Overview

The VisualPQL Procedures create output in various formats such as reports, cross-tabulations or interfaces to other software such as statistical packages. Reports and other output intended for printing are written as text files. The interface files are in formats that are directly useable by other software packages.

A VisualPQL main routine can be followed by one or more procedures. The main routine specifies the data the procedures use by putting it into a procedure table using the `PERFORM PROCS` command. This copies local variables from the main routine into the procedure table. The procedures either operate on the procedure record immediately it is put into the procedure table or, if the table has to be sorted, the procedures operate after all the data has been put in the table and sorted.

The procedure specifications describe how the data in the procedure table is treated. Other than Full Report, the procedures are specified as a single VisualPQL command with numerous options. There are some common options that are the same for all procedures.

The general structure of a program that uses procedures is:

```
RETRIEVAL | PROGRAM
.
.   VisualPQL code that creates variables for the procedure
.   PERFORM PROCS
.
PQL_PROCEDURE command and options.
PQL_PROCEDURE command and options.
END RETRIEVAL | PROGRAM
```

Following is a listing and brief description of the VisualPQL Procedures available.

`BMDP SAVE FILE`	Generates a file that can be directly accessed by the `BMDP`® statistical package.
`CONDESCRIPTIVE`	Produces descriptive statistics for specified variables. The statistics available include sum, mean, minimum, maximum, standard deviation, skewness, kurtosis, variance, standard error of the mean, coefficient of variability and confidence interval.
`CSV SAVE FILE`	Generates a file that can be accessed by any package that reads Comma Separated Variable format.
`DBASE SAVE FILE`	Generates a file that can be directly accessed by `DBASE` ®.
`DESCRIPTIVE`	Produces descriptive statistics and a frequency bar chart for a specified variable. The statistics include sum, mean, minimum, maximum, standard deviation, skewness, kurtosis, variance, standard error of the mean, coefficient of variability and

confidence interval. This essentially replaces both `CONDESCRIPTIVE` and `FREQUENCIES`. The frequency chart is produced by the `SIRGRAPH` module. Output can be in HTML format suitable for viewing through a browser.

`DIF SAVE FILE`	Generates a file in the Data Interchange Format accepted by many PC packages.
`FREQUENCIES`	Produces descriptive statistics of variables with counts and percentage distributions for values of the variables. Seventeen statistics are available.
`GRAPH`	Produces file suitable for input to the `SIRGRAPH` module (only available on Windows). The file is a text file that can be transferred to a Windows based computer if necessary. A wide variety of 2D and 3D graphs can be produced.
`MINITAB SAVE FILE`	Generates a file that can be directly accessed by the `MINITAB`® statistical package.
`PLOT`	Produces a file for graphical display of line or scatter plots. Linear regression statistics are also produced. The graphic output is produced by the `SIRGRAPH` module.
`REPORT` - Quick	Produces columnar reports with a minimum of specification using keywords to specify formats, sorting, break-points, totals and subtotals.
`REPORT` - Full	Extends the VisualPQL programming language to handle complex reports with changing report formats, conditional branching, nesting and computations.
`SAS SAVE FILE`	Generates files that can be directly accessed by the `SAS`® statistical package. The schema information and data are output in `SAS` text export format.
`SAVE TABLE`	Creates tables on SIR/XS tabfiles.
`SIR SAVE FILE`	Creates a sequential copy of a SIR/XS database.
`SPREAD SHEET`	Displays data in a spreadsheet style format.
`SPSS SAVE FILE`	Generates a file that can be accessed by the `SPSS`® statistical package.
`SYSTAT SAVE FILE`	Generates a file that can be accessed by the `SYSTAT`® statistical package.
`TABULATE`	Produces cross tabulations with options for nesting categories and concatenating tables. Cells can contain various statistics in addition to counts, percentages and quantiles. Output can be in HTML format suitable for viewing through a browser.
`WRITE RECORDS`	Generates a fixed format text data file that may be in any sequence.
`XML SAVE FILE`	Generates a file in XML (eXtensible Markup Language), that can

be accessed by many other packages and applications.

Syntax

Each of the procedures is specified with a command that has keywords to specify options. Some keywords are common to all the procedure commands. Keywords and associated options are continuations of the command. Continuation lines can be used to continue a command across multiple lines.

Except for the Full Report procedure, the actions of the procedure are completely specified by the keywords and options on the procedure command. Full Report is a programming language and subsequent VisualPQL commands specify the processing performed.

As many procedures as necessary may be specified in a single program. Options specified on one procedure have no effect on subsequent procedures.

