

SQL Processing with SAS® Tip Sheet

This tip sheet is associated with the SAS® Certified Professional Prep Guide Advanced Programming Using SAS® 9.4. For more information, visit www.sas.com/certify

Basic Queries

```
PROC SQL <options>;
SELECT column-1 <, ...column-n>
FROM input-table
<WHERE expression>
<GROUP BY col-name>
<HAVING expression>
<ORDER BY col-name> <DESC> <,...col-name>;
```

SQL Query Order of Execution:

Clause	Description
SELECT	Retrieve data from a table
FROM	Choose and join tables
WHERE	Filter the data
GROUP BY	Aggregate the data
HAVING	Filter the aggregate data
ORDER BY	Sort the final data

Managing Tables

CREATE TABLE	CREATE TABLE <i>table-name</i> (<i>column-specification-1</i> <, ... <i>column-specification-n</i> >);
DESCRIBE TABLE	DESCRIBE TABLE <i>table-name-1</i> <,... <i>table-name-n</i> >;
DROP TABLE	DROP TABLE <i>table-name-1</i> <,... <i>table-name-n</i> >;

Managing Views

CREATE VIEW	CREATE VIEW <i>table-name</i> AS <i>query</i> ;
DESCRIBE VIEW	DESCRIBE VIEW <i>view-name-1</i> <,... <i>view-name-n</i> >;
DROP VIEW	DROP VIEW <i>view-name-1</i> <,... <i>view-name-n</i> >;

Modifying Columns

LABEL=	SELECT <i>col-name</i> LABEL= ' <i>column label</i> '
FORMAT=	SELECT <i>col-name</i> FORMAT= <i>format</i> .
Creating a new column	SELECT <i>col-name</i> AS <i>new-col-name</i>
Filtering new columns	WHERE CALCULATED <i>new-col-name</i>

Modifying Rows

Inserting rows into tables	INSERT INTO <i>table</i> SET <i>column-name=value</i> <,... <i>column-name=value</i> >; INSERT INTO <i>table</i> <(column-list)> VALUES (<i>value</i> <,... <i>value</i> >); INSERT INTO <i>table</i> <(column-list)> SELECT <i>column-1</i> <,... <i>column-n</i> > FROM <i>input-table</i> ;
Eliminating duplicate rows	SELECT DISTINCT <i>col-name</i> <,... <i>col-name</i> >
Filtering rows	WHERE <i>col-name</i> IN (<i>value1</i> , <i>value2</i> , ...) WHERE <i>col-name</i> LIKE "_ <i>string</i> %" WHERE <i>col-name</i> BETWEEN <i>value</i> AND <i>value</i> WHERE <i>col-name</i> IS NULL WHERE <i>date-value</i> "<01JAN2019>" d WHERE <i>time-value</i> "<14:45:35>" t WHERE <i>datetime-value</i> "<01JAN201914:45:35>" dt

Remerging Summary Statistics

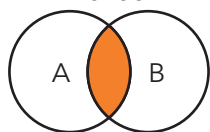
```
SELECT col-name, summary function(argument)
FROM input table;
```

SQL Processing with SAS® Tip Sheet

This tip sheet is associated with the SAS® Certified Professional Prep Guide Advanced Programming Using SAS® 9.4. For more information, visit www.sas.com/certify

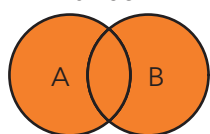
Joins Summary

Inner Join



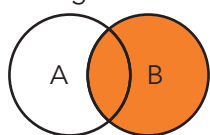
```
SELECT <list>
FROM table-A INNER JOIN table-B
ON A.Key=B.Key;
```

Full Join



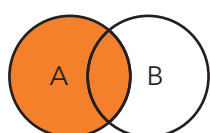
```
SELECT <list>
FROM table-A FULL JOIN table-B
ON A.Key=B.Key;
```

