considerations for selecting records using WHERE if you use ACCESS view descriptors. In general, limitations exist because the WHERE clause will be passed to the DBMS for processing. Also note that WHERE is more efficient than SubIF because using SubIF returns all rows instead of just the matching subset.

For more efficient WHERE clauses for view descriptors, you should avoid:

- using >= and <= operators instead of BETWEEN
- including numeric conversions
- padding character strings with blanks
- using % or _ in LIKE comparisons
- putting arithmetic expressions on right side of comparison
- searching for NULL or NOT NULL.

There are a variety of issues specific to which DBMS is involved. For instance, the reason searching for NULL can be inefficient for ORACLE is that the index is not used in this situation. You should check the Technical Support notes before creating production view descriptors.

The Original WHERE

Standard SQL WHERE clauses are implemented in PROC SQL in the CREATE, SELECT, DELETE, and UPDATE statements. In addition to allowing basic where-expressions, you can use much more complex constructions. For instance, you can match records across tables just for the purpose of selecting records. The resulting records can be stored in a data set or simply listed in a report. Obviously, a full discussion of SQL is beyond the scope of this paper.

To create a simple subset with all variables, the syntax would be:

```
PROC SQL;
  CREATE TABLE subset AS
   SELECT *
    FROM libref.indata
    WHERE catvar=value;
RUN;
```

Now consider an example given in a manual for PROC SQL:

```
PROC SQL;
  SELECT distinct prodname
    FROM sql.customer AS c, sql.invoice AS i
    WHERE c.custname=i.custname AND
          c.custnum=i.custnum AND
          c.custcity='Myrtle Beach'
   INTERSECT
    SELECT distinct prodname
    FROM sql.invoice AS i, sql.employee AS e
    WHERE e.empnum=i.empnum AND
          e.empcity='Virginia Beach';
```

The resulting records will contain the products that were sold in Myrtle Beach by employees who live in Virginia Beach. Using the selection criteria in a CREATE TABLE statement would allow the data to be saved. How many steps would an equivalent DATA step program require?

WHERE PROCESSING

Using Indexes

Selecting observations using WHERE or WHERE= can take advantage of indexes. SAS software automatically uses an appropriate index by following a standard algorithm that considers index availability and predicted machine resource requirements. Simple or complex indexes are considered given the variables in the where-expression. Usually, a relevant index is used when the predicted number of the selected observations is less than one third. When an index applies, the resulting subset will be sorted in index order as opposed to physical order. If the NOMISS option has been used to create the index, then WHERE processing will use that index only if missing values do not satisfy the where-expression, e.g., DEPT='01'. In Release 6.12, you can optimize WHERE expressions with IN or with both upper and lower bounds against an appropriate composite index.

Efficient Strategies

The primary advantage of using WHERE strategies is machine efficiency. Obviously, if you can avoid a DATA step completely then subset analyses are more efficient both in terms of programming effort and machine resources. If appropriate indexes are created and maintained, WHERE processing will take advantage of them automatically. Planning ahead when creating analysis data sets is the best way to make use of WHERE processing for optimal machine efficiency.

When DATA steps using WHERE or SubIF produce the same subset, the difference in execution logic means that using the WHERE strategy requires less machine resources. This can be true whether or not WHERE processing uses an index. The percentage difference varies widely with the type of data sets and hardware platform but is generally in the 10% to 30% range. Example 3 provides a specific comparison. The advantage of using WHERE processing is greatest when a subset is a small percentage of the original data set and/or the original data set is wide.

Human efficiency becomes a factor when you consider PROC SQL as an alternative. Situations exist when SQL code is much easier to understand that the equivalent manipulation with DATA step code. However, predicting which method is more machine efficient is difficult. As always, testing with your data in your computer environment is necessary to make informed choices.

Procedures

Using WHERE processing is straightforward for Procedures that allow only one input data set. Four procedures adjust to allow for multiple input data sets: APPEND, CALENDAR, COMPARE, DATASETS(APPEND). APPEND allows either a WHERE statement or separate WHERE=. COMPARE allows a WHERE statement that applies to both BASE= and COMPARE= or a WHERE data set option associated with BASE= only.

End-user Interfaces

WHERE functionality is also available in end-user interfaces. With these interfaces, the WHERE clause is constructed by pointing-and-clicking rather than by writing code directly. Examples include the SAS System Viewer and SAS/ASSIST.

The SAS System Viewer was provided with Release 6.12 for personal computers. A user can view data sets, programs, logs, and other types of SAS files without having SAS software. In the viewer, you can subset the data with a WHERE clause.