The end product of any procedure is a file. Specify a filename on the command. Frequencies, Condescriptive and Tabulate can append the output of one procedure to the end of the previous procedure's output. If there are multiple procedures of the same type, specify the output file on the first such procedure. If subsequent procedures of the same type have no output file specification, the output is appended to the end of the previously specified file.

All procedures can produce sorted output.

Common keywords

The following clauses are standard to all VisualPQL procedures. Where there are any exceptions, these are noted:

```
FILENAME  = filename | CONSOL | STDOUT
VARIABLES = variable list
SORT      = [ (n) ] variable [(A)|(D)], ...
BOOLEAN   = (logical_expression )
SAMPLE    = fraction
```

`FILENAME`

Specify the filename to be created by the procedure. Enclose filenames that are not in the same format as standard SIR/XS names in quotes. Specify either `STDOUT` or `CONSOL` (*Not* in quotes) to display the result of a procedure in normal place that output is being written to. In the case of a normal interactive session this is the scrolled output buffer. `FILENAME` is a required clause except where noted.

`VARIABLES`

Specifies the procedure variables that are used by the procedure. For procedures that allow multiple variables, the order in which variables are specified is the order they appear in the output file. If `VARIABLES` is not specified, the default variables are output. (See `INCLUDE` and `EXCLUDE`.)

A variable list contains variable names or three keywords:
`ALL` Specifies that the default variables are used (taking regard to `INCLUDE/EXCLUDE`)
`AS` Specifies an alternate name for the variable. This can also be achieved simply by following the variable name with a new name in quotes e.g.

```
  VARIABLES = S(1) AS SALARY or
  VARIABLES = S(1) 'Salary‘
```

`TO` Specifies a list of selected variables A to B. The variables included in the list depend on the sequence in which variables are defined and include all variables defined between the two specified variables as well as the specified variables themselves. If summary variables are created with a `GET VARS ALL` rowm a record type, then the sequence is the same as the record.

The whole list can be enclosed in brackets (recommended) and thus the list starts and stops with brackets (). If the list is not in brackets then one of the following signifies the end of the list:
A new command is read. That is a command that starts in the first column, not simply the next clause on the same command.

	A special character is read, in particular a slash /. Other special characters may be invalid. A name is processed that is not a valid variable. This is taken to be a keyword for the command and is processed as such. If it is not valid keyword, then an error message ‘Error 4 Keyword is invalid' is generated.
	Some procedures allow certain non-standard specifications in their variable lists and these are noted in the documentation for those procedures.
`SORT`	Specifies the sequence of the output. **n** is an integer that specifies the maximum number of records to be sorted. The default for this parameter is either the number of records in the database or the value specified in the sortn parameter and need only be specified if the number of records in the procedure table is greater than the default. The procedure table is sorted by the specified variables in variable list order. A variable name followed by (**A**) or (**D**) specifies that for that variable the sort is in **A**scending order (the default) or in **D**escending order. A sort is implicit when a BREAK clause is specified in Quick Report.
`BOOLEAN`	Selects procedure table records used by the procedure. The procedure table records that match the logical expression are selected. If this option is not specified, all procedure table records are used.
`SAMPLE`	Selects a random sample of the procedure table records for use by the procedure. The **fraction** specifies the percent of records used and is specified as a positive decimal number less than or equal to 1 (one). e.g. Specify .25 to use a 25% sample. If you wish to alter the default **seed** used to initialise the random number generator, use the `SEED` option on the retrieval command.

Procedure Table

The procedure table is built during the execution of the main routine and contains a number of data records . Each record in the table contains the values of a number of variables, referred to as the *Procedure Variables*. The values in these variables are copied into the procedure table whenever the `PERFORM PROCS` command is executed.

Procedure Variables

Each VisualPQL program, retrieval and subroutine has its own local or program variables. VisualPQL can explicitly declare variables using commands such as `INTEGER, REAL, STRING, DATE` and `TIME`. In addition, local variables are implicitly declared when a value is assigned to an undeclared variable through commands such as `COMPUTE, SET` and `GET VARS`. See variables for further details.

Each variable and array has a *schema* entry that includes information about data type, valid values, missing values, variable labels and value labels. The `GET VARS` command copies schema information from the database or table into the schema for that local variable. The schema for other variables is created from the variable declaration and definition commands in the program. The VisualPQL Procedures use these schema entries as they format their output.

A routine may access database or table data and may reference an external block of variables.