Right Join



```
SELECT <list>
FROM table-A RIGHT JOIN table-B
ON A.Key=B.Key;
```

Left Join



```
SELECT <list>
FROM table-A LEFT JOIN table-B
ON A.Key=B.Key;
```

Creating Macro Variables

Storing a value in a macro variable using SQL:

```
SELECT col-name-1 <,...col-name-n>
INTO:macvar_1<,...macvar-n>
FROM input-table;
```

Storing a list of values in a macro variable using SQL:

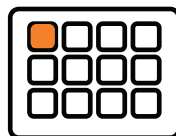
```
SELECT col-name-1 <,...col-name-n>
INTO:macvar_1 SEPARATED BY 'delimiter'
FROM input-table;
```

Viewing the value of the macro variable in the SAS Log:

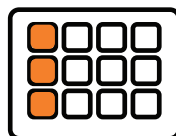
```
%PUT &=macvar;
```

Subqueries

```
SELECT col-name,
(SELECT function(argument)
FROM input-table)
FROM input-table;
```

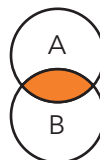


```
SELECT col-name, <,...col-name>
FROM input-table
WHERE col-name
(SELECT function(argument)
FROM input-table)
```



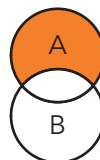
Set Operators

The INTERSECT operator selects unique rows that are common to both tables.



```
SELECT <list>
FROM table-A INTERSECT
SELECT <list>
FROM table-B;
```

The EXCEPT operator selects unique rows from table A that are not found in table B.



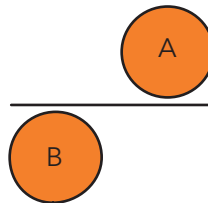
```
SELECT <list>
FROM table-A EXCEPT
SELECT <list>
FROM table-B;
```

The UNION operator selects unique rows from both tables.



```
SELECT <list>
FROM table-A UNION
SELECT <list>
FROM table-B;
```

The OUTER UNION operator selects all rows from both tables.



```
SELECT <list>
FROM table-A OUTER UNION
SELECT <list>
FROM table-B;
```

Accessing DBMS Data

The SQL pass-through facility enables you to code in the native DBMS SQL syntax and pass the query to the database.

```
PROC SQL;
CONNECT TO DBMS-name <AS alias>
(DBMS-connection-options);

SELECT col-name
FROM CONNECTION TO DBMS-name|alias (dbms-query);
DISCONNECT FROM DBMS-name|alias;
QUIT;
```

SAS[®] Macro Language Processing

This tip sheet is associated with the SAS[®] Certified Professional Prep Guide Advanced Programming Using SAS[®] 9.4. For more information, visit www.sas.com/certify

SAS Macro Facility



Creating Macro Variables

Syntax	Description
%GLOBAL <i>macro-variable-1</i> ... <i>macro-variable-n</i> ;	Creates a macro variable that is available during the execution of the entire SAS session.
%LET <i>variable=value</i> ;	Creates a macro variable and assigns it a value.
%LOCAL <i>macro-variable-1</i> ... <i>macro-variable-n</i> ;	Creates a macro variable that is available only during the execution of the macro where it is defined.

Defining a Macro

%MACRO *macro-name* <(parameter-list)>;
macro-text

%MEND <*macro-name*>;

The *parameter-list* can be:

<*positional-parameter-1*, ...*positional-parameter-n*> or
<*keyword-1=value-1*, ...*keyword-n=value-n*>

Calling a Macro

%macro-name

%macro-name(*positional-parameter-1*,
...*positional-parameter-n*)

%macro-name(*keyword-1=value-1*, ...*keyword-n=value-n*)

Referencing a Macro Variable

Use the name of the macro variable with an ampersand.

