CHAPTER

18

WHERE-Expression Processing

Definitions

WHERE-expression processing
: allows you to conditionally select a subset of observations, so that the SAS System processes only the observations that meet a set of specified conditions. For example, if you have a SAS data set containing sales records, you may want to print just the subset of observations for which the sales are greater than $300,000 but less than $600,000. In addition, WHERE-expression processing may improve efficiency of a request. For example, if a WHERE expression can be optimized with an index, SAS does not have to read all observations in the data set to perform the request.

WHERE expression
defines a condition that selected observations must satisfy in order to be processed. You can have a single WHERE expression, referred to as a *simple expression*, such as

```
where sales gt 600000;
```

Or you can have multiple WHERE expressions, referred to as a *compound expression*, such as

```
where sales gt 600000 and salary lt 100000;
```

Where to Use a WHERE Expression

In the SAS System, you can use a WHERE expression in the following situations:

- WHERE statement in both DATA and PROC steps. For example, the following PRINT procedure includes a WHERE statement so that only the observations where the year is greater than 1990 are printed:

```
proc print data=employees;
   where startdate > '01jan1990'd;
run;
```

- WHERE= data set option. The following PRINT procedure includes the WHERE= data set option:

```
proc print data=employees (where=(startdate > '01jan1990'd));
run;
```

- WHERE clause in the SQL procedure, SCL, and SAS/IML software. For example, the following SQL procedure includes a WHERE clause to select only the states where the murder count is greater than seven:

```
proc sql;
   select state from crime
   where murder > 7;
```

- WHERE command in windowing environments like SAS/FSP software. For example,

```
where age > 15
```

- SAS view (DATA step view, SAS/ACCESS view, PROC SQL view), stored with the definition. For example, the following SQL procedure creates an SQL view named STAT from the data file CRIME and defines a WHERE expression for the SQL view definition:

```
proc sql;
   create view stat as
   select * from crime
   where murder > 7;
```

In some cases, you can combine the methods that you use to specify a WHERE expression. That is, you can

- use a WHERE statement in conjunction with a WHERE= data set option
- use a WHERE statement and the WHERE= data set option in windowing procedures and in conjunction with the WHERE command
- use a WHERE statement on a SAS view that has a stored WHERE expression.

For example, it might be useful to combine methods when you merge data sets. That is, you might want different criteria to apply to each data set when you create a subset of data. However, when you combine methods to create a subset of data, there are some restrictions. For example, in the DATA step, if a WHERE statement and a WHERE= data set option apply to the same data set, the data set option takes precedence. For details, see the documentation for the method you are using to specify a WHERE expression.

Note: By default, a WHERE expression does not evaluate added and modified observations. To specify whether a WHERE expression should evaluate updates, you can specify the WHEREUP= data set option. See the WHEREUP= data set option in *SAS Language Reference: Dictionary*. △

Syntax of WHERE Expression

A WHERE expression is a type of SAS expression that defines a condition for selecting observations. A WHERE expression can be as simple as a single variable name or a constant (which is a fixed value). A WHERE expression can be a SAS function, or it can be a sequence of operands and operators that define a condition for selecting observations. In general, the syntax of a WHERE expression is as follows:

WHERE *operand* <*operator*> <*operand*>

operand something to be operated on. An operand can be a variable, a SAS function, or a constant. See “Specifying an Operand” on page 231.

operator a symbol that requests a comparison, logical operation, or arithmetic calculation. All SAS expression operators are valid for a WHERE expression, which include arithmetic, comparison, logical, minimum and maximum, concatenation, parentheses to control order of evaluation, and prefix operators. In addition, you can use special WHERE expression operators, which include BETWEEN-AND, CONTAINS, IS NULL or IS MISSING, LIKE, sounds-like, and SAME-AND. See “Specifying an Operator” on page 233.

For more information on SAS expressions, see Chapter 12, “Expressions,” on page 131.

Specifying an Operand

Variable

A variable is a column in a SAS data set. Each SAS variable has attributes like name and type (character or numeric). The variable type determines how you specify the value for which you are searching. For example:

```
where score > 50;
where date >= '01jan1998'd and time >= '9:00't;
where state = 'Texas';
```

In a WHERE expression, you cannot use automatic variables created by the DATA step (for example, FIRST.*variable*, LAST.*variable*, _N_, or variables created in assignment statements).

