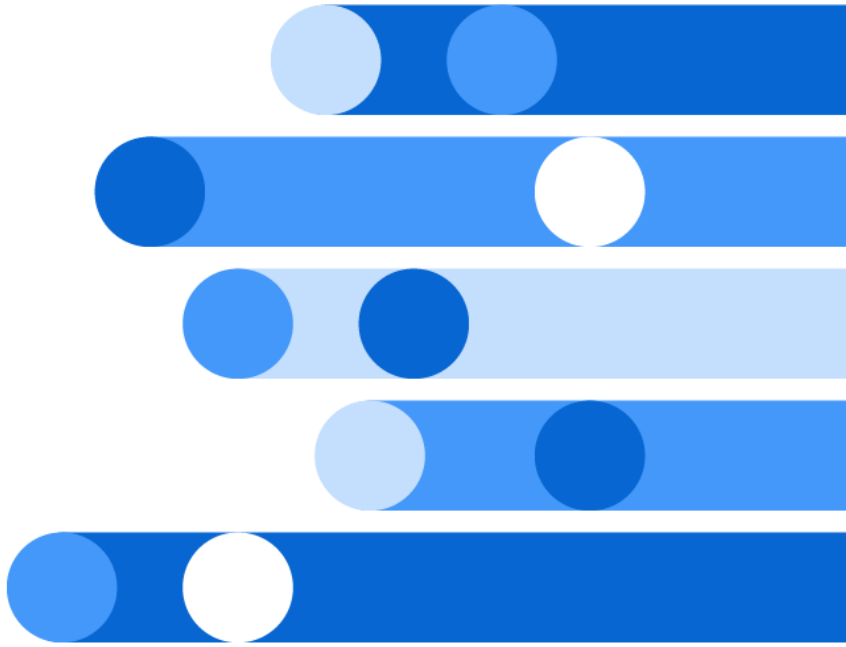
SAS/ETS® 15.4 User's Guide
The COMPUTAB Procedure



SAS® Documentation
June 17, 2025

This document is an individual chapter from *SAS/ETS® 15.4 User's Guide*.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2025. *SAS/ETS® 15.4 User's Guide*. Cary, NC: SAS Institute Inc.

SAS/ETS® 15.4 User's Guide

June 2025

Chapter 10
The COMPUTAB Procedure

Contents

Overview: COMPUTAB Procedure

The COMPUTAB (computing and tabular reporting) procedure produces tabular reports generated using a programmable data table.

The COMPUTAB procedure is especially useful when you need both the power of a programmable spreadsheet and a report generation system, but you want to set up a program to run in a batch mode and generate routine reports.

With PROC COMPUTAB, you can select a subset of observations from the input data set, define the format of a table, operate on its row and column values, and create new columns and rows. Access to individual table values is available when needed.

The COMPUTAB procedure can tailor reports to almost any desired specification and provide consolidation reports over summarization variables. The generated report values can be stored in an output data set. PROC COMPUTAB is especially useful in creating tabular reports such as income statements, balance sheets, and other row and column reports.

Getting Started: COMPUTAB Procedure

The following example shows the different types of reports that can be generated by PROC COMPUTAB.

Suppose a company has monthly expense data on three of its divisions and wants to produce the year-to-date expense report shown in Figure 10.1. This section starts out with the default report produced by the COMPUTAB procedure and modifies it until the desired report is achieved.

Figure 10.1 Year-to-Date Expense Report

Year to Date Expenses

	Division A	Division B	Division C	All Divisions
Travel Expenses within U.S.	18700	211000	12800	$242,500
Advertising	18500	176000	34500	$229,000
Permanent Staff Salaries	186000	1270000	201000	$1,657,000
Benefits Including Insurance	3900	11100	17500	$32,500
Total	227100	1668100	265800	$2,161,000

Producing a Simple Report

Without any specifications, the COMPUTAB procedure transposes and prints the input data set. The variables in the input data set become rows in the report, and the observations in the input data set become columns. The variable names are used as the row titles. The column headings default to COL1 through COLn. For example, the following input data set contains the monthly expenses reported by different divisions of the company:

```
data report;
   length compdiv $ 1;
   input compdiv $ date:date7. salary travel insure advrtise;
   format date date7.;
   label travel   = 'Travel Expenses within U.S.'
         advrtise = 'Advertising'
         salary   = 'Permanent Staff Salaries'
         insure   = 'Benefits Including Insurance';
datalines;
A 31JAN1989 95000  10500  2000 6500
B 31JAN1989 668000 112000 5600 90000
C 31JAN1989 105000 6800   9000 18500
A 28FEB1989 91000  8200   1900 12000
B 28FEB1989 602000 99000  5500 86000
C 28FEB1989 96000  6000   8500 16000
;
```

You can get a listing of the data set by using the PRINT procedure, as follows:

```
title 'Listing of Monthly Divisional Expense Data';
proc print data=report;
run;
```

Figure 10.2 Listing of Data Set by PROC PRINT

Listing of Monthly Divisional Expense Data

Obs	compdiv	date	salary	travel	insure	advrtise
1	A	31JAN89	95000	10500	2000	6500
2	B	31JAN89	668000	112000	5600	90000
3	C	31JAN89	105000	6800	9000	18500
4	A	28FEB89	91000	8200	1900	12000
5	B	28FEB89	602000	99000	5500	86000
6	C	28FEB89	96000	6000	8500	16000

To get a simple, transposed report of the same data set, use the following PROC COMPUTAB statement:

```
title 'Monthly Divisional Expense Report';
proc computab data=report;
run;
```

Figure 10.3 Listing of Data Set by PROC COMPUTAB

Monthly Divisional Expense Report

	COL1	COL2	COL3	COL4	COL5	COL6
compdiv	A	B	C	A	B	C
date	31JAN89	31JAN89	31JAN89	28FEB89	28FEB89	28FEB89
salary	95000.00	668000.00	105000.00	91000.00	602000.00	96000.00
travel	10500.00	112000.00	6800.00	8200.00	99000.00	6000.00
insure	2000.00	5600.00	9000.00	1900.00	5500.00	8500.00
advrtise	6500.00	90000.00	18500.00	12000.00	86000.00	16000.00

Using PROC COMPUTAB

The COMPUTAB procedure is best understood by examining the following features:

- definition of the report layout with ROWS and COLUMNS statements
- input block
- row blocks
- column blocks

PROC COMPUTAB builds a table according to the specifications in the ROWS and COLUMNS statements. Row names and column names define the rows and columns of the table. Options in the ROWS and COLUMNS statements control titles, spacing, and formatting.

The input block places input observations into the appropriate columns of the report. It consists of programming statements used to select observations to be included in the report, to determine the column into which the observation should be placed, and to calculate row and column values that are not in the input data set.

Row blocks and column blocks perform operations on the values of rows and columns of the report after the input block has executed. Row blocks are a block of programming statements labeled ROW*xxxxx*: that create or modify row values; column blocks are a block of programming statements labeled COL*xxxxx*: that create or modify column values. Row and column blocks can make multiple passes through the report for final calculations.

For most reports, these features are sufficient. More complicated applications might require knowledge of the program data vector and the COMPUTAB data table. These topics are discussed in the section "Details: COMPUTAB Procedure" on page 483.

Defining Report Layout

ROWS and COLUMNS statements define the rows and columns of the report. The order of row and column names in these statements determines the order of rows and columns in the report. Additional ROWS and COLUMNS statements can be used to specify row and column formatting options.

The following statements select and order the variables from the input data set and produce the report in Figure 10.4:

```
proc computab data=report;
   rows travel advrtise salary;
run;
```

Figure 10.4 Report Produced Using a ROWS Statement

	COL1	COL2	COL3	COL4	COL5	COL6
TRAVEL	10500.00	112000.00	6800.00	8200.00	99000.00	6000.00
ADVRTISE	6500.00	90000.00	18500.00	12000.00	86000.00	16000.00
SALARY	95000.00	668000.00	105000.00	91000.00	602000.00	96000.00

When a COLUMNS statement is not specified, each observation becomes a new column. If you use a COLUMNS statement, you must specify to which column each observation belongs by using program statements for column selection. When more than one observation is selected for the same column, values are summed.

The following statements produce Figure 10.5:

```
proc computab data= report;
   rows travel advrtise salary insure;
   columns a b c;
   *----select column for company division,
        based on value of compdiv----*;
   a = compdiv = 'A';
   b = compdiv = 'B';
   c = compdiv = 'C';
run;
```

The statement A=COMPDIV='A'; illustrates the use of logical operators as a selection technique. If COMPDIV='A', then the current observation is added to the A column. For more information, see *SAS Language: Reference, Version 6, First Edition*.