&*macro-variable*;

Macro Character Functions

Syntax	Description
%INDEX (<i>source</i> , <i>string</i>)	Determines the position of the first character of a string within another string.
%SCAN (<i>argument</i> , <i>n</i> , < <i>charlist</i> < , <i>modifiers</i> >>)	Searches the argument and returns the <i>n</i> th word.
%SUBSTR (<i>argument</i> , <i>position</i> , <, <i>length</i> >)	Produces a substring of character string (<i>argument</i>) by extracting the specified number of characters (<i>length</i>) beginning at the specified starting position.
%UPCASE (<i>character-string</i> <i>text-expression</i>)	Converts lowercase characters in the argument to uppercase.

SAS Functions with Macro Variables

Syntax	Description
%EVAL (<i>arithmetic</i> or <i>logical expression</i>)	Evaluates arithmetic and logical expressions using integer arithmetic.
%SYSEVALF (<i>expression</i> <, <i>conversion-type</i> >)	Evaluates arithmetic and logical expressions using floating-point arithmetic.
%SYSFUNC (<i>function</i> (<i>argument-1</i> <... <i>argument-n</i> >) <, <i>format</i> >)	Executes SAS functions or user-written functions in the macro facility.

Troubleshooting Macro Variable References

Enables you to write your own messages to the SAS log.

%PUT *text*;

Deletes the specified variables from the macro global symbol table.

%SYMDEL *macro-variable-1* <...*macro-variable-n*></option>;

SAS[®] Macro Language Processing

This tip sheet is associated with the SAS[®] Certified Professional Prep Guide Advanced Programming Using SAS[®] 9.4. For more information, visit www.sas.com/certify

Masking Special Characters

Syntax	Description
%STR (<i>argument</i>)	Hides the usual meaning of a semicolon(;) so it appears as constant text.
%NRSTR (<i>character-string</i>)	Hides the usual meaning of an ampersand (&) or a percent sign (%) so they appear as constant text.
%SUPERQ (<i>argument</i>)	Masks all special characters and mnemonic operators at macro execution but prevents further resolution of the value.
%BQUOTE (<i>character-string</i> <i>text-expression</i>)	Masks special characters and mnemonic operators in a resolved value at macro execution.
%QUPCASE (<i>character-string</i> <i>text-expression</i>)	Converts values to uppercase and returns a result that masks special characters and mnemonic operators.
%QSUBSTR (<i>argument</i> , <i>position</i> <, <i>length</i> >)	Produces a substring of a character string.
%QSCAN (<i>argument</i> , <i>n</i> <, <i>charlist</i> <, <i>modifiers</i> >>)	Searches for a word and masks special characters and mnemonic operators.
%QSYSFUNC (<i>function</i> (<i>arguments</i>) <, <i>format</i> >)	Executes functions and masks special characters and mnemonic operators.

Conditional Processing

```
%IF expression %THEN text;  
<%ELSE text>;
```

```
%IF expression %THEN %DO;  
    text and/or macro language statements;  
%END;  
%ELSE %DO;  
    text and/or macro language statements;  
%END;
```

```
%DO index-variable=start %TO stop <%BY increment>;  
    text  
%END;
```

Options

```
OPTIONS MCOMPILENOTE= NONE | NOAUTOCALL | ALL;  
OPTIONS MPRINT | NOMPRINT;  
OPTIONS MLOGIC | NOMLOGIC;  
OPTIONS MAUTOSOURCE | NOAUTOSOURCE;
```

Creating Macros in SQL

```
PROC SQL NOPRINT;  
    SELECT column1 <,column2,...>  
        INTO :macro-variable-1 <,:macro-variable-2,...>  
            <TRIMMED>  
    FROM table-1 | view-1  
    <WHERE expression>  
    <other clauses>;  
QUIT;
```

DATA Step Interface

```
CALL SYMPUTX(macro-variable-name, value  
            <,symbol-table>);  
PUT(source,format.);
```