Selecting Variables for the Procedure Table

By default, all of the local variables, excluding arrays, are procedure variables. The `DEFINE PROCEDURE VARIABLES` command specifies particular variables to use to build the procedure table. Using this command can result in performance improvements if the program has a large number of local variables that are not required by the procedures. This command is also used to include data from arrays or from `EXTERNAL VARIABLE BLOCK`s in the procedure table.

Conditional Use of Procedure Table Records

The `BOOLEAN` option specifies selects records from the procedure table to include in the procedure. This feature is useful when specifying multiple procedures in a single VisualPQL program where each procedure is to work with a subset of the procedure table records.

Selecting Variables from the Procedure Table

The procedures allow any of the variables in the procedure table to be accessed by name. Certain procedures output all variables if particular variables are not specified with the `VARIABLES` keyword on the procedure command. The `INCLUDE` and `EXCLUDE` commands alter the default variables used by a procedure; they do not alter the content of the procedure table, merely the default list of variables. These commands are placed in the main routine anywhere before the first procedure command. `INCLUDE` and `EXCLUDE` are mutually exclusive; specify either one or the other, not both.

If the `DEFINE PROCEDURE VARIABLES` command has excluded a particular variable from the procedure table, do not specify that variable on an `INCLUDE` or `EXCLUDE` as this has no effect.

INCLUDE

```
INCLUDE variable, .....
```

Specifies the default variables referenced by all procedures. The order of the variables determines the sequence used on the output file.

EXCLUDE

```
EXCLUDE variable, ...
```

Excludes variables from the default variable list.

Disk Space

If a large procedure table has to be sorted, a temporary file is used. Be sure that sufficient disk space is available. Use the `DEFINE PROCEDURE VARIABLES` command to include only the variables that are needed.

Saved Executables

If large programs are run frequently, consider saving the compiled, executable version of the program. See the SAVE = option on the `RETRIEVAL` and `PROGRAM` commands. The `NOPROCS` option on these commands saves the executable version of the program without the procedure specifications. This executable can be run and procedure commands appended to it. The general structure for this is:

Create the executable:

```
RETRIEVAL | PROGRAM   SAVE=member_name  NOPROCS NOEXECUTE
 PQL_COMMAND ...
 PERFORM PROCS ...
 ...
 PQL_PROC_COMMAND
 END RETRIEVAL | PROGRAM
```

Run the executable:

```
 RETRIEVAL | PROGRAM   GET = member_name
 PQL_PROC_COMMAND . . .
 PQL_PROC_COMMAND . . .
 END RETRIEVAL | PROGRAM
```

Saving the Procedure Table

Use `SAVE TABLE` to save the Procedure Table as a SIR/XS Tabfile table if necessary. Run any subsequent procedures against this subset of data rather than the full database. For example

Create the table:

```
RETRIEVAL | PROGRAM   SAVE=member_name NOPROCS NOEXECUTE
 PQL_COMMAND ...
 PERFORM PROCS ...
 ...
 SAVE TABLE MYTABFILE.MYTABLE
 END RETRIEVAL | PROGRAM
```

Use the table:

```
PROGRAM
.  PROCESS ROWS MYTABFILE.MYTABLE
.    GET VARS ALL
.    PERFORM PROCS
.  END ROW
PQL_PROC_COMMAND option
END PROGRAM
```

BMDP Save File

BMDP $^{®}$ is a statistical package produced by the Department of Biomathematics, at the University of California, Los Angeles.

The `BMDP SAVE FILE` procedure produces a BMDP system file that contains data and data dictionary (schema) information in a format that is directly useable by the BMDP programs. The file contains procedure table data, value labels, variable labels and missing value indicators.

Values are assigned to all missing values when the file is prepared for `BMDP` and original missing values are no longer accessible when the file is read by the `BMDP` programs.

The procedure produces a summary report listing the variables in the file, the name of the output file, the file code and other information.

```
BMDP SAVE FILE     FILENAME = filename
   [ VARIABLES = varlist | ALL ]
   [ SORT      = [ (n) ] variable [(A)|(D)], ...]
   [ BOOLEAN   = ( logical_expression ) ]
   [ SAMPLE    = fraction
```

`FILENAME`	Specifies the filename created by the procedure.
`VARIABLES`	Specifies the procedure variables that are written to the output file. The order that variables are specified is the order that they appear in the output file. If this option is not specified, the default variables are output.
`SORT`	Specifies the sequence of the output. **n** is an integer that specifies the maximum number of records to be sorted. The default for this parameter is either the number of records in the database or the value specified in the sortn parameter and need only be specified if the number of records in the procedure table is greater than the default. The procedure table is sorted by the specified variables in variable list order. A variable name followed by (**A**) or (**D**) specifies that for that variable the sort is in **A**scending order (the default) or in **D**escending order.
`BOOLEAN`	Specifies which procedure table records are used by the procedure. The procedure table records for which the logical expression is true are used by the procedure. If this option is not specified, all procedure table records are used.
`SAMPLE`	Specifies that a random sample of the procedure table records is

used by the procedure.
The **fraction** specifies the percent of records used and is specified as a positive decimal number less than or equal to 1 (one). .25, for example specifies that a 25% sample is used.