Figure 10.5 Report Produced Using ROWS and COLUMNS Statements

	A	B	C
TRAVEL	18700.00	211000.00	12800.00
ADVRTISE	18500.00	176000.00	34500.00
SALARY	186000.00	1270000.0	201000.00
INSURE	3900.00	11100.00	17500.00

Adding Computed Rows and Columns

In addition to the variables and observations in the input data set, you can create additional rows or columns by using SAS programming statements in PROC COMPUTAB. You can do the following:

- modify input data and select columns in the input block
- create or modify columns in column blocks
- create or modify rows in row blocks

The following statements add one computed row (SUM) and one computed column (TOTAL) to the report in Figure 10.5. In the input block the logical operators indicate the observations that correspond to each column of the report. After the input block reads in the values from the input data set, the column block creates the column variable TOTAL by summing the columns A, B, and C. The additional row variable, SUM, is calculated as the sum of the other rows. The result is shown in Figure 10.6.

```
proc computab data= report;
   rows travel advrtise salary insure sum;
   columns a b c total;
   a = compdiv = 'A';
   b = compdiv = 'B';
   c = compdiv = 'C';
   colblk: total = a + b + c;
   rowblk: sum   = travel + advrtise + salary + insure;
run;
```

Figure 10.6 Report Produced Using Row and Column Blocks

	A	B	C	TOTAL
TRAVEL	18700.00	211000.00	12800.00	242500.00
ADVRTISE	18500.00	176000.00	34500.00	229000.00
SALARY	186000.00	1270000.0	201000.00	1657000.0
INSURE	3900.00	11100.00	17500.00	32500.00
SUM	227100.00	1668100.0	265800.00	2161000.0

Enhancing the Report

To enhance the appearance of the final report, you can use the following:

- TITLE and LABEL statements
- column headings
- row titles
- row and column spacing control
- overlining and underlining

- formats

The following example enhances the report in the previous example. The enhanced report is shown in Figure 10.7.

The TITLE statement assigns the report title. The column headings in Figure 10.7 (Division A, Division B, and Division C) are assigned in the first COLUMNS statement by "Division" _name_ specification. The second COLUMNS statement assigns the column heading ("All" "Divisions"), sets the spacing (+4), and formats the values in the TOTAL column.

Similarly, the first ROWS statement uses previously assigned variable labels for row labels by specifying the _LABEL_ option. The DUL option in the second ROWS statement double-underlines the INSURE row. The third ROWS statement assigns the row label TOTAL to the SUM row.

```
title 'Year to Date Expenses';

proc computab cwidth=8 cdec=0;

   columns a  b  c / 'Division' _name_;
   columns total / 'All' 'Divisions' +4 f=dollar10.0;

   rows travel advrtise salary insure / _label_;
   rows insure / dul;
   rows sum / 'Total';

   a = compdiv = 'A';
   b = compdiv = 'B';
   c = compdiv = 'C';

   colblk: total = a + b + c;
   rowblk: sum   = travel + advrtise + salary + insure;
run;
```

Figure 10.7 Report Produced by PROC COMPUTAB Using Enhancements

Year to Date Expenses

	Division A	Division B	Division C	All Divisions
Travel Expenses within U.S.	18700	211000	12800	$242,500
Advertising	18500	176000	34500	$229,000
Permanent Staff Salaries	186000	1270000	201000	$1,657,000
Benefits Including Insurance	3900	11100	17500	$32,500
Total	227100	1668100	265800	$2,161,000

Syntax: COMPUTAB Procedure

The following statements are used with the COMPUTAB procedure:

PROC COMPUTAB *options* **;**
BY *variables* **;**
COLUMNS *column-list / options* **;**
ROWS *row-list / options* **;**
CELL *cell-names* / **FORMAT=** *format* **;**
INIT *anchor-name locator-name values locator-name values* **;**
programming *statements* **;**
SUMBY *variables* **;**

The PROC COMPUTAB statement is the only required statement. The COLUMNS, ROWS, and CELL statements define the COMPUTAB table. The INIT statement initializes the COMPUTAB table values. Programming statements process COMPUTAB table values. The BY and SUMBY statements provide BY-group processing and consolidation (roll up) tables.

Functional Summary

Table 10.1 summarizes the COMPUTAB procedure statements and options.

Table 10.1 COMPUTAB Functional Summary

Description	Statement	Option
Statements		
Specify BY-group processing	BY	
Specify the format for printing a particular cell	CELL	
Define columns of the report	COLUMNS	
Initialize values in the COMPUTAB data table	INIT	
Define rows of the report	ROWS	
Produce consolidation tables	SUMBY	
Data Set Options		
Specify the input data set	COMPUTAB	DATA=
Specify an output data set	COMPUTAB	OUT=
Input Options		
Specify a value to use when testing for 0	COMPUTAB	FUZZ=
Initialize the data table to missing	COMPUTAB	INITMISS
Prevent the transposition of the input data set	COMPUTAB	NOTRANS
Printing Control Options		
Suppress printing of the listed columns	COLUMNS	NOPRINT
Suppress all printed output	COMPUTAB	NOPRINT
Suppress printing of the listed rows	ROWS	NOPRINT

Table 10.1 *continued*

Description	Statement	Option
Suppress columns with all 0 or missing values	COLUMNS	NOZERO
Suppress rows with all 0 or missing values	ROWS	NOZERO
List option values	COMPUTAB	OPTIONS
Overprint titles, values, overlining, and underlining associated with listed rows	ROWS	OVERPRINT
Print only consolidation tables	COMPUTAB	SUMONLY
Report Formatting Options		
Specify number of decimal places to print	COMPUTAB	CDEC=
Specify number of spaces between columns	COMPUTAB	CSPACE=
Specify column width for the report	COMPUTAB	CWIDTH=
Overline the listed rows with double lines	ROWS	DOL
Underline the listed rows with double lines	ROWS	DUL
Specify a format for printing the cell values	CELL	FORMAT=
Specify a format for printing column values	COLUMNS	FORMAT=
Specify a format for printing the row values	ROWS	FORMAT=
Left-align the column headings	COLUMNS	LJC
Left-justify character rows in each column	ROWS	LJC
Specify indention from the margin	ROWS	+n
Suppress printing of row titles on later pages	COMPUTAB	NORTR
Overline the listed rows with a single line	ROWS	OL
Start a new page before printing the listed rows	ROWS	_PAGE_
Specify number of spaces before row titles	COMPUTAB	RTS=
Print a blank row	ROWS	SKIP
Underline the listed rows with a single line	ROWS	UL
Specify text to print if column is 0 or missing	COLUMNS	ZERO=
Specify text to print if row is 0 or missing	ROWS	ZERO=
Row and Column Type Options		
Specify that columns contain character data	COLUMNS	CHAR
Specify that rows contain character data	ROWS	CHAR
Options for Column Headings		
Specify literal column headings	COLUMNS	‘column heading’
Use variable labels in column headings	COLUMNS	_LABEL_
Specify a master title centered over columns	COLUMNS	MTITLE=
Use column names in column headings	COLUMNS	_NAME_
Options for Row Titling		
Use labels in row titles	ROWS	_LABEL_
Use row names in row titles	ROWS	_NAME_
Specify literal row titles	ROWS	‘row title’

The following sections describe the PROC COMPUTAB statement and then describe the other statements in

alphabetical order.

PROC COMPUTAB Statement

PROC COMPUTAB *options* **;**

The following options can be used in the PROC COMPUTAB statement.

Input Options

DATA=*SAS-data-set*
names the SAS data set that contains the input data. If this option is not specified, the last created data set is used. If you are not reading a data set, use DATA=_NULL_.

FUZZ=*value*
specifies the criterion to use when testing for 0. If a number is within the FUZZ= value of 0, the number is set to 0.

INITMISS
initializes the COMPUTAB data table to missing rather than to 0. The COMPUTAB data table is discussed further in the section "Details: COMPUTAB Procedure" on page 483.

NOTRANSPOSE
NOTRANS
prevents the transposition of the input data set in building the COMPUTAB report tables. The NOTRANS option causes input data set variables to appear among the columns of the report rather than among the rows.

Report Formatting Options

The formatting options specify default values. Many of the formatting options can be modified for specific columns in COLUMNS statements and for rows in ROWS statements.

CDEC=*d*
specifies the default number of decimal places for printing. The default is CDEC=2. See the FORMAT= option in the sections on the COLUMN, ROWS, and CELL statements later in this chapter.

CSPACE=*n*
specifies the default number of spaces to insert between columns. The value of the CSPACE= option is used as the default value for the +*n* option in the COLUMNS statement. The default is CSPACE=2.

CWIDTH=*w*
specifies a default column width for the report. The default is CWIDTH=9. The width must be in the range of 1–32.

NORTR
suppresses the printing of row titles on each page. The NORTR (no row-title repeat) option is useful to suppress row titles when report pages are to be joined together in a larger report.

RTS=*n*
specifies the default number of spaces to be inserted before row titles when row titles appear after the first printed column. The default row-title spacing is RTS=2.

Output Options

NOPRINT
suppresses all printed output. Use the NOPRINT option with the OUT= option to produce an output data set but no printed reports.

OPTIONS
lists PROC COMPUTAB option values. The option values appear on a separate page preceding the procedure's normal output.

OUT=*SAS-data-set*
names the SAS data set to contain the output data. For a description of the structure of the output data set, see the section "Details: COMPUTAB Procedure" on page 483.

SUMONLY
suppresses printing of detailed reports. When the SUMONLY option is used, PROC COMPUTAB generates and prints only consolidation tables as specified in the SUMBY statement.

BY Statement

BY *variables* **;**

A BY statement can be used with PROC COMPUTAB to obtain separate reports for observations in groups defined by the BY variables. At the beginning of each BY group, before PROC COMPUTAB reads any observations, all table values are set to 0 unless the INITMISS option or an INIT statement is specified.