As in other SAS expressions, the names of numeric variables can stand alone. SAS treats numeric values of 0 or missing as false; other values are true. For example, the following WHERE expression returns all values for EMPNUM and SSN that are not missing or that have a value of 0:

```
where empnum and ssn;
```

The names of character variables can also stand alone. SAS selects observations where the value of the character variable is not blank. For example, the following WHERE expression returns all values not equal to blank:

```
where lastname;
```

SAS Function

A SAS function returns a value from a computation or system manipulation. Most functions use arguments that you supply, but a few obtain their arguments from the operating environment. To use a SAS function in a WHERE expression, type its name and argument(s) enclosed in parentheses. Some functions you may want to specify include:

- SUBSTR extracts a substring
- TODAY returns the current date
- PUT returns a given value using a given format.

The following DATA step produces a SAS data set that contains only observations from data set CUSTOMER in which the value of NAME begins with **Mac** and the value of variable CITY is **Charleston** or **Atlanta**:

```
data testmacs;
   set customer;
   where substr (name,1,3) = 'Mac' and
   (city='Charleston' or city='Atlanta');
run;
```

Note: SAS functions used in a WHERE expression that can be optimized by an index are the SUBSTR function and the TRIM function. △

For more information on SAS functions, see Chapter 6, "Functions and CALL Routines," on page 43

Constant

A constant is a fixed value such as a number or quoted character string, that is, the value for which you are searching. A constant is a value of a variable obtained from the SAS data set, or values created within the WHERE expression itself. Constants are also called literals. For example, a constant could be a flight number or the name of a city. A constant can also be a time, date, or datetime value.

The value will be either numeric or character. Note the following rules regarding whether to use quotation marks:

- If the value is numeric, do not use quotation marks. For example,

  ```
  where price > 200;
  ```

- If the value is character, use quotation marks. For example,

  ```
  where lastname eq 'Martin';
  ```

- You can use either single or double quotation marks, but do not mix them. Quoted values must be exact matches, including case.

- It may be necessary to use single quotation marks when double quotation marks appear in the value, or use double quotation marks when single quotation marks appear in the value. For example,

```
where item = '6" decorative pot';
where name ? "D'Amico";
```

- A SAS date constant must be enclosed in quotation marks. When you specify date values, case is not important. You can use single or double quotation marks. The following expressions are equivalent:

```
where birthday = '24sep1975'd;
where birthday = "24sep1975"d;
```

Specifying an Operator

Arithmetic Operators

Arithmetic operators allow you to perform a mathematical operation. The arithmetic operators include the following:

Table 18.1 Arithmetic Operators

Symbol	Definition	Example
*	multiplication	where bonus = salary * .10;
/	division	where f = g/h;
+	addition	where c = a+b;
-	subtraction	where f = g-h;
**	exponentiation	where y = a**2;

Comparison Operators

Comparison operators (also called binary operators) compare a variable with a value or with another variable. Comparison operators propose a relationship and ask SAS to determine whether that relationship holds. For example, the following WHERE expression accesses only those observations that have the value 78753 for the numeric variable ZIPCODE:

```
where zipcode eq 78753;
```

The following table lists the comparison operators:

Table 18.2 Comparison Operators

Symbol	Mnemonic Equivalent	Definition	Example
=	EQ	equal to	where empnum eq 3374;
^= or ~= or ¬=	NE	not equal to	where status ne fulltime;
>	GT	greater than	where hiredate gt '01jun1982'd;

Symbol	Mnemonic Equivalent	Definition	Example
<	LT	less than	where empnum < 2000;
>=	GE	greater than or equal to	where empnum >= 3374;
<=	LE	less than or equal to	where empnum <= 3374;
	IN	equal to one from a list of values	where state in ('NC','TX');

When you do character comparisons, you can use the colon (:) modifier to compare only a specified prefix of a character string. For example, in the following WHERE expression, the colon modifier, used after the equals sign, tells SAS to look at only the first character in the values for variable LASTNAME and to select the observations with names beginning with the letter **S**:

```
where lastname=: 'S';
```

Note that in the SQL procedure, the colon modifier used in conjunction with an operator is not supported; you can use the LIKE operator instead.

IN Operator

The IN operator, which is a comparison operator, searches for character and numeric values that are equal to one from a list of values. The list of values must be in parentheses, with each character value in quotation marks and separated by either a comma or blank.