SAS/ASSIST provides a lookup facility in the WHERE clause window, starting with Release 6.12. The WHERE clause window provides a point-and-click way to build the expressions. The lookup facility provides an added feature of listing the values for the variable selected.

EXAMPLES

Example 1

This example shows an automatic variable that cannot be used with WHERE. The objective of the DATA step in Figure 1 is to merge two data sets and select observations using IN=. Specifically, the patient and investigator data sets are merged in order to select only patients that match the investigator data set. The error message appears because WHERE selection occurs before the IN= variables are defined. This situation requires SubIF code.

Example 2

The results shown in Figures 2b and 2c demonstrate the difference between WHERE and SubIF when MERGE and BY are involved. When WHERE is used, all three sites are selected since the values of NUMPATS are all ten before the merge is done. Since SubIF selection comes after the merge, only Site AA is included in that subset. There are no error messages in this situation, just the potential for erroneous results.

Figure 1. Automatic Variables

```
DATA outdata;
  MERGE      patdata(IN= inpat)
             Invdata(IN= ininv) ;
  BY invest;
   WHERE ininv;  /* same as invinv=1 */

ERROR: VARIABLE ININV is not on file WORK.PATDATA.
ERROR: VARIABLE ININV is not on file WORK.INVDATA
```

Figure 2a. Input Data Sets for MERGE

SITEDATA

SITE	INVNAME	NUMPATS
AA	ALBERT	10
AB	BROWN	10
AC	CORWIN	10

SITEDONE

SITE	INVNAME	NUMPATS
AA	ALBERT	10
AB	BROWN	12
AC	CORWIN	8

Figure 2b. MERGE using WHERE

```
DATA outwhere;
  MERGE sitedata sitedone;
  BY site;
    WHERE numpats=10;
RUN;
```

SITE	INVNAME	NUMPATS
AA	ALBERT	10
AB	BROWN	10
AC	CORWIN	10

Figure 2c. MERGE using SubIF

```
DATA outif;
  MERGE sitedata sitedone;
  BY site;
    IF numpats=10;
RUN;
```

SITE	INVNAME	NUMPATS
AA	ALBERT	10

Example 3

The table in Figure 3 shows differences in processing time when comparing specific WHERE and SubIF strategies. The objective of each program was to process a subset of an existing SAS data set in order to produce a simple frequency table. The five strategies were:

1) DATA with Subsetting IF followed by PROC FREQ
2) DATA using WHERE followed by PROC FREQ
3) PROC SQL followed by PROC FREQ
4) DATA using WHERE on index followed by PROC FREQ
5) PROC FREQ using WHERE

No data manipulation was done in the DATA step besides selecting observations. For the fourth method, a simple index of the variable used to select records was created using PROC DATASETS.

The programs were run on a Pentium II microcomputer using Release 6.12 under Microsoft Windows95. The input data set had 36 variables, mostly numeric, and 27,567 observations. The subset contained 9189 observations (33%). The timing figures are averages of four repetitions.

The exact figures in this example are not as important as the conclusion that using WHERE can reduce processing time in certain situations. Using WHERE with a relevant index clearly can increase

Figure 3. Strategy Comparisons

Strategies	Time in Seconds DATA	PROC	Total
1) DATA with Subsetting IF, PROC	1.44	0.42	1.86
2) DATA with WHERE, PROC	1.33	0.41	1.74
3) PROC SQL, PROC	1.32	0.45	1.77
4) DATA with WHERE, index, PROC	1.25	0.39	1.64
5) PROC using WHERE	--	0.44	0.44

machine efficiency. However, the best strategy is to plan ahead and create data sets so that you can avoid DATA steps altogether in analysis programs. This strategy assumes, of course, that machine efficiency is a priority over the human time that advanced planning may involve.

ENHANCEMENTS

Version 7 Additions

Most of the Version 7 enhancements to WHERE processing involve indexed data sets. To summarize,

- enhanced processing for LIKE and NOT LIKE
- WHERE expressions that contain EQ or IN can now contain
  - directional inequalities (<, <, >=, and so on)
  - NOT operators
  - truncated comparisons
- IDXNAME= to use a specific index
- IDXWHERE= to override automatic index use (YES or NO).

Version 7 makes greater use of composite indexes for compound optimization, which is the process of optimizing multiple WHERE conditions with a single composite index. Version 6 includes the use of compound optimization for a limited number of situations, including where-expressions with

- EQ comparison operators joined with ANDs
- IN operators
- Fully-bounded comparisons, such as 55<AGE<65.

Version 7 expands this list to include many more operators and is correspondingly more complex. The paper by Beatrous and Clifford (1998) gives examples of when a composite index will be used. Generally, it depends on the variables used in the WHERE, the position of those variables in the index, the comparison operators used in the WHERE, and the logical operators used to join clauses (must be AND).