CELL Statement

CELL *cell-names* / **FORMAT=***format* **;**

The CELL statement specifies the format for printing a particular cell in the COMPUTAB data table. Cell variable names are compound SAS names of the form *name1.name2*, where *name1* is the name of a row variable and *name2* is the name of a column variable. Formats specified with the FORMAT= option in CELL statements override formats specified in ROWS and COLUMNS statements.

COLUMNS Statement

COLUMNS *column-list / options* **;**

COLUMNS statements define the columns of the report. The COLUMNS statement can be abbreviated COLUMN, COLS, or COL.

The specified column names must be valid SAS names. Abbreviated lists, as described in *SAS Language: Reference*, can also be used.

You can use as many COLUMNS statements as you need. A COLUMNS statement can describe more than one column, and one column of the report can be described with several different COLUMNS statements. The order of the columns on the report is determined by the order of appearance of column names in COLUMNS statements. The first occurrence of the name determines where in the sequence of columns a particular column is located.

The following options can be used in the COLUMNS statement.

Option for Column Type

CHAR
indicates that the columns contain character data.

Options for Column Headings

You can specify as many lines of column headings as needed. If no options are specified, the column names from the COLUMNS statement are used as column headings. Any or all of the following options can be used in a column heading:

"column heading"
specifies that the characters enclosed in quotes are to be used in the column heading for the variable or variables listed in the COLUMNS statement. Each quoted string appears on a separate line of the heading.

LABEL
uses labels, if provided, in the heading for the column or columns listed in the COLUMNS statement. If a label has not been provided, the name of the column is used. For information about the LABEL statement, see *SAS Language: Reference*.

MTITLE=*"text"*
specifies that the string of characters enclosed in quotes is a master title to be centered over all the columns listed in the COLUMNS statement. The list of columns must be consecutive. Special characters ("+", "*", "=", and so forth) placed on either side of the text expand to fill the space. The MTITLE= option can be abbreviated M=.

NAME
uses column names in column headings for the columns listed in the COLUMNS statement. This option allows headings (*"text"*) and names to be combined in a heading.

Options for Column Print Control

+n

inserts *n* spaces before each column listed in the COLUMNS statement. The default spacing is given by the CSPACE= option in the PROC COMPUTAB statement.

NOPRINT

suppresses printing of columns listed in the COLUMNS statement. This option enables you to create columns to be used for intermediate calculations without having those columns printed.

NOZERO

suppresses printing of columns when all the values in a column are 0 or missing. Numbers within the FUZZ= value of 0 are treated as 0.

PAGE

starts a new page of the report before printing each of the columns in the list that follows.

TITLES

prints row titles before each column in the list. The _TITLES_ option can be abbreviated as _TITLE_.

Options for Column Formatting

Column formats override row formats for particular table cells only when the input data set is not transposed (when the NOTRANS option is specified).

FORMAT=*format*

specifies a format for printing the values of the columns listed in the COLUMNS statement. The FORMAT= option can be abbreviated F=.

LJC

left-justifies the column headings for the columns listed. By default, columns are right-justified. When the LJC (left-justify character) option is used, any character row values in the column are also left-justified rather than right-justified.

ZERO=*"text"*

substitutes *"text"* when the value in the column is 0 or missing.

INIT Statement

INIT *anchor-name [locator-name] values [locator-name values]* **;**

The INIT statement initializes values in the COMPUTAB data table at the beginning of each execution of the procedure and at the beginning of each BY group if a BY statement is present.

The INIT statement in the COMPUTAB procedure is similar in function to the RETAIN statement in the DATA step, which initializes values in the program data vector. The INIT statement can be used at any point after the variable to which it refers has been defined in COLUMNS or ROWS statements. Each INIT statement initializes one row or column. Any number of INIT statements can be used.

The first term after the keyword INIT, *anchor-name*, anchors initialization to a row or column. If *anchor-name* is a row name, then all *locator-name* values in the statement are columns of that row. If *anchor-name* is a column name, then all *locator-name* values in the statement are rows of that column.

The following terms appear in the INIT statement:

anchor-name names the row or column in which values are to be initialized. This term is required.

locator-name identifies the starting column in the row (or starting row in the column) into which values are to be placed. For example, in a table with a row SALES and a column for each month of the year, the following statement initializes values for columns JAN, FEB, and JUN:

```
init sales jan 500 feb 600 jun 800;
```

If you do not specify *locator-name* values, the first value is placed into the first row or column, the second value into the second row or column, and so on. For example, the following statement assigns 500 to column JAN, 600 to FEB, and 450 to MAR:

```
init sales 500 600 450;
```

+n specifies the number of columns in a row (or rows in a column) that are to be skipped when initializing values. For example, the following statement assigns 500 to JAN and 900 to JUL:

```
init sales jan 500 +5 900;
```

*n*value* assigns *value* to *n* columns in the row (or rows in the column). For example, both of the following statements assign 500 to columns JAN through JUN and 1000 to JUL through DEC:

```
init sales jan 6*500 jul 6*1000;
```

```
init sales 6*500 6*1000;
```

ROWS Statement

ROWS *row-list / options* **;**

ROWS statements define the rows of the report. The ROWS statement can be abbreviated ROW.

The specified row names must be valid SAS names. Abbreviated lists, as described in *SAS Language: Reference*, can also be used.

You can use as many ROWS statements as you need. A ROWS statement can describe more than one row, and one row of the report can be described with several different ROWS statements. The order of the rows in the report is determined by the order of appearance of row names in ROWS statements. The first occurrence of the name determines where the row is located.

The following options can be used in the ROWS statement.

Option for Row Type

CHAR
indicates that the rows contain character data.

Options for Row Titling

You can specify as many lines of row titles as needed. If no options are specified, the names from the ROWS statement are used as row titles. Any or all of the following options can be used in a row title:

LABEL
uses labels as row titles for the row or rows listed in the ROWS statement. If a label is not provided, the name of the row is substituted. For more information about the LABEL statement, see *SAS Language: Reference*.

NAME
uses row names in row titles for the row or rows listed in the ROWS statement.

"row title"
specifies that the string of characters enclosed in quotes is to be used in the row title for the row or rows listed in the ROWS statement. Each quoted string appears on a separate line of the heading.

Options for Row Print Control

+n
indents *n* spaces from the margin for the rows in the ROWS statement.

DOL
overlines the rows listed in the ROWS statement with double lines. Overlines are printed on the line before any row titles or data for the row.

DUL
underlines the rows listed in the ROWS statement with double lines. Underlines are printed on the line after the data for the row. A row can have both an underline and an overline option.

NOPRINT
suppresses printing of the rows listed in the ROWS statement. This option enables you to create rows to be used for intermediate calculations without having those rows printed.

NOZERO
suppresses the printing of a row when all the values are 0 or missing.

OL
overlines the rows listed in the ROWS statement with a single line. Overlines are printed on the line before any row titles or data for the row.

OVERPRINT
overprints titles, values, overlining, and underlining associated with rows listed in the ROWS statement. The OVERPRINT option can be abbreviated OVP. This option is valid only when the system option OVP is in effect. For more information about the OVP option, see *SAS Language: Reference*.

PAGE
starts a new page of the report before printing these rows.

SKIP
prints a blank line after the data lines for these rows.

UL
underlines the rows listed in the ROWS statement with a single line. Underlines are printed on the line after the data for the row. A row can have both an underline and an overline option.

Options for Row Formatting

Row formatting options take precedence over column-formatting options when the input data set is transposed. Row print width can never be wider than column width. Character values are truncated on the right.

FORMAT=*format*
specifies a format for printing the values of the rows listed in the ROWS statement. The FORMAT= option can be abbreviated as F=.

LJC
left-justifies character rows in each column.

ZERO=*"text"*
substitutes *"text"* when the value in the row is 0 or missing.

SUMBY Statement

SUMBY *variables* **;**

The SUMBY statement produces consolidation tables for variables whose names are in the SUMBY list. Only one SUMBY statement can be used.

To use a SUMBY statement, you must use a BY statement. The SUMBY and BY variables must be in the same relative order in both statements. For example:

```
by a b c;
sumby a b;
```

This SUMBY statement produces tables that consolidate over values of C within levels of B and over values of B within levels of A. Suppose A has values 1, 2; B has values 1, 2; and C has values 1, 2, 3. Table 10.2 indicates the consolidation tables produced by the SUMBY statement.

Table 10.2 Consolidation Tables Produced by the SUMBY Statement

SUMBY Consolidations	Consolidated BY Groups		
A=1, B=1	C=1	C=2	C=3
A=1, B=2	C=1	C=2	C=3
A=1	B=1, C=1 B=2, C=1	B=1, C=2 B=2, C=2	B=1, C=3 B=2, C=3
A=2, B=1	C=1	C=2	C=3
A=2, B=2	C=1	C=2	C=3
A=2	B=1, C=1 B=2, C=1	B=1, C=2 B=2, C=2	B=1, C=3 B=2, C=3

Two consolidation tables for B are produced for each value of A. The first table consolidates the three tables produced for the values of C while B is 1; the second table consolidates the three tables produced for C while B is 2.

Tables are similarly produced for values of A. Nested consolidation tables are produced for B (as described previously) for each value of A. Thus, this SUMBY statement produces a total of six consolidation tables in addition to the tables produced for each BY group.