For example, suppose you want all sites that are in North Carolina or Texas. You could specify:

```
where state = 'NC' or state = 'TX';
```

However, the easier way would be to use the IN operator, which says you want any state in the list:

```
where state in ('NC','TX');
```

In addition, you can use the NOT logical operator to exclude a list. For example,

```
where state not in ('CA', 'TN', 'MA');
```

Fully-Bounded Range Condition

A fully-bounded range condition consists of a variable between two comparison operators, specifying both an upper and lower limit. For example, the following expression returns the employee numbers that fall within the range of 500 to 1000 (inclusive):

```
where 500 <= empnum <= 1000;
```

Note that the previous range condition expression is equivalent to the following:

```
where empnum >= 500 and empnum <= 1000;
```

You can combine the NOT logical operator with a fully-bounded range condition to select observations that fall outside the range. Note that parentheses are required:

```
where not (500 <= empnum <= 1000);
```

BETWEEN-AND Operator

The BETWEEN-AND operator is also considered a fully-bounded range condition that selects observations in which the value of a variable falls within an inclusive range of values.

You can specify the limits of the range as constants or expressions. Any range you specify is an inclusive range, so that a value equal to one of the limits of the range is within the range. The general syntax for using BETWEEN-AND is:

WHERE *variable* BETWEEN *value* AND *value*;

For example:

```
where empnum between 500 and 1000;
where taxes between salary*0.30 and salary*0.50;
```

You can combine the NOT logical operator with the BETWEEN-AND operator to select observations that fall outside the range:

```
where empnum not between 500 and 1000;
```

Note: The BETWEEN-AND operator and a fully-bounded range condition produce the same results. That is, the following WHERE expressions are equivalent:

```
where 500 <= empnum <= 1000;
where empnum between 500 and 1000;
```

△

CONTAINS Operator

The most common usage of the CONTAINS (?) operator is to select observations by searching for a specified set of characters within the values of a character variable. The position of the string within the variable's values does not matter; however, the operator is case sensitive when making comparisons.

The following examples select observations having the values **Mobay** and **Brisbayne** for the variable COMPANY, but they do not select observations containing **Bayview**:

```
where company contains 'bay';
where company ? 'bay';
```

You can combine the NOT logical operator with the CONTAINS operator to select observations that are not included in a specified string:

```
where company not contains 'bay';
```

You can also use the CONTAINS operator with two variables, that is, to determine if one variable is contained in another. When you specify two variables, keep in mind the possibility of trailing spaces, which can be resolved using the TRIM function. For example:

```
proc sql;
   select *
   from table1 as a, table 2 as b
   where a.fullname contains trim(b.lastname) and
      a.fullname contains trim(b.firstname);
```

In addition, the TRIM function is helpful when you search on a macro variable. For example:

```
proc print;
   where fullname contains trim("&lname");
```

```
run;
```

IS NULL or IS MISSING Operator

The IS NULL or IS MISSING operator selects observations in which the value of a variable is missing. The operator selects observations with both regular or special missing value characters and can be used for both character and numeric variables. For example:

```
where idnum is missing
where name is null;
```

Using the above examples, the following is equivalent for character data:

```
where name = ' ';
```

And the following is equivalent for numeric data for which missing values can be differentiated with special missing value characters:

```
where idnum <= .Z;
```

You can combine the NOT logical operator with IS NULL or IS MISSING to select nonmissing values, as follows:

```
where salary is not missing;
```

LIKE Operator

The LIKE operator selects observations by comparing the values of a character variable to a specified pattern, which is referred to as pattern matching. The LIKE operator is case sensitive. There are two special characters available for specifying a pattern:

percent sign (%) specifies that any number of characters can occupy that position. The following WHERE expression selects all employees with a name that starts with the letter **N**. The names can be of any length.

```
where lastname like 'N%';
```

underscore (_) matches just one character in the value for each underscore character. You can specify more than one consecutive underscore character in a pattern, and you can specify a percent sign and an underscore in the same pattern. For example, you can use different forms of the LIKE operator to select character values from this list of first names:

Diana

Diane

Dianna

Dianthus

Dyan

The following table shows which of these names is selected by various forms using the LIKE operators:

Pattern	Name Selected
like 'D_an'	Dyan
like 'D_an_'	Diana, Diane
like 'D_an__'	Dianna
like 'D_an%'	all names from list

You can use a SAS character expression to specify a pattern, but you cannot use a SAS character expression that uses a SAS function.