A new option, WHEREUP=, allows you to specify whether or not to evaluate added or modified observations against a WHERE clause.

Frequently Requested Enhancements

WHERE is a valuable part of the SAS programming toolbox and is widely used. In the results from the 1998 SASware Ballot, three WHERE-related items appeared in the top-100. These were:

- Add a note indicating the number of observations that meet the WHERE condition (overall rank of 8)
- Provide the ability to display the WHERE clause in printed output (overall rank of 28)
- In PROC FSEDIT, provide a selection list of previously applied WHERE clauses (overall rank of 72.5).

CONCLUSION

The various forms of WHERE are tools worth exploring when subset creation or processing is necessary. Because of the wide variety of areas in which WHERE can be found, you should use care in researching related problems. While the online Technical Support Notes are an excellent source of information, they can also cause information overload!

As an example, searching on "WHERE" gave 761 hits and searching on "WHERE Statement" gave 393 hits in early 1999. Unless your goal is to exhaustively research WHERE, that is too many. Suppose your goal is to find situations when WHERE gives incorrect results. Your WHERE clause is not giving what you expect and you want to find out if someone else has found the same problem before calling Technical Support. Use "WHERE incorrect results" as a search and you get 195 hits. After thinking further, you use "WHERE incorrect results ASSIST" as a search because you are working with the WHERE clause window in SAS/ASSIST. Now you get 16 hits, which is a much more reasonable number of issues to review.

We recommend that anyone routinely working with subsets make the effort to understand the execution logic behind WHERE processing. Knowing what types of WHERE exist and how to construct where-expressions is the easy part. By considering machine and human efficiency, in the context of processing files with or without indexes and SQL code possibilities, you can better determine when WHERE is the better programming choice in your computer environment.

RECOMMENDED READING

Beatrous, S. and Clifford, W., (1998), "Sometimes You Get What You Want: SAS I/O Enhancements for Version 7," *Proc. of the Twenty-third Annual SAS Group Intl. Conference*, Cary, NC: SAS Institute Inc., 1416–1424. (Version 7 WHERE enhancements.)

Clifford, W., Beatrous, S., Stokes, J.T., and Mosmon, K. (1989), "Using New SAS Database Features and Options," *Proc. of the Fourteenth Annual SAS Group Intl. Conference*, Cary, NC: SAS Institute Inc., 335–346. (Database performance tuning: buffers, compression, index files.)

Gilsen, B., (1998), "SAS Program Efficiency for Beginners," *Proc. of the Twenty-third Annual SAS Group Intl. Conference*, Cary, NC: SAS Institute Inc., 310–320. (PDV, WHERE special operators.)

Henderson, D.J, Rabb, M.G., and Polzin, J.A. (1991), "The SAS Supervisor – A Version 6 Update," *Proc. of the Sixteenth Annual SAS Group Intl. Conference*, Cary, NC: SAS Institute Inc., 249–257. (Logical PDV, DATA step compile and execution differences.)

Hinnenkamp, S. (1997), "Automating Creation of Subsets in SAS/FSP Using Pulldown Menus, SCL and Macros," *Proc. of the Eleventh Annual Northeast SAS Group Conference,* Cary, NC: SAS Institute Inc., 20–30.

Ma, J.M. (1995), "How to Use the WHERE Statement," *Client/Server Computing with the SAS System: Tips and Techniques,* Cary, NC: SAS Institute Inc., 185–189.

Ma, J.M. and Karp, A. (1997), "Efficiency Ideas for Large Files," *Proc. of the Twenty-second Annual SAS Group Intl. Conference*, Cary, NC: SAS Institute Inc., 353–361.

SAS Institute Inc. (1995), *Combining and Modifying SAS Data Sets: Examples, Version 6, First Edition*, Cary, NC: SAS Institute Inc. 197 pp. (SQL examples.)

SAS Institute Inc. (1989), *SAS Guide to the SQL Procedure: Usage and Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc. 210 pp.

Virgile, R. (1997), "Using the WHERE Statement," *Proc. of the Fifth Annual Southeast SAS Group Conference,* Cary, NC: SAS Institute Inc., 63–67.

CONTACT ADDRESS

Dr. J. Meimei Ma
Quintiles
P. O. Box 13979
Research Triangle Park, NC 27709
Voice: 919-998-7136
Fax: 919-998-0972
Internet: mma@qrtp.quintiles.com

Sandra Schlotzhauer
Schlotzhauer Consulting
400 Gibbon Drive
Chapel Hill, NC 27516
Voice: 919-933-6341
Fax: 919-933-5569
Internet: sdschlotz@earthlink.net