To produce a table that consolidates the entire data set (the equivalent of using PROC COMPUTAB with neither BY nor SUMBY statements), use the special name _TOTAL_ as the first entry in the SUMBY variable list. For example:

```
sumby _total_ a b;
```

PROC COMPUTAB then produces consolidation tables for SUMBY variables as well as a consolidation table for all observations.

To produce only consolidation tables, use the SUMONLY option in the PROC COMPUTAB statement.

Programming Statements

You can use most SAS programming statements the same way you use them in the DATA step. Also, all DATA step functions can be used in the COMPUTAB procedure.

Lines written by the PUT statement are not integrated with the COMPUTAB report. PUT statement output is written to the SAS log.

The automatic variable _N_ can be used; its value is the number of observations read or the number read in the current BY group, if a BY statement is used. FIRST.*variable* and LAST.*variable* references cannot be used.

The following statements are also available in PROC COMPUTAB:

```
ABORT                    FORMAT
ARRAY                    GOTO
ATTRIB                   IF-THEN/ELSE
assignment statement     LABEL
CALL                     LINK
DELETE                   PUT
DO                       RETAIN
iterative DO             SELECT
DO UNTIL                 STOP
DO WHILE                 sum statement
END                      TITLE
FOOTNOTE
```

The programming statements can be assigned labels ROW*xxxxx*: or COL*xxxxx*: to indicate the start of a row and column block, respectively. Statements in a row block create or change values in all the columns in the specified rows. Similarly, statements in a column block create or change values in all the rows in the specified columns.

There is an implied RETURN statement before each new row or column block. Thus, the flow of execution does not leave the current row (column) block before the block repeats for all columns (rows.) Row and column variables and nonretained variables are initialized prior to each execution of the block.

The next COL*xxxxx*: label, ROW*xxxxx*: label, or the end of the PROC COMPUTAB step signals the end of a row (column) block. Column blocks and row blocks can be mixed in any order. In some cases, performing calculations in different orders can lead to different results.

For more information, see the sections “Program Flow Example” on page 483, “Order of Calculations” on page 486, and “Controlling Execution within Row and Column Blocks” on page 489.

Details: COMPUTAB Procedure

Program Flow Example

This example shows how the COMPUTAB procedure processes observations in the program working storage and the COMPUTAB data table (CDT).

Assume you have three years of figures for sales and cost of goods sold (CGS), and you want to determine total sales and cost of goods sold and calculate gross profit and the profit margin.

```
title 'Program Flow Example for PROC COMPUTAB';

data example;
   input year sales cgs;
datalines;
1988    83      52
1989   106      85
1990   120     114
;

proc computab data=example;

   columns c88 c89 c90 total;
   rows sales cgs gprofit pctmarg;

   /* calculate gross profit */
   gprofit = sales - cgs;

   /* select a column */
   c88 = year = 1988;
   c89 = year = 1989;
   c90 = year = 1990;

   /* calculate row totals for sales */
   /* and cost of goods sold */
   col: total = c88 + c89 + c90;

   /* calculate profit margin */
   row: pctmarg = gprofit / cgs * 100;
run;
```

Table 10.3 shows the CDT before any observation is read in. All the columns and rows are defined with the values initialized to 0.

Table 10.3 CDT before Any Input

	C88	C89	C90	TOTAL
SALES	0	0	0	0
CGS	0	0	0	0
GPROFIT	0	0	0	0
PCTMARG	0	0	0	0

When the first input is read in (YEAR=1988, SALES=83, and CGS=52), the input block puts the values for SALES and CGS in the C88 column since year=1988. Also the value for the gross profit for that year (GPROFIT) is calculated as indicated in the following statements:

```
gprofit = sales-cgs;
c88 = year = 1988;
c89 = year = 1989;
c90 = year = 1990;
```

Table 10.4 shows the CDT after the first observation is input.

Table 10.4 CDT after First Observation Input (C88=1)

	C88	C89	C90	TOTAL
SALES	83	0	0	0
CGS	52	0	0	0
GPROFIT	31	0	0	0
PCTMARG	0	0	0	0

Similarly, the second observation (YEAR=1989, SALES=106, CGS=85) is put in the second column, and the GPROFIT is calculated to be 21. The third observation (YEAR=1990, SALES=120, CGS=114) is put in the third column, and the GPROFIT is calculated to be 6. Table 10.5 shows the CDT after all observations are input.

Table 10.5 CDT after All Observations Input

	C88	C89	C90	TOTAL
SALES	83	106	120	0
CGS	52	85	114	0
GPROFIT	31	21	6	0
PCTMARG	0	0	0	0

After the input block is executed for each observation in the input data set, the first row or column block is processed. In this case, the column block is

```
col: total = c88 + c89 + c90;
```

The column block executes for each row, calculating the TOTAL column for each row. Table 10.6 shows the CDT after the column block has executed for the first row (TOTAL=83 + 106 + 120). The total sales for the three years is 309.

Table 10.6 CDT after Column Block Executed for First Row

	C88	C89	C90	TOTAL
SALES	83	106	120	309
CGS	52	85	114	0
GPROFIT	31	21	6	0
PCTMARG	0	0	0	0

Table 10.7 shows the CDT after the column block has executed for all rows and the values for total cost of goods sold and total gross profit have been calculated.

Table 10.7 CDT after Column Block Executed for All Rows

	C88	C89	C90	TOTAL
SALES	83	106	120	309
CGS	52	85	114	251
GPROFIT	31	21	6	58
PCTMARG	0	0	0	0

After the column block has been executed for all rows, the next block is processed. The row block is

```
row: pctmarg = gprofit / cgs * 100;
```

The row block executes for each column, calculating the PCTMARG for each year and the total (TOTAL column) for three years. Table 10.8 shows the CDT after the row block has executed for all columns.

Table 10.8 CDT after Row Block Executed for All Columns

	C88	C89	C90	TOTAL
SALES	83	106	120	309
CGS	52	85	114	251
GPROFIT	31	21	6	58
PCTMARG	59.62	24.71	5.26	23.11

Order of Calculations

The COMPUTAB procedure provides alternative programming methods for performing most calculations. New column and row values are formed by adding values from the input data set, directly or with modification, into existing columns or rows. New columns can be formed in the input block or in column blocks. New rows can be formed in the input block or in row blocks.

This example illustrates the different ways to collect totals. Table 10.9 is the total sales report for two products, SALES1 and SALES2, during the years 1988–1990. The values for SALES1 and SALES2 in columns C88, C89, and C90 come from the input data set.

Table 10.9 Total Sales Report

	C88	C89	C90	SALESTOT
SALES1	15	45	80	140
SALES2	30	40	50	120
YRTOT	45	85	130	260

The new column SALESTOT, which is the total sales for each product over three years, can be computed in several different ways:

- in the input block by selecting SALESTOT for each observation:

```
salestot = 1;
```

- in a column block:

```
coltot: salestot = c88 + c89 + c90;
```

In a similar fashion, the new row YRTOT, which is the total sales for each year, can be formed as follows:

- in the input block:

```
yrtot = sales1 + sales2;
```

- in a row block:

```
rowtot: yrtot = sales1 + sales2;
```

Performing some calculations in PROC COMPUTAB in different orders can yield different results, because many operations are not commutative. Be sure to perform calculations in the proper sequence. It might take several column and row blocks to produce the desired report values.

Notice that in the previous example, the grand total for all rows and columns is 260 and is the same whether it is calculated from row subtotals or column subtotals. It makes no difference in this case whether you compute the row block or the column block first.

However, consider the example in Table 10.10, where a new column and a new row are formed.

Table 10.10 Report Sensitive to Order of Calculations

	STORE1	STORE2	STORE3	MAX
PRODUCT1	12	13	27	27
PRODUCT2	11	15	14	15
TOTAL	23	28	41	?

The new column MAX contains the maximum value in each row, and the new row TOTAL contains the column totals. MAX is calculated in a column block:

```
col: max = max(store1,store2,store3);
```

TOTAL is calculated in a row block:

```
row: total = product1 + product2;
```

Notice that either of two values, 41 or 42, is possible for the element in column MAX and row TOTAL. If the row block is first, the value is the maximum of the column totals (41). If the column block is first, the value is the sum of the MAX values (42). Whether to compute a column block before a row block can be a critical decision.

Column Selection

The following discussion assumes that the NOTRANS option has not been specified. When NOTRANS is specified, this section applies to rows rather than columns.

If a COLUMNS statement appears in PROC COMPUTAB, a target column must be selected for the incoming observation. If there is no COLUMNS statement, a new column is added for each observation. When a COLUMNS statement is present and the selection criteria fail to designate a column, the current observation is ignored. Faulty column selection can result in columns or entire tables of 0s (or missing values if the INITMISS option is specified).

During execution of the input block, when an observation is read, its values are copied into row variables in the program data vector (PDV).

To select columns, use either the column variable names themselves or the special variable _COL_. Use the column names by setting a column variable equal to some nonzero value. The example in the section "Getting Started: COMPUTAB Procedure" on page 466 uses the logical expression COMPDIV= *value*, and the result is assigned to the corresponding column variable.

```
a = compdiv = 'A';
b = compdiv = 'B';
c = compdiv = 'C';
```

IF statements can also be used to select columns. The following statements are equivalent to the preceding example:

```
if      compdiv = 'A' then a = 1;
else if compdiv = 'B' then b = 1;
else if compdiv = 'C' then c = 1;
```

At the end of the input block for each observation, PROC COMPUTAB multiplies numeric input values by any nonzero selector values and adds the result to selected columns. Character values simply overwrite the contents already in the table. If more than one column is selected, the values are added to each of the selected columns.