You can combine the NOT logical operator with LIKE to select values that do not have the specified pattern, such as:

```
where frstname not like 'D_an%';
```

Sounds-like Operator

The sounds-like (=*) operator selects observations that contain a spelling variation of a specified word or words. The operator uses the Soundex algorithm to compare the variable value and the operand. For more information on the Soundex algorithm, see the SOUNDEX function in the *SAS Language Reference: Dictionary*.

Although the sounds-like operator is useful, it does not always select all possible values. For example, consider that you want to select observations from the following list of names that sound like Smith:

Schmitt
Smith
Smithson
Smitt
Smythe

The following WHERE expression selects all the names from this list except **Schmitt** and **Smithson**:

```
where lastname=* 'Smith';
```

You can combine the NOT logical operator with the sounds-like operator to select values that do not contain a spelling variation of a specified word or words, such as:

```
where lastname not =* 'Smith';
```

Note: The sounds-like operator cannot be optimized with an index. △

SAME-AND Operator

Use the SAME-AND operator to add more conditions to an existing WHERE expression later in the program without retyping the original conditions. This is useful with:

- interactive SAS procedures
- full-screen SAS procedures that allow you to type a WHERE expression on the command line
- any kind of RUN-group processing.

Use the SAME-AND operator when you already have a WHERE expression defined and you want to insert additional conditions. The SAME-AND operator has the following form:

where-expression-1;
. . . *SAS statements*. . .
WHERE SAME AND *where-expression-2*;
. . . *SAS statements*. . .
WHERE SAME AND *where-expression-n*;

SAS selects observations that satisfy the conditions after the SAME-AND operator in addition to any previously defined conditions. SAS treats all of the existing conditions as though they were conditions separated by AND operators in a single WHERE expression.

The following example shows how to use the SAME-AND operator within RUN groups in the GPLOT procedure. The SAS data set YEARS has three variables and contains quarterly data for the 1990–1997 period:

```
proc gplot data=years;
   plot unit*quar=year;
run;

   where year > '01jan1991'd;
run;

   where same and year < '01jan1996'd;
run;
```

The following WHERE expression is equivalent to the preceding code:

```
where year > '01jan1991'd and year < '01jan1996'd;
```

MIN and MAX Operators

Use the MIN and MAX operators to find the minimum or maximum value of two quantities. Surround the operators with the two quantities whose minimum or maximum value you want to know.

- The MIN operator returns the lower of the two values.
- The MAX operator returns the higher of two values.

For example, if A is less than B, then the following would return the value of A:

```
where x = (a min b);
```

Note: The symbol representation >< is not supported, and <> is interpreted as not equals. △

Concatenation Operator

The concatenation operator concatenates character values. You indicate the concatenation operator as follows:

- || (two OR symbols)
- !! (two explanation marks)
- ¦¦ (two broken vertical bars).

For example,

```
where name = 'John'||'Smith';
```

Prefix Operators

The plus sign (+) and minus sign (-) can be either prefix operators or arithmetic operators. They are prefix operators when they appear at the beginning of an

expression or immediately preceding a left parentheses. A prefix operator is applied to the variable, constant, SAS function, or parenthetic expression. For example,

```
where z = -(x + y);
```

Note: The NOT operator is also considered a prefix operator. △

Combining Expressions Using Logical Operators

Syntax

You can combine or modify WHERE expressions by using the logical operators (also called Boolean operators) AND, OR, and NOT. The basic syntax of a compound WHERE expression is:

WHERE *where-expression-1 logical-operator where-expression-n* ;

AND — combines two conditions by finding observations that satisfy both conditions. For example:

```
where skill eq 'cobol' and years eq 4;
```

OR — combines two conditions by finding observations that satisfy either condition or both. For example:

```
where skill eq 'cobol' or years eq 4;
```

NOT — modifies a condition by finding the complement of the specified criteria. You can use the NOT logical operator in combination with any SAS and WHERE expression operator. And you can combine the NOT operator with AND and OR. For example:

```
where skill not eq 'cobol' or years not eq 4;
```

The logical operators and their equivalent symbols are shown in the following table:

Table 18.3 Logical (Boolean) Operators

Symbol	Mnemonic Equivalent
&	AND
! or | or ¦	OR
^ or ~ or ¬	NOT

Processing Compound Expressions

When SAS encounters a compound WHERE expression (multiple conditions), the software follows rules to determine the order in which to evaluate each expression. When WHERE expressions are combined, SAS processes the conditions in a specific order:

1. The NOT expression is processed first.
2. Then the expressions joined by AND are processed.

3 Finally, the expressions joined by OR are processed.

For a complete discussion of the rules for evaluating compound expressions, see "Order of Evaluation in Compound Expressions" on page 144.