Advanced Macro Techniques

```
%INCLUDE file-specification </SOURCE2>;  
DOSUBL(text-string);
```

Default Autocall Library

```
%LOWCASE(argument)  
%QLOWCASE(argument)  
%LEFT(argument)  
%TRIM(argument)  
%CMPRES(argument)  
%DATATYP(argument)
```

Advanced SAS® Programming Techniques

This tip sheet is associated with the SAS® Certified Professional Prep Guide Advanced Programming Using SAS® 9.4. For more information, visit www.sas.com/certify

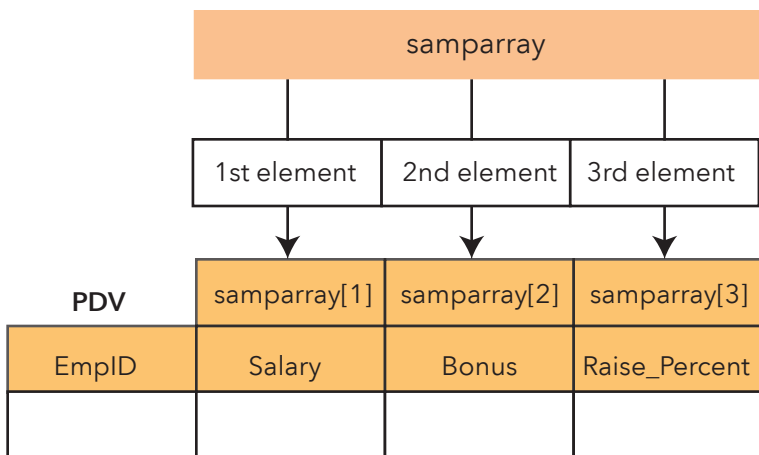
Arrays

Defining an array

```
ARRAY array-name<[number-of-array-elements]>  
<$> <length> <array-elements>  
<_TEMPORARY_> <(initial-values)>;
```

Referencing an array

```
array-name[element-number];
```



The number of elements must be enclosed in either parentheses (), braces { }, or brackets [].

Unknown Number of Elements

Use an asterisk (*) within your brackets when defining an array.

Use the DIM function to return the number of elements in an array.

```
DIM(array-name);
```

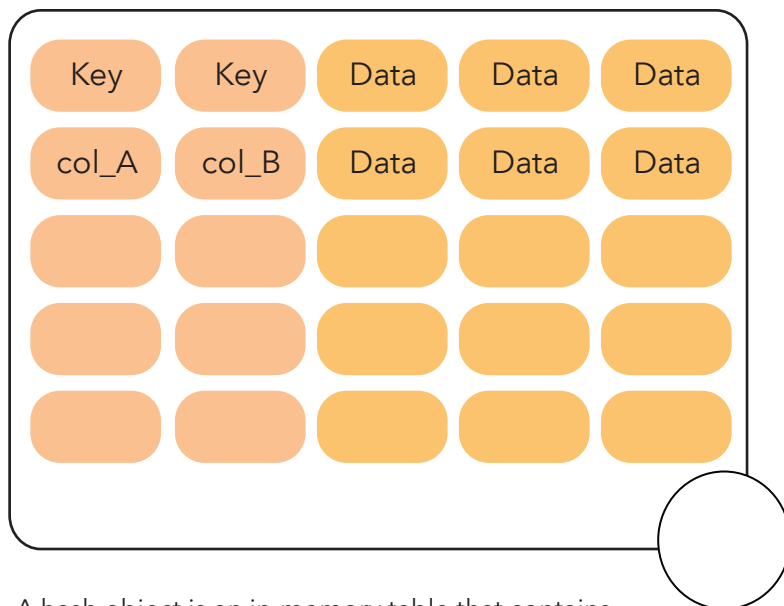
Two-Dimensional Arrays

```
ARRAY array-name  
[number-of-rows,number-of-columns];
```

```
array samplearray[3,2];
```

The example above creates an array named SampleArray which has 3 rows and 2 columns.