Use the _COL_ variable to select a column by assigning the column number to it. The COMPUTAB procedure automatically initializes column variables and sets the _COL_ variable to 0 at the start of each execution of the input block. At the end of the input block for each observation, PROC COMPUTAB examines the value of _COL_. If the value is nonzero and within range, the row variable values are added to the CDT cells of the _COL_th column. For example:

```
title 'Column Selection Example for PROC COMPUTAB';

   data rept;
      input div sales cgs;
   datalines;
   2   106    85
   3   120   114
   1    83    52
   ;

proc computab data=rept;
   row div sales cgs;
   columns div1 div2 div3;
   _col_ = div;
run;
```

The code in this example places the first observation (DIV=2) in column 2 (DIV2), the second observation (DIV=3) in column 3 (DIV3), and the third observation (DIV=1) in column 1 (DIV1).

Controlling Execution within Row and Column Blocks

Row names, column names, and the special variables _ROW_ and _COL_ can be used to limit the execution of programming statements to selected rows or columns. A row block operates on all columns of the table for a specified row unless restricted in some way. Likewise, a column block operates on all rows for a specified column. Use column names or _COL_ in a row block to execute programming statements conditionally; use row names or _ROW_ in a column block.

For example, consider a simple column block that consists of only one statement:

```
col: total = qtr1 + qtr2 + qtr3 + qtr4;
```

This column block assigns a value to each row in the TOTAL column. As each row participates in the execution of a column block, the following changes occur:

- Its row variable in the program data vector is set to 1.
- The value of _ROW_ is the number of the participating row.
- The value from each column of the row is copied from the COMPUTAB data table to the program data vector.

To avoid calculating TOTAL on particular rows, use row names or _ROW_. For example:

```
col: if sales|cost then total = qtr1 + qtr2 + qtr3 + qtr4;
```

or

```
col: if _row_ < 3  then total = qtr1 + qtr2 + qtr3 + qtr4;
```

Row and column blocks can appear in any order, and rows and columns can be selected in each block.

Program Flow

This section describes in detail the different steps in PROC COMPUTAB execution.

Step 1: Define Report Organization and Set Up the COMPUTAB Data Table

Before the COMPUTAB procedure reads in data or executes programming statements, the columns list from the COLUMNS statements and the rows list from the ROWS statements are used to set up a matrix of all columns and rows in the report. This matrix is called the COMPUTAB data table (CDT). When you define columns and rows of the CDT, the COMPUTAB procedure also sets up corresponding variables in working storage called the program data vector (PDV) for programming statements. Data values reside in the CDT but are copied into the program data vector as they are needed for calculations.

Step 2: Select Input Data with Input Block Programming Statements

The input block copies input observations into rows or columns of the CDT. By default, observations go to columns; if the data set is not transposed (the NOTRANS option is specified), observations go to rows of the report table. The input block consists of all executable statements before any ROW*xxxxx*: or COL*xxxxx*: statement label. Use programming statements to perform calculations and select a given observation to be added into the report.

Input Block

The input block is executed once for each observation in the input data set. If there is no input data set, the input block is not executed. The program logic of the input block is as follows:

1. Determine which variables, row or column, are selector variables and which are data variables. Selector variables determine which rows or columns receive values at the end of the block. Data variables contain the values that the selected rows or columns receive. By default, column variables are selector variables and row variables are data variables. If the input data set is not transposed (the NOTRANS option is specified), the roles are reversed.
2. Initialize nonretained program variables (including selector variables) to 0 (or missing if the INITMISS option is specified). Selector variables are temporarily associated with a numeric data item supplied by the procedure. Using these variables to control row and column selection does not affect any other data values.
3. Transfer data from an observation in the data set to data variables in the PDV.
4. Execute the programming statements in the input block by using values from the PDV and storing results in the PDV.
5. Transfer data values from the PDV into the appropriate columns of the CDT. If a selector variable for a row or column has a nonmissing and nonzero value, multiply each PDV value for variables used in the report by the selector variable and add the results to the selected row or column of the CDT.

Step 3: Calculate Final Values by Using Column Blocks and Row Blocks

Column Blocks

A column block is executed once for each row of the CDT. The program logic of a column block is as follows:

1. Indicate the current row by setting the corresponding row variable in the PDV to 1 and the other row variables to missing. Assign the current row number to the special variable _ROW_.
2. Move values from the current row of the CDT to the respective column variables in the PDV.
3. Execute programming statements in the column block by using the column values in the PDV. Here new columns can be calculated and old ones adjusted.
4. Move the values back from the PDV to the current row of the CDT.

Row Blocks

A row block is executed once for each column of the CDT. The program logic of a row block is as follows:

1. Indicate the current column by setting the corresponding column variable in the PDV to 1 and the other column variables to missing. Assign the current column number to the special variable _COL_.
2. Move values from the current column of the CDT to the respective row variables in the PDV.
3. Execute programming statements in the row block by using the row values in the PDV. Here new rows can be calculated and old ones adjusted.
4. Move the values back from the PDV to the current column of the CDT.

See the section "Controlling Execution within Row and Column Blocks" on page 489.

Any number of column blocks and row blocks can be used. Each can include any number of programming statements.

The values of row variables and column variables are determined by the order in which different row-block and column-block programming statements are processed. These values can be modified throughout the COMPUTAB procedure, and final values are printed in the report.

Direct Access to Table Cells

You can insert or retrieve numeric values from specific table cells by using the special reserved name TABLE with row and column subscripts. References to the TABLE have the form

```
TABLE[ row-index, column-index ]
```

where *row-index* and *column-index* can be numbers, character literals, numeric variables, character variables, or expressions that produce a number or a name. If an index is numeric, it must be within range; if it is character, it must name a row or column.

References to TABLE elements can appear on either side of an equal sign in an assignment statement and can be used in a SAS expression.

Reserved Words

Certain words are reserved for special use by the COMPUTAB procedure, and using these words as variable names can lead to syntax errors or warnings. The reserved words are as follows:

- COLUMN
- COLUMNS
- COL
- COLS

- _COL_
- ROW
- ROWS
- _ROW_
- INIT
- _N_
- TABLE

Missing Values

Missing values for variables in programming statements are treated in the same way that missing values are treated in the DATA step; that is, missing values used in expressions propagate missing values to the result. For more information about missing values, see *SAS Language: Reference*.

Missing values in the input data are treated as follows in the COMPUTAB report table. At the end of the input block, either one or more rows or one or more columns can have been selected to receive values from the program data vector (PDV). Numeric data values from variables in the PDV are added into selected report table rows or columns. If a PDV value is missing, the values already in the selected rows or columns for that variable are unchanged by the current observation. Other values from the current observation are added to table values as usual.

OUT= Data Set

The output data set contains the following variables:

- BY variables
- a numeric variable _TYPE_
- a character variable _NAME_
- the column variables from the COMPUTAB data table

The BY variables contain values for the current BY group. For observations in the output data set from consolidation tables, the consolidated BY variables have missing values.

The special variable _TYPE_ is a numeric variable that can have one of three values: 1, 2, or 3. _TYPE_ = 1 indicates observations from the normal report table produced for each BY group; _TYPE_ = 2 indicates observations from the _TOTAL_ consolidation table; _TYPE_ = 3 indicates observations from other consolidation tables. _TYPE_ = 2 and _TYPE_ = 3 observations have one or more BY variables with missing values.

The special variable _NAME_ is a character variable of length 8 that contains the row or column name associated with the observation from the report table. If the input data set is transposed, _NAME_ contains column names; otherwise, _NAME_ contains row names.

If the input data set is transposed, the remaining variables in the output data set are row variables from the report table. They are column variables if the input data set is not transposed.

NOTRANS Option

The NOTRANS option in the PROC COMPUTAB statement prevents the transposition of the input data set. The NOTRANS option affects the input block, the precedence of row and column options, and the structure of the output data set if the OUT= option is specified.

When the input data set is transposed, input variables are among the rows of the COMPUTAB report, and observations compose columns. The reverse is true if the data set is not transposed; therefore, the input block must select rows to receive data values, and input variables are among the columns.

Variables from the input data set dominate the format specification and data type. When the input data set is transposed, input variables are among the rows of the report, and row options take precedence over column options. When the input data set is not transposed, input variables are among the columns, and column options take precedence over row options.

Variables for the output data set are taken from the dimension (row or column) that contains variables from the input data set. When the input data set is transposed, this dimension is the row dimension; otherwise, the output variables come from the column dimension.

Examples: COMPUTAB Procedure

Example 10.1: Using Programming Statements

This example illustrates two ways of operating on the same input variables and producing the same tabular report. To simplify the example, no report enhancements are shown.

The manager of a hotel chain wants a report that shows the number of bookings at its hotels in each of four cities, the total number of bookings in the current quarter, and the percentage of the total coming from each location for each quarter of the year. Input observations contain the following variables: REPTDATE (report date), LA (number of bookings in Los Angeles), ATL (number of bookings in Atlanta), CH (number of bookings in Chicago), and NY (number of bookings in New York).