Using Parentheses to Control Order of Evaluation

Even though SAS evaluates logical operators in a specific order, you can control the order of evaluation by nesting expressions in parentheses. That is, an expression enclosed in parentheses is processed before one not enclosed. The expression within the innermost set of parentheses is processed first, followed by the next deepest, moving outward until all parentheses have been processed.

For example, suppose you want a list of all the Canadian sites that have both SAS/GRAPH and SAS/FSP software, so you issue the following expression:

```
where product='GRAPH' or product='FSP' and country='Canada';
```

The result, however, includes all sites that license SAS/GRAPH software along with the Canadian sites that license SAS/FSP software. To obtain the correct results, you can use parentheses, which causes SAS to evaluate the comparisons within the parentheses first, providing a list of sites with either product licenses, then the result is used for the remaining condition:

```
where (product='GRAPH' or product='FSP') and country='Canada';
```

Constructing Efficient WHERE Expressions

Indexing a SAS data set can significantly improve performance of WHERE processing. An index is an optional file that you can create for SAS data files in order to provide direct access to specific observations. Processing a WHERE expression without an index requires SAS to sequentially read every observation to find the ones that match the selection criteria. Having an index allows the software to determine which observations satisfy the criteria without having to read all the observations, which is referred to as optimizing the WHERE expression. However, by default, SAS decides whether to use the index or read the entire data set sequentially. For details on how SAS uses an index to process a WHERE expression, see "Using an Index for WHERE Processing" on page 441.

In addition to creating indexes for the data set, here are some guidelines for writing efficient WHERE expressions:

Table 18.4 Constructing Efficient WHERE Expressions

Guideline	Efficient	Inefficient
Avoid using the LIKE operator that begins with % or _.	where country like 'A%INA';	where country like '%INA';
Avoid using arithmetic expressions.	where salary > 48000;	where salary > 12*4000;
Use the IN operator instead of a compound expression.	where state in ('NC' , 'PA' , 'VA');	where state ='NC' or state = 'PA' or state = 'VA';

Deciding Whether to Use a WHERE Expression or a Subsetting IF Statement

To conditionally select observations from a SAS data set, you can use either a WHERE expression or a subsetting IF statement. While they both test a condition to determine if SAS should process an observation, they differ as follows:

- The subsetting IF statement can be used only in a DATA step. A subsetting IF statement tests the condition after an observation is read into the Program Data Vector (PDV). If the condition is true, SAS continues processing the current observation. Otherwise, the observation is discarded, and processing continues with the next observation.
- You can use a WHERE expression in both a DATA step and SAS procedures, as well as in a windowing environment, SCL programs, and as a data set option. A WHERE expression tests the condition before an observation is read into the PDV. If the condition is true, the observation is read into the PDV and processed. If the condition is false, the observation is not read into the PDV, and processing continues with the next observation, which can yield substantial savings when observations contain many variables or very long character variables (up to 32K bytes). Additionally, a WHERE expression can be optimized with an index, and the WHERE expression allows more operators, such as LIKE and CONTAINS.

  Note: Although it is generally more efficient to use a WHERE expression and avoid the move to the PDV prior to processing, if the data set contains observations with very few variables, the move to the PDV could be cheap. However, one variable containing 32K bytes of character data is not cheap, even though it is only one variable. △

In most cases, you can use either method. However, the following table provides a list of tasks that require you to use a specific method:

Table 18.5 Tasks Requiring Either WHERE Expression or Subsetting IF Statement

If you want to ...	Use a ...
Make the selection in a procedure without using a preceding DATA step	WHERE expression
Take advantage of the efficiency available with an indexed data set	WHERE expression

If you want to …	Use a …
Use one of a group of special operators, such as BETWEEN-AND, CONTAINS, IS MISSING or IS NULL, LIKE, SAME-AND, and SOUNDS LIKE	WHERE expression
Base the selection on anything other than a variable value that already exists in a SAS data set, for example, on a value that is read from raw data or on a value that is calculated or assigned during the course of the DATA step	subsetting IF
Make the selection at some point during a DATA step rather than at the beginning	subsetting IF
Execute the selection conditionally	subsetting IF

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS Language Reference: Concepts*, Cary, NC: SAS Institute Inc., 1999. 554 pages.

SAS Language Reference: Concepts

ISBN 1–58025–441–1

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, November 1999