Example

The following produces a `BMDP` system file with the variables `ID, GENDER, SALARY, EDUC` and the computed variable `AGE`:

```
RETRIEVAL
INTEGER * 1 AGE
PROCESS CASES
.  PROCESS REC EMPLOYEE
.  GET VARS ID GENDER SALARY EDUC
.  COMPUTE AGE = (TODAY(0) -BIRTHDAY) / 365
.  PERFORM PROCS
.  END REC
END CASE
BMDP SAVE FILE FILENAME = BMDP.SYS
END RETRIEVAL
```

The following report is displayed upon completion of the program.

```
BMDP SAVE FILE SYNOPSIS
-----------------------

WRITTEN TO:                 "BMDP.SYS"
CODE IS:                    SYSUSR2
NUMBER OF CASES:            20
NUMBER OF VARIABLES:        5

VARIABLES IN SAVE FILE ORDER
----------------------------

AGE   ID     GENDER    SALARY    EDUC
```

Condescriptive

`CONDESCRIPTIVE` produces statistics for numeric variables in the procedure table. It is usually used with continuous variables that can assume any value in a given range. Discrete variables are usually analysed with `FREQUENCIES`.

`CONDESCRIPTIVE` may not be used with string variables. If a string variable is specified, a warning message is issued and the variable is ignored.

```
CONDESCRIPTIVE    [FILENAME = filename ]
         [ VARIABLES = varname [AS varname] .... | ALL ]
         [ WEIGHT  = varname]
         [ SAMPLE  = fraction]
         [ BOOLEAN = (logical expression)]
         [ TITLE   = 'text' ]
         [ STATISTICS = keyword list ]
         [ NOLABELS ]
```

`FILENAME`	Specifies the filename created by the procedure. If multiple `CONDESCRIPTIVE` procedures are specified and this clause is omitted, output is appended to the end of the previous output file. If no filename is specified, the output is written to the default output file (the scrolled output buffer in an interactive session).
`VARIABLES`	Specifies the procedure table variables for which statistics are produced. The order in which they are specified is the order in which they appear in the output file. If this option is not specified the default variables are output. Specify the `AS varname` option to rename variables on output. This is particularly useful for array elements, where otherwise just the array name is used. Specify the `AS` option separately for each variable to rename: `VARIABLES = S(1) AS SALARY, S(2) AS TAX`
`WEIGHT`	Specifies the procedure variable used to weight the variables. A weight of 1 is the default. The `WEIGHT` affects all statistics except `MINIMUM` and `MAXIMUM` values.
`BOOLEAN`	Specifies which procedure table records are used by the procedure. The procedure table records for which the logical expression is true are used by the procedure. If this option is not specified, all procedure table records are used.
`SAMPLE`	Specifies that a random sample of the procedure table records are used by the procedure. The **fraction** specifies the percent of records used and is specified as a positive decimal number less than or equal to 1 (one). .25, for

example specifies that a 25% sample be used.

`TITLE`	The `TITLE` clause defines the subtitle that is printed on the `CONDESCRIPTIVE` report. The subtitle text must be enclosed in quotes. If the `TITLE` clause is not used, the subtitle line is left blank.
`STATISTICS`	The `STATISTICS` clause contains a list of keywords that name the statistics produced by `CONDESCRIPTIVE`. `ALL` is the default. The keywords are:

`ALL`	produces all the statistics. Has the same result as omitting the `STATISTICS` clause or specifying all of the keywords.
`WCOUNT`	weighted count of non-missing values
`MAX`	maximum value
`MIN`	minimum value
`MEAN`	mean value
`STDV`	standard deviation
`SKEW`	skewness
`KURT`	kurtosis
`VAR`	variance
`STDE`	standard error of the mean
`CV`	coefficient of variability
`CI`	95 percent confidence interval about the mean
`SUM`	sum

`NOLABELS`	Specifies that variable labels are not printed. By default, the `CONDESCRIPTIVE` report lists both the variable name and label.