The following DATA step creates the SAS data set BOOKINGS:

```
title 'Using Programming Statements in PROC COMPUTAB';

data bookings;
   input reptdate date9. la atl ch ny;
datalines;
```

```
01JAN1989 100 110 120 130
01FEB1989 140 150 160 170
01MAR1989 180 190 200 210
01APR1989 220 230 240 250
01MAY1989 260 270 280 290
01JUN1989 300 310 320 330
01JUL1989 340 350 360 370
01AUG1989 380 390 400 410
01SEP1989 420 430 440 450
01OCT1989 460 470 480 490
01NOV1989 500 510 520 530
01DEC1989 540 550 560 570
;
```

The following PROC COMPUTAB statements select columns by setting _COL_ to an appropriate value. The PCT1, PCT2, PCT3, and PCT4 columns represent the percentage contributed by each city to the total for the quarter. These statements produce Output 10.1.1.

```
proc computab data=bookings cspace=1 cwidth=6;

   columns qtr1 pct1 qtr2 pct2 qtr3 pct3 qtr4 pct4;
   columns qtr1-qtr4 / format=6.;
   columns pct1-pct4 / format=6.2;
   rows la atl ch ny total;

   /* column selection */
   _col_ = qtr( reptdate ) * 2 - 1;

   /* copy qtr column values temporarily into pct columns */
   colcopy:
      pct1 = qtr1;
      pct2 = qtr2;
      pct3 = qtr3;
      pct4 = qtr4;

   /* calculate total row for all columns */
   /* calculate percentages for all rows in pct columns only  */
   rowcalc:
      total = la + atl + ch + ny;
      if mod( _col_, 2 ) = 0 then do;
         la  = la  / total * 100;
         atl = atl / total * 100;
         ch  = ch  / total * 100;
         ny  = ny  / total * 100;
         total = 100;
         end;
run;
```

Output 10.1.1 Quarterly Report of Hotel Bookings

Using Programming Statements in PROC COMPUTAB

	QTR1	PCT1	QTR2	PCT2	QTR3	PCT3	QTR4	PCT4
LA	420	22.58	780	23.64	1140	24.05	1500	24.27
ATL	450	24.19	810	24.55	1170	24.68	1530	24.76
CH	480	25.81	840	25.45	1200	25.32	1560	25.24
NY	510	27.42	870	26.36	1230	25.95	1590	25.73
TOTAL	1860	100.00	3300	100.00	4740	100.00	6180	100.00

Using the same input data, the next set of statements shows the usefulness of arrays in allowing PROC COMPUTAB to work in two directions at once. Arrays in larger programs can both reduce the amount of program source code and simplify otherwise complex methods of referring to rows and columns. The same report as in Output 10.1.1 is produced.

```
proc computab data=bookings cspace=1 cwidth=6;

   columns qtr1 pct1 qtr2 pct2 qtr3 pct3 qtr4 pct4;
   columns qtr1-qtr4 / format=6.;
   columns pct1-pct4 / format=6.2;
   rows la atl ch ny total;

   array pct[4] pct1-pct4;
   array qt[4] qtr1-qtr4;
   array rowlist[5] la atl ch ny total;

   /* column selection */
   _col_ = qtr(reptdate) * 2 - 1;

   /* copy qtr column values temporarily into pct columns */
   colcopy:
      do i = 1 to 4;
         pct[i] = qt[i];
         end;

   /* calculate total row for all columns */
   /* calculate percentages for all rows in pct columns only */

   rowcalc:
      total = la + atl + ch + ny;
      if mod(_col_,2) = 0 then
         do i = 1 to 5;
            rowlist[i] = rowlist[i] / total * 100;
         end;
run;
```

Example 10.2: Enhancing a Report

This example shows how a report can be enhanced from a simple listing to a complex report. The simplest COMPUTAB report is a transposed listing of the data in the SAS data set INCOMREP shown in Output 10.2.1. To produce this output, nothing is specified except the PROC COMPUTAB statement and a TITLE statement.

```
title 'Enhancing a Report in PROC COMPUTAB';

data incomrep;
   length type $ 8;
   input type :$8. date :monyy7.
         sales retdis tcos selling randd
         general admin deprec other taxes;
   format date monyy7.;
datalines;
BUDGET JAN1989 4600 300 2200 480 110 500 210 14 -8 510
BUDGET FEB1989 4700 330 2300 500 110 500 200 14  0 480
BUDGET MAR1989 4800 360 2600 500 120 600 250 15  2 520
ACTUAL JAN1989 4900 505 2100 430 130 410 200 14 -8 500
ACTUAL FEB1989 5100 480 2400 510 110 390 230 15  2 490
;

title 'Computab Report without Any Specifications';
proc computab data=incomrep;
run;
```

Output 10.2.1 Simple Report

Computab Report without Any Specifications

	COL1	COL2	COL3	COL4	COL5
type	BUDGET	BUDGET	BUDGET	ACTUAL	ACTUAL
date	JAN1989	FEB1989	MAR1989	JAN1989	FEB1989
sales	4600.00	4700.00	4800.00	4900.00	5100.00
retdis	300.00	330.00	360.00	505.00	480.00
tcos	2200.00	2300.00	2600.00	2100.00	2400.00
selling	480.00	500.00	500.00	430.00	510.00
randd	110.00	110.00	120.00	130.00	110.00
general	500.00	500.00	600.00	410.00	390.00
admin	210.00	200.00	250.00	200.00	230.00
deprec	14.00	14.00	15.00	14.00	15.00
other	-8.00	0.00	2.00	-8.00	2.00
taxes	510.00	480.00	520.00	500.00	490.00

To exclude the budgeted values from your report, select columns for ACTUAL observations only. To remove unwanted variables, specify the variables you want in a ROWS statement.

```
title 'Column Selection by Month';

proc computab data=incomrep;
   rows sales--other;
```

```
   columns jana feba mara;
   mnth = month(date);
   if type = 'ACTUAL';
      jana = mnth = 1;
      feba = mnth = 2;
      mara = mnth = 3;
run;
```

The report is shown in Output 10.2.2.

Output 10.2.2 Report That Uses Column Selection Techniques

Column Selection by Month

	JANA	FEBA	MARA
sales	4900.00	5100.00	0.00
retdis	505.00	480.00	0.00
tcos	2100.00	2400.00	0.00
selling	430.00	510.00	0.00
randd	130.00	110.00	0.00
general	410.00	390.00	0.00
admin	200.00	230.00	0.00
deprec	14.00	15.00	0.00
other	-8.00	2.00	0.00

To complete the report, compute new rows from existing rows. This is done in a row block (although it can also be done in the input block). Add a new column (QTR1) that accumulates all the actual data. The NOZERO option suppresses the zero column for March. The output produced by these statements is shown in Output 10.2.3.

```
title 'Completing the Report';

proc computab data=incomrep;

   /* add a new column to be selected */
   /* qtr1 column will be selected several times */
   columns actual1-actual3 qtr1 / nozero;
   array collist[3] actual1-actual3;
   rows sales retdis netsales tcos grosspft selling randd general
        admin deprec operexp operinc other taxblinc taxes netincom;

   if type='ACTUAL';
   i = month(date);
   if i <= 3 then qtr1 = 1;
   collist[i]=1;

   rowcalc:
      if sales = . then return;
      netsales = sales - retdis;
      grosspft = netsales - tcos;
      operexp  = selling + randd + general + admin + deprec;
      operinc  = grosspft - operexp;
      taxblinc = operinc + other;
```

```
      netincom = taxblinc - taxes;
run;
```

Output 10.2.3 Report That Uses Techniques to Compute New Rows

Completing the Report

	ACTUAL1	ACTUAL2	QTR1
SALES	4900.00	5100.00	10000.00
RETDIS	505.00	480.00	985.00
NETSALES	4395.00	4620.00	9015.00
TCOS	2100.00	2400.00	4500.00
GROSSPFT	2295.00	2220.00	4515.00
SELLING	430.00	510.00	940.00
RANDD	130.00	110.00	240.00
GENERAL	410.00	390.00	800.00
ADMIN	200.00	230.00	430.00
DEPREC	14.00	15.00	29.00
OPEREXP	1184.00	1255.00	2439.00
OPERINC	1111.00	965.00	2076.00
OTHER	-8.00	2.00	-6.00
TAXBLINC	1103.00	967.00	2070.00
TAXES	500.00	490.00	990.00
NETINCOM	603.00	477.00	1080.00

Now that you have all the numbers calculated, add specifications to improve the report's appearance. Specify titles, row and column labels, and formats. The report produced by these statements is shown in Output 10.2.4.

```
/* now get the report to look the way you want it */
title  'Pro Forma Income Statement';
title2 'XYZ Computer Services, Inc.';
title3 'Period to Date Actual';
title4 'Amounts in Thousands';

proc computab data=incomrep;

   columns actual1-actual3 qtr1 /
            nozero f=comma7. +3 ' ';
   array collist[3] actual1-actual3;
   columns actual1 / 'Jan';
   columns actual2 / 'Feb';
   columns actual3 / 'Mar';
   columns qtr1 / 'Total' 'Qtr 1';
   rows sales    / ' '
                   'Gross Sales ';
   rows retdis   / 'Less Returns & Discounts';
   rows netsales / 'Net Sales'          +3 ol;
   rows tcos     / ' '
                   'Total Cost of Sales';
   rows grosspft / ' '
                   'Gross Profit';
   rows selling  / ' '
                   'Operating Expenses:'
                   '   Selling';
```

```
   rows randd    / '   R & D';
   rows general  / +3;
   rows admin    / '   Administrative';
   rows deprec   / '   Depreciation'       ul;
   rows operexp  / ' '                     skip;
   rows operinc  / 'Operating Income';
   rows other    / 'Other Income/-Expense' ul;
   rows taxblinc / 'Taxable Income';
   rows taxes    / 'Income Taxes'          ul;
   rows netincom / '   Net Income'         dul;

   if type = 'ACTUAL';
   i = month( date );
   collist[i] = 1;

   colcalc:
      qtr1 = actual1 + actual2 + actual3;

   rowcalc:
      if sales = . then return;
      netsales = sales - retdis;
      grosspft = netsales - tcos;
      operexp = selling + randd + general + admin + deprec;
      operinc = grosspft - operexp;
      taxblinc = operinc + other;
      netincom = taxblinc - taxes;
run;
```