Hash Objects



A hash object is an in-memory table that contains key and data components.

Hash Object and Iterator Process

Declaring hash object or hash iterator object:

```
DECLARE hash object-name  
(<argument_tag-1:value-1, ...>);
```

```
DECLARE hiter object-name  
( 'hash-object-name' );
```

Defining a hash object:

```
object-name.ADD( );  
object-name.DEFINEKEY('key-1' <, ...'key-n'>);  
object-name.DEFINEDATA('data-1' <, ...'data-n'>);  
object-name.DEFINEDONE( );  
object-name.OUTPUT( );
```

Using a hash object:

```
object-name.FIND( );
```

Retrieving a hash object with a hash iterator object:

```
object-name.FIRST( );  
object-name.LAST();  
object-name.NEXT();  
object-name.PREV( );
```

Advanced SAS® Programming Techniques

This tip sheet is associated with the SAS® Certified Professional Prep Guide Advanced Programming Using SAS® 9.4. For more information, visit www.sas.com/certify

Picture Formats

```
PROC FORMAT;  
  PICTURE format-name <(format-options)>  
  <value-range-set-1= 'template-value' (template-options)>  
  <value-range-set-n= 'template-value' (template-options)>;  
RUN;
```

Options

Creating Custom Date, Time, Datetime Formats

DATATYPE=DATE | TIME | DATETIME
enables the use of directives in the picture as a template to format date, time, or datetime values.

DEFAULT=length
specifies the default length of the picture.

Creating Custom Numeric Formats

MULT|MULTIPLIER=*n*
specifies a number to multiply the value by.

PREFIX=*'prefix'*
specifies a character prefix to place in front of the formatted value.

ROUND
rounds the value to the nearest integer before formatting.

Creating Functions

```
PROC FCMP OUTLIB=libref.table.package;  
  FUNCTION function-name(arguments) <$> <length>;  
  ... programming statements ...  
  RETURN(expression);  
ENDSUB;  
QUIT;
```

Using Custom Functions

OPTIONS CMPLIB=*libref.table* | (*libref.table-1...libref.table-n*)

Advanced Functions

```
LAG<n>(column);  
COUNT(string, substring<,modifiers>);  
COUNTC(string, character-list<,modifiers>);  
COUNTW(string, <,delimiters><,modifiers>);  
FIND(string, substring<,modifiers><,start-position>);  
FINDC(string, character-list<,modifiers> <,start-position>);  
FINDW(string, word<,delimiters><,modifiers><,start-position>);
```

Perl Regular Expressions

Perl Regular Expressions Metacharacters

Metacharacter	Description
/.../	Starting and ending delimiter
(...)	Enables grouping
	Denotes the OR situation
\d	Matches a digit (0-9)
\D	Matches a non-digit such as letter
\s	Matches a whitespace character
\w	Matches a group of characters
.	Matches any character
[...]	Matches a character in brackets
[^...]	Matches a character not in brackets
^	Matches the beginning of the string
\$	Matches the end of the string
\b	Matches a word boundary
\B	Matches a non-word boundary
*	Matches the preceding character 0 or more times
+	Matches the preceding character 1 or more times
?	Matches the preceding character 0 or 1 times
{ <i>n</i> }	Matches exactly <i>n</i> times
\	Overrides the next metacharacter such as a (or ?)

PRXMATCH Function

PRXMATCH function searches for a pattern match and returns the position at which the pattern is found.

PRXMATCH (*Perl-regular-expression*, *source*);

PRXPARSE Function

PRXPARSE function returns a pattern identifier number that is used by other PRX functions and call routines.

pattern-ID-number=PRXPARSE (*Perl-regular-expression*);

PRXCHANGE Function

PRXCHANGE function performs a substitution for a pattern match

PRXCHANGE (*Perl-regular-expression*, *times*, *source*)