Example

In the following simple example, the procedure table contains one record for each record type 1 record in the database. Statistics are then produced by the condescriptive procedure.

```
RETRIEVAL
PROCESS CASES
.  PROCESS REC 1
.    GET VARS SALARY
.    PERFORM PROCS
.  END REC
END CASE
CONDESCRIPTIVE VARIABLES =SALARY /
               FILENAME  = COND.TXT
END RETRIEVAL
```

The output is:

```
SALARY    Current monthly salary

WGT CNT            20.000            MAXIMUM                  4000.000
MINIMUM          1650.000            MEAN                     2780.000
STD               573.631            SKEWNESS                     .217
KURTOS              -.161            VARIANCE               329052.632
STD ERR           128.268            C.V. PCT                   20.634
.95 C.I.         2511.532            TO                       3048.468
SUM             55600.000

VALID OBSERVATIONS                           20
MISSING OBSERVATIONS                         0
```

CSV Save File

The `CSV SAVE FILE` procedure creates a file that can be used to transfer data to a software package that recognises Comma Separated Variables format.

```
     CSV SAVE FILE [ FILENAME = ldi ]
                   [ VARIABLES = { varname [( ['header_text'] [format]
)]... | ALL }]
                   [ BOOLEAN = ( logical_expression ) ]
                   [ HEADER]
                   [ SAMPLE = fraction ]
                   [ SEPARATOR = "single_character" | TAB ]
                   [ SORT = [(n)]variable [(A)|(D)], ...]
```

`FILENAME` Specifies the filename created by the procedure. If no filename is specified a file named `sirproc.csv` is produced.

`VARIABLES` Specifies the procedure variables that are written to the output file. Specify the variables in the order in which they are to appear in the output file. If this option is not specified or the keyword `ALL` is specified, the default variables are output.

When specific variables are output, header text and/or a format can be specified following the variable name within parentheses.

String variables, categorical variables and value labels are output within double quotes - "". Numeric variables, dates and times are output as per the format without quotes.

`header_text` If a header record is written, by default, the variable name is used. Specify header text in quotes to use for this column in the header.

`Format [` The default format is taken from the schema. Specify a *format expression* to alter this. The width of a column is calculated from the format expression. If the data for the column does not fit in the specified width, a numeric column is filled by **X**'s and a character column is truncated to fit. In the format expressions below, "**w**"

	is a number specifying the width in columns.
`Aw`	Specifies **A**lphanumeric string format.
`Bw`	Specifies reverse string format. If there is a column header it is written as normal and the data is written in reverse (**b**ackwards).
`Lw`	Specifies that Value **L**abels are written instead of the data value. If a value has no defined label, blanks are output.
`Iw`	Specifies **I**nteger format. Any decimal portion of the number is ignored.
`Fw.d`	Specifies **F**loating point format. Use for either floating point or scaled integers with decimal portions. "**d**" is the number of decimal places.
`Ew`	Specifies **E**xponential (scientific).

`DATE`	Specifies a date variable formatted using the date format. See date formats for a complete description of date formats. The width of the column is specified by the characters in the format. For example:
	`VARIABLES = BIRTHDAY`
	`(DATE'Wwwwww Mmm DD, YYYY')`

`TIME`	Specifies a time variable formatted using the time format. See time formats for a complete description of time formats. The width of the column is specified by the characters in the format. For example:
	`VARIABLES = TESTTIME`
	`(TIME'HH:MM:SS PP')`

`D, C, P`	D puts a dollar sign (**$**) before the numeric value. C separates thousands with the comma character. P puts a percent sign (**%**) after the numeric value. These can be specified in addition to other numeric format expressions. For example:

```
VARIABLES = MONEY (F9.2, D, C)
```

`BOOLEAN`	Specifies which procedure table records are used by the procedure. The procedure table records for which the logical expression is true are used by the procedure. If this option is not specified, all procedure table records are used.
`HEADER`	Specifies that a header is written to the output file. This contains one entry per column, entries separated by columns. Each entry is up to 32 characters in quotes. Each entry is either the name of the variable or any specified header text.
`SAMPLE`	Specifies that a random sample of the procedure table records are used by the procedure. The **fraction** specifies the percent of records used and is specified as a positive decimal number less than or equal to 1 (one). .25, for example specifies that a 25% sample be used.
`SEPARATOR`	Specifies that a character is used instead of the default comma to separate fields. Specify the character in quotes or use the keyword `TAB` to use a tab character as the separator.
`SORT`	Specifies the sequence of the output. **n** is an integer that specifies the maximum number of records to be sorted. The default for this parameter is either the number of records in the database or the value specified in the sortn parameter and need only be specified if the number of records in the procedure table is greater than the default. The procedure table is sorted by the specified variables in variable list order. A variable name followed by (**A**) or (**D**) specifies that for that variable the sort is in **A**scending order (the default) or in **D**escending order.