Output 10.2.4 Specifying Titles, Row and Column Labels, and Formats

Pro Forma Income Statement
XYZ Computer Services, Inc.
Period to Date Actual
Amounts in Thousands

	Jan	Feb	Total Qtr 1
Gross Sales	4,900	5,100	10,000
Less Returns & Discounts	505	480	985
Net Sales	4,395	4,620	9,015
Total Cost of Sales	2,100	2,400	4,500
Gross Profit	2,295	2,220	4,515
Operating Expenses:			
Selling	430	510	940
R & D	130	110	240
GENERAL	410	390	800
Administrative	200	230	430
Depreciation	14	15	29
	1,184	1,255	2,439
Operating Income	1,111	965	2,076
Other Income/-Expense	-8	2	-6
Taxable Income	1,103	967	2,070
Income Taxes	500	490	990
Net Income	603	477	1,080

Example 10.3: Comparison of Actual and Budget

This example shows a more complex report that compares the actual data with the budgeted values. The same input data as in the previous example is used.

The report produced by these statements is shown in Output 10.3.1. The report shows the values for the current month and the year-to-date totals for budgeted amounts, actual amounts, and the actuals as a percentage of the budgeted amounts. The data have the values for January and February. Therefore, the CURMO variable (current month) in the RETAIN statement is set to 2. The values for the observations where the month of the year is 2 (February) are accumulated for the current month values. The year-to-date values are accumulated from those observations where the month of the year is less than or equal to 2 (January and February).

```
title 'Comparison of Actual Data and Budgeted Values in PROC COMPUTAB';

data incomrep;
   length type $ 8;
   input type :$8. date :monyy7.
         sales retdis tcos selling randd
         general admin deprec other taxes;
   format date monyy7.;
datalines;
BUDGET JAN1989 4600 300 2200 480 110 500 210 14 -8 510
BUDGET FEB1989 4700 330 2300 500 110 500 200 14  0 480
BUDGET MAR1989 4800 360 2600 500 120 600 250 15  2 520
ACTUAL JAN1989 4900 505 2100 430 130 410 200 14 -8 500
ACTUAL FEB1989 5100 480 2400 510 110 390 230 15  2 490
;

title  'Pro Forma Income Statement';
title2 'XYZ Computer Services, Inc.';
title3 'Budget Analysis';
title4 'Amounts in Thousands';

options linesize=96;
proc computab data=incomrep;

   columns cmbud cmact cmpct ytdbud ytdact ytdpct /
           zero=' ';
   columns cmbud--cmpct / mtitle='- Current Month: February -';
   columns ytdbud--ytdpct / mtitle='- Year To Date -';
   columns cmbud ytdbud / 'Budget' f=comma6.;
   columns cmact ytdact / 'Actual' f=comma6.;
   columns cmpct ytdpct / '%  ' f=7.2;
   columns cmbud--ytdpct / '-';
   columns ytdbud / _titles_;
   retain curmo 2; /* current month: February */
   rows sales    / ' '
                   'Gross Sales';
   rows retdis   / 'Less Returns & Discounts';
   rows netsales / 'Net Sales'            +3 ol;
   rows tcos     / ' '
                   'Total Cost of Sales';
   rows grosspft / ' '
                   'Gross Profit'         +3;
   rows selling  / ' '
                   'Operating Expenses:'
                   '   Selling';
   rows randd    / '   R & D';
   rows general  / +3;
   rows admin    / '   Administrative';
   rows deprec   / '   Depreciation'       ul;
   rows operexp  / ' ';
   rows operinc  / 'Operating Income'      ol;
   rows other    / 'Other Income/-Expense' ul;
```

```
   rows taxblinc / 'Taxable Income';
   rows taxes    / 'Income Taxes'          ul;
   rows netincom / '   Net Income'         dul;

   cmbud = type = 'BUDGET' & month(date) = curmo;
   cmact = type = 'ACTUAL' & month(date) = curmo;
   ytdbud = type = 'BUDGET' & month(date) <= curmo;
   ytdact = type = 'ACTUAL' & month(date) <= curmo;

   rowcalc:
      if cmpct | ytdpct then return;
      netsales = sales - retdis;
      grosspft = netsales - tcos;
      operexp  = selling + randd + general + admin + deprec;
      operinc  = grosspft - operexp;
      taxblinc = operinc + other;
      netincom = taxblinc - taxes;

   colpct:
      if cmbud  & cmact  then cmpct  = 100 * cmact  / cmbud;
      if ytdbud & ytdact then ytdpct = 100 * ytdact / ytdbud;
run;
```

Output 10.3.1 Report That Uses Specifications to Tailor Output

Pro Forma Income Statement
XYZ Computer Services, Inc.
Budget Analysis
Amounts in Thousands

--- Current Month: February ---				-------- Year To Date ---------		
Budget	Actual	%		Budget	Actual	%
4,700	5,100	108.51	Gross Sales	9,300	10,000	107.53
330	480	145.45	Less Returns & Discounts	630	985	156.35
4,370	4,620	105.72	Net Sales	8,670	9,015	103.98
2,300	2,400	104.35	Total Cost of Sales	4,500	4,500	100.00
2,070	2,220	107.25	Gross Profit	4,170	4,515	108.27
			Operating Expenses:			
500	510	102.00	Selling	980	940	95.92
110	110	100.00	R & D	220	240	109.09
500	390	78.00	GENERAL	1,000	800	80.00
200	230	115.00	Administrative	410	430	104.88
14	15	107.14	Depreciation	28	29	103.57
1,324	1,255	94.79		2,638	2,439	92.46
746	965	129.36	Operating Income	1,532	2,076	135.51
	2		Other Income/-Expense	-8	-6	75.00
746	967	129.62	Taxable Income	1,524	2,070	135.83
480	490	102.08	Income Taxes	990	990	100.00
266	477	179.32	Net Income	534	1,080	202.25

Example 10.4: Consolidations

This example consolidates product tables by region and region tables by corporate division. Output 10.4.1 shows the North Central and Northeast regional summaries for the Equipment division for the first quarter. Output 10.4.2 shows the profit summary for the Equipment division. Similar tables for the Publishing division are produced but not shown here.

```
title 'Consolidations in PROC COMPUTAB';

data product;
   input pcode div region month sold revenue recd cost;
datalines;
1 1 1 1 56 5600 29 2465
1 1 1 2 13 1300 30 2550
```

```
1 1 1 3 17 1700 65 5525

   ... more lines ...


proc format;
   value divfmt 1='Equipment'
                2='Publishing';
   value regfmt 1='North Central'
                2='Northeast'
                3='South'
                4='West';
run;

proc sort data=product;
   by div region pcode;
run;

title1 '     XYZ Development Corporation      ';
title2 ' Corporate Headquarters: New York, NY ';
title3 '           Profit Summary            ';
title4 '                                     ';

options linesize=96;
proc computab data=product sumonly;
   by div region pcode;
   sumby _total_ div region;

   format div    divfmt.;
   format region regfmt.;
   label  div = 'DIVISION';

   /* specify order of columns and column titles */
   columns jan feb mar qtr1 /
                    mtitle='- first quarter -' ' '  nozero;
   columns apr may jun qtr2 /
                    mtitle='- second quarter -' ' ' nozero;
   columns jul aug sep qtr3 /
                    mtitle='- third quarter -' ' '  nozero;
   columns oct nov dec qtr4 /
                    mtitle='- fourth quarter -' ' ' nozero;
   column  jan  / ' ' 'January' '=';
   column  feb  / ' ' 'February' '=';
   column  mar  / ' ' 'March' '=';
   column  qtr1 / 'Quarter' 'Summary' '=';

   column  apr  / ' ' 'April' '=' _page_;
   column  may  / ' ' 'May' '=';
   column  jun  / ' ' 'June' '=';
   column  qtr2 / 'Quarter' 'Summary' '=';

   column  jul  / ' ' 'July' '=' _page_;
   column  aug  / ' ' 'August' '=';
   column  sep  / ' ' 'September' '=';
```

```
   column  qtr3 / 'Quarter' 'Summary' '=';

   column  oct  / ' ' 'October' '=' _page_;
   column  nov  / ' ' 'November' '=';
   column  dec  / ' ' 'December' '=';
   column  qtr4 / 'Quarter' 'Summary' '=';

   /* specify order of rows and row titles */
   row     sold    / ' ' 'Number Sold' f=8.;
   row     revenue / ' ' 'Sales Revenue';
   row     recd    / ' ' 'Number Received' f=8.;
   row     cost    / ' ' 'Cost of' 'Items Received';
   row     profit  / ' ' 'Profit' 'Within Period' ol;
   row     pctmarg / ' ' 'Profit Margin' dul;

   /* select column for appropriate month */
   _col_ = month + ceil( month / 3 ) - 1;

   /* calculate quarterly summary columns */
   colcalc:
      qtr1 = jan + feb + mar;
      qtr2 = apr + may + jun;
      qtr3 = jul + aug + sep;
      qtr4 = oct + nov + dec;

   /* calculate profit rows */
    rowcalc:
       profit = revenue - cost;
       if cost > 0 then pctmarg = profit / cost * 100;
run;
```