Examples

The following produces a `CSV` file with the variables `ID`, `SALARY`, `EDUC` and `BIRTHDAY`.

```
retrieval
process cases
.  process rec employee
.    get vars id salary educ birthday
.    perform procs
.  end rec
end case
csv save file filename = example.csv  header
end retrieval
```

When the above program is run, the following summary report is displayed:

```
CSV SAVE FILE Summary Report
------------------------------
```

```
No of data records ..... 20
No of columns .......... 4
```

The file itself contains:

```
"ID","SALARY","EDUC","BIRTHDAY"
1,2150,1,01 15 38
2,2650,4,12 07 42
3,3150,3,08 10 53
....
```

The following produces a `CSV` file with some variables to illustrate format options.

```
program
integer * 1 sex
string*10
value labels sex (1)'Male'(2)'Female'
real * 8 realnum

DATE filedate ('MM/DD/YYYY')
DATE bdate    ('MM/DD/YYYY')

.  COMPUTE filedate = today(0);bdate=today(0)
.  compute realnum=12.12345
.  compute sex=1
.  compute string = 'abcdefhij'
.  perform procs

CSV SAVE FILE FILENAME = "test.csv" /
              VARIABLES= realnum ('Salary' f8.2)
                         filedate(date"mm/dd/yyyy")
                         bdate(date"mm/dd/yyyy")
                         sex(L6) string(b8) /
              HEADER
END RETRIEVAL
```

The file itself contains:

```
"Salary","FILEDATE","BDATE","SEX","STRING"
   12.12,03/06/2002,03/06/2002,"Male  ","jihfedcb"
```

DBASE Save File

The `DBASE SAVE FILE` procedure creates a file that can be used to transfer data to a software package that recognises `DBASE`® format.

```
DBASE SAVE FILE   FILENAME = filename
         [ VARIABLES = { varname [('header_text')] ...| ALL }]
         [ SORT      = [(n)]variable [(A)|(D)], ...]
         [ BOOLEAN   = ( logical_expression ) ]
         [ SAMPLE    = fraction]
```

`FILENAME`	Specifies the filename created by the procedure.
`VARIABLES`	Specifies the procedure variables that are written to the output file. Specify the variables in the order in which they are to appear in the output file. If this option is not specified or the keyword `ALL` is specified, the default variables are output. Specify any header text in quotes in parentheses following the variable name. If header text is not specified, the variable name is used.
`SORT`	Specifies the sequence of the output. **n** is an integer that specifies the maximum number of records to be sorted. The default for this parameter is either the number of records in the database or the value specified in the sortn parameter and need only be specified if the number of records in the procedure table is greater than the default. The procedure table is sorted by the specified variables in variable list order. A variable name followed by (**A**) or (**D**) specifies that for that variable the sort is in **A**scending order (the default) or in **D**escending order.
`BOOLEAN`	Specifies which procedure table records are used by the procedure. The procedure table records for which the logical expression is true are used by the procedure. If this option is not specified, all procedure table records are used.
`SAMPLE`	Specifies that a random sample of the procedure table records are used by the procedure. The **fraction** specifies the percent of records used and is specified as a positive decimal number less than or equal to 1 (one). .25, for example specifies that a 25% sample be used.

Example 1: Using Defaults

In the following example a `DBASE` file is created containing each employee's name, social security number and birth date. The variable names are used as the default column headers. `BIRTHDAY` is converted to a string in a date format before being sent to the procedure.

```
RETRIEVAL
STRING*12 BDAY
PROCESS CASES
.  PROCESS REC EMPLOYEE
.    GET VARS NAME SSN
.    COMPUTE BDAY = DATEC(BIRTHDAY,'Mmm DD, YYYY')
.    PERFORM PROCS
.  END REC
END CASE
Dbase SAVE FILE FILENAME = DBASE.OUT /
END RETRIEVAL
```

When the above program is run, the following summary report is displayed:

```
DBASE SAVE FILE Summary Report
------------------------------

No of data records ..... 20
No of columns .......... 3
```

Example 2: Sorting and Custom Headers

In the following example, custom column headers for each variable are specified and the output file is sorted by social security number.

```
RETRIEVAL
PROCESS CASES
.  PROCESS REC EMPLOYEE
.    GET VARS NAME SSN
.    COMPUTE BDAY = DATEC(BIRTHDAY,'Mmm DD, YYYY')
.    PERFORM PROCS
.  END REC
END CASE
DBASE SAVE FILE FILENAME  = DBASE.OUT /
                SORT      = SSN /
                VARIABLES = NAME ('Employee Name')
                            SSN  ('Soc Sec Number')
                            BDAY ('Birthday')
END RETRIEVAL
```

Example 3: Dummy Columns

In the following example, two dummy columns are created. The first column called 'Dummy One' and the column between `SSN` and `BDAY` called 'Dummy Two'

```
RETRIEVAL
PROCESS CASES
.  PROCESS REC EMPLOYEE
.    GET VARS NAME SSN
.    COMPUTE BDAY = DATEC(BIRTHDAY,'Mmm DD, YYYY')
.    PERFORM PROCS
.  END REC
END CASE
```

```
DBASE SAVE FILE FILENAME  =      DBASE.OUT/
                SORT      =      SSN /
                VARIABLES =      ('Dummy One')
                            NAME ('Employee Name')
                            SSN  ('Soc SecNumber')
                                 ('Dummy Two')
                            BDAY ('Birthday')
END RETRIEVAL
```

Descriptive

The `DESCRIPTIVE` procedure produces a frequency barchart and descriptive statistics on a numeric or short string (up to eight characters) variable. Frequency counts show how many times a variable had a particular value, or had a value which fell within a range.

```
     DESCRIPTIVE  VARIABLE = var
                       [ BOOLEAN = (log_expression) ]
                       [ FILENAME = ldi ]
                       [ HTML ]
                       [ INTERVALS = (n,n+,n++,...)]
                       [ RANGE = (categories,min,max)]
                       [ SAMPLE = sample]
                       [ STRINGS   = ('n','n+','n++',...)]
                       [ SUBTITLE = 'text' ]
                       [ TITLE    = 'text' ]
                       [ WEIGHT   = varname]
```

Specify a variable name. This produces a frequency table and counts of included observations (values in the frequency table) and missing observations (missing and undefined values) plus a set of descriptive statistics.

Default frequency ranges for numeric variables are calculated from the minimum and maximum values of observations. The number of ranges is taken from the square root of the number of observations with a minimum of seven and a maximum of fifty.
There are no default ranges for string variables and these must be specified as part of the `STRING` parameter.

`BOOLEAN`	Specifies which procedure table records are used by the procedure. The procedure table records for which the logical expression is true are used by the procedure. If this option is not specified, all procedure table records are used.
`FILENAME`	Specify the filename created by the procedure. If no `FILENAME` clause is specified, the output is written to a default file `sirdesc.srg`. If the filename `CGI` is specified then output is written to the user's internet browser if the procedure is run through the CGI interface. If this filename is used and the procedure is run when the CGI interface is not active, output is written to a file `sircgi.htm`. Output written to CGI is normally in HTML format (see below).
`HTML`	Specifies that output is produced in HTML format suitable for viewing through a browser.
`INTERVALS`	Specify a set of values, in increasing order, that represent the boundaries of the ranges in the frequency table. Values below the first and beyond the last value are omitted. For example, to create

	three ranges for Salary:

```
DESCRIPTIVE  VARIABLE = SALARY /
             INTERVALS = (0,2000,3000,9999)
```

Cannot specify both `INTERVALS` and `RANGES`.

`RANGE`	Specify that the frequency table consists of n ranges with the specified lowest and highest limits. Values below the first and beyond the last value are omitted. For example, to create three ranges for Salary (0-2000,2000-4000,4000-6000):

```
DESCRIPTIVE  VARIABLE = SALARY /
             RANGE = (3,0,6000)
```

`SAMPLE`	Specifies that a random sample of the procedure table records is used by the procedure. The **fraction** specifies the percent of records used and is specified as a positive decimal number less than or equal to 1 (one). .25, for example specifies that a 25% sample be used.
`STRINGS`	Specifies that the variable to be analysed is a string (up to eight characters in length) and the set of values to use for the frequency chart. Specify a set of values, in increasing order, that represent the boundaries of the ranges in the frequency table. Values below the first and beyond the last value are omitted. Note that descriptive statistics are not produced for strings.
`SUBTITLE`	Specifies the text on the `DESCRIPTIVE` report footer. Enclose the text in quotes. If `SUBTITLE` is not used, the subtitle is "Created by SIR/XS on DATE at TIME".
`TITLE`	Specifies the text on the `DESCRIPTIVE` report title. Enclose the text in quotes. If `TITLE` is not used, the title is the variable label or name when no label.
`WEIGHT`	Specifies the procedure variable used as a weighting factor for the variables in `FREQUENCIES`. Normally, cell counts are incremented by one for each appropriate occurrence. When a weighting value is specified, the cell count is incremented by the value in the specified variable.

See examples.

Examples

Example 1

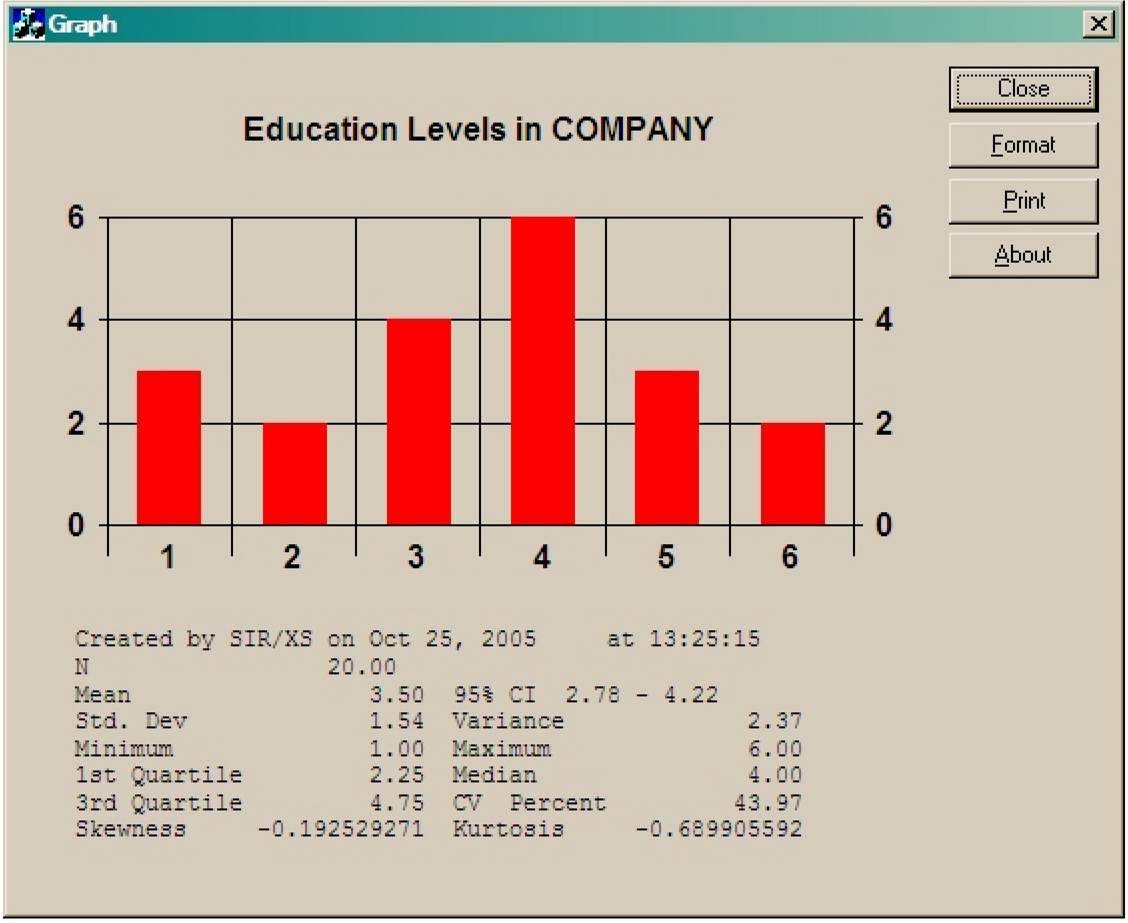
Produces a default analyses on Education Level (`EDUC`).

```
RETRIEVAL
PROCESS CASES
.  PROCESS REC EMPLOYEE
.    GET VARS EDUC
.    PERFORM PROCS
.  END REC
END CASE
DESCRIPTIVE VARIABLE   = EDUC
            FILENAME   = EDUC.SRG  /
            TITLE      = 'Education Levels in Company' /
END RETRIEVAL
```

On completion of the program, the file EDUC.SRG contains text which can be viewed with

```
ESCAPE 'sirgraph.exe educ.srg'
```

which looks something like:



```
Created by SIR/XS on Oct 25, 2005     at 13:25:15
N                 20.00
Mean                 3.50  95% CI  2.78 - 4.22
Std. Dev             1.54  Variance             2.37
Minimum              1.00  Maximum              6.00
1st Quartile         2.25  Median               4.00
3rd Quartile         4.75  CV  Percent         43.97
Skewness     -0.192529271  Kurtosis     -0.689905592
```

Example 2