Output 10.4.1 Summary by Regions for the Equipment Division

XYZ Development Corporation
Corporate Headquarters: New York, NY
Profit Summary

DIVISION=Equipment region=Northeast pcode=1

SUMMARY TABLE: DIVISION=Equipment region=North Central

	first quarter			
	January	February	March	Quarter Summary
Number Sold	198	223	119	540
Sales Revenue	22090.00	26830.00	14020.00	62940.00
Number Received	255	217	210	682
Cost of Items Received	24368.00	20104.00	19405.00	63877.00
Profit Within Period	-2278.00	6726.00	-5385.00	-937.00
Profit Margin	-9.35	33.46	-27.75	-1.47

Output 10.4.1 *continued*

XYZ Development Corporation
Corporate Headquarters: New York, NY
Profit Summary

DIVISION=Publishing region=North Central pcode=4

----------SUMMARY TABLE: DIVISION=Equipment region=Northeast----------

	first quarter			
	January	February	March	Quarter Summary
Number Sold	82	180	183	445
Sales Revenue	9860.00	21330.00	21060.00	52250.00
Number Received	162	67	124	353
Cost of Items Received	16374.00	6325.00	12333.00	35032.00
Profit Within Period	-6514.00	15005.00	8727.00	17218.00
Profit Margin	-39.78	237.23	70.76	49.15

Output 10.4.2 Profit Summary for the Equipment Division

XYZ Development Corporation
Corporate Headquarters: New York, NY
Profit Summary

DIVISION=Publishing region=North Central pcode=4

SUMMARY TABLE: DIVISION=Equipment

	first quarter			
	January	February	March	Quarter Summary
Number Sold	280	403	302	985
Sales Revenue	31950.00	48160.00	35080.00	115190.00
Number Received	417	284	334	1035
Cost of Items Received	40742.00	26429.00	31738.00	98909.00
Profit Within Period	-8792.00	21731.00	3342.00	16281.00
Profit Margin	-21.58	82.22	10.53	16.46

Output 10.4.3 shows the consolidation report of profit summary over both divisions and regions.

Output 10.4.3 Profit Summary

XYZ Development Corporation
Corporate Headquarters: New York, NY
Profit Summary

SUMMARY TABLE: TOTALS

	first quarter			
	January	February	March	Quarter Summary
Number Sold	590	683	627	1900
Sales Revenue	41790.00	55910.00	44800.00	142500.00
Number Received	656	673	734	2063
Cost of Items Received	46360.00	35359.00	40124.00	121843.00
Profit Within Period	-4570.00	20551.00	4676.00	20657.00
Profit Margin	-9.86	58.12	11.65	16.95

Example 10.5: Creating an Output Data Set

This example uses data and reports similar to those in Example 10.3 to illustrate the creation of an output data set.

```
title 'Creating an Output Data Set in PROC COMPUTAB';

data product;
   input pcode div region month sold revenue recd cost;
datalines;
1 1 1 1 56 5600 29 2465
1 1 1 2 13 1300 30 2550
1 1 1 3 17 1700 65 5525

   ... more lines ...


proc sort data=product out=sorted;
   by div region;
run;
```

```
/* create data set, profit */
proc computab data=sorted notrans out=profit noprint;
   by div region;
   sumby div;

   /* specify order of rows and row titles */
   row     jan feb mar qtr1;
   row     apr may jun qtr2;
   row     jul aug sep qtr3;
   row     oct nov dec qtr4;

   /* specify order of columns and column titles */
   columns sold revenue recd cost profit pctmarg;

   /* select row for appropriate month */
   _row_ = month + ceil( month / 3 ) - 1;

   /* calculate quarterly summary rows */
   rowcalc:
      qtr1 = jan + feb + mar;
      qtr2 = apr + may + jun;
      qtr3 = jul + aug + sep;
      qtr4 = oct + nov + dec;

   /* calculate profit columns */
   colcalc:
      profit = revenue - cost;
      if cost > 0 then pctmarg = profit / cost * 100;
run;

/* make a partial listing of the output data set */
options linesize=96;
proc print data=profit(obs=10) noobs;
run;
```

Because the NOTRANS option is specified, column names become variables in the data set. REGION has missing values in the output data set for observations associated with consolidation tables. The output data set PROFIT, in conjunction with the option NOPRINT, illustrates how you can use the computational features of PROC COMPUTAB for creating additional rows and columns as in a spreadsheet without producing a report. Output 10.5.1 shows a partial listing of the output data set PROFIT.

Output 10.5.1 Partial Listing of the PROFIT Data Set

Creating an Output Data Set in PROC COMPUTAB

div	region	_TYPE_	_NAME_	sold	revenue	recd	cost	PROFIT	PCTMARG
1	1	1	JAN	198	22090	255	24368	-2278	-9.348
1	1	1	FEB	223	26830	217	20104	6726	33.456
1	1	1	MAR	119	14020	210	19405	-5385	-27.751
1	1	1	QTR1	540	62940	682	63877	-937	-1.467
1	1	1	APR	82	9860	162	16374	-6514	-39.783
1	1	1	MAY	180	21330	67	6325	15005	237.233
1	1	1	JUN	183	21060	124	12333	8727	70.761
1	1	1	QTR2	445	52250	353	35032	17218	49.149
1	1	1	JUL	194	23210	99	10310	12900	125.121
1	1	1	AUG	153	17890	164	16704	1186	7.100

Example 10.6: Cash Flows

The COMPUTAB procedure can be used to model cash flows from one time period to the next. The RETAIN statement is useful for enabling a row or column to contribute one of its values to its successor. Financial functions such as IRR (internal rate of return) and NPV (net present value) can be used on PROC COMPUTAB table values to provide a more comprehensive report. The following statements produce Output 10.6.1:

```
title 'Modeling Cash Flows in PROC COMPUTAB';

data cashflow;
   input date date9. netinc depr borrow invest tax div adv ;
datalines;
30MAR1982 65 42 32 126 43 51 41
30JUN1982 68 47 32 144 45 54 46
30SEP1982 70 49 30 148 46 55 47
30DEC1982 73 49 30 148 48 55 47
;

title1 'Blue Sky Endeavors';
title2 'Financial Summary';
title4 '(Dollar Figures in Thousands)';

proc computab data=cashflow;

   cols qtr1 qtr2 qtr3 qtr4 / 'Quarter' f=7.1;
   col  qtr1 / 'One';
   col  qtr2 / 'Two';
   col  qtr3 / 'Three';
   col  qtr4 / 'Four';
   row  begcash / 'Beginning Cash';
   row  netinc  / 'Income' '   Net income';
   row  depr    / 'Depreciation';
```

```
   row  borrow;
   row  subtot1 / 'Subtotal';
   row  invest  / 'Expenditures' '   Investment';
   row  tax     / 'Taxes';
   row  div     / 'Dividend';
   row  adv     / 'Advertising';
   row  subtot2 / 'Subtotal';
   row  cashflow/  skip;
   row  irret   / 'Internal Rate' 'of Return' zero=' ';
   rows depr borrow subtot1 tax div adv subtot2 / +3;

   retain cashin -5;
   _col_ = qtr( date );

   rowblock:
      subtot1 = netinc + depr + borrow;
      subtot2 = tax + div + adv;
      begcash = cashin;
      cashflow = begcash + subtot1 - subtot2;
      irret = cashflow;
      cashin = cashflow;

   colblock:
      if begcash then cashin = qtr1;
      if irret then do;
         temp = irr( 4, cashin, qtr1, qtr2, qtr3, qtr4 );
         qtr1 = temp;
         qtr2 = 0; qtr3 = 0; qtr4 = 0;
         end;
run;
```

Output 10.6.1 Report That Uses a RETAIN Statement and the IRR Financial Function

Blue Sky Endeavors
Financial Summary

(Dollar Figures in Thousands)

	Quarter One	Quarter Two	Quarter Three	Quarter Four
Beginning Cash	-5.0	-1.0	1.0	2.0
Income				
Net income	65.0	68.0	70.0	73.0
Depreciation	42.0	47.0	49.0	49.0
BORROW	32.0	32.0	30.0	30.0
Subtotal	139.0	147.0	149.0	152.0
Expenditures				
Investment	126.0	144.0	148.0	148.0
Taxes	43.0	45.0	46.0	48.0
Dividend	51.0	54.0	55.0	55.0
Advertising	41.0	46.0	47.0	47.0
Subtotal	135.0	145.0	148.0	150.0
CASHFLOW	-1.0	1.0	2.0	4.0
Internal Rate of Return	20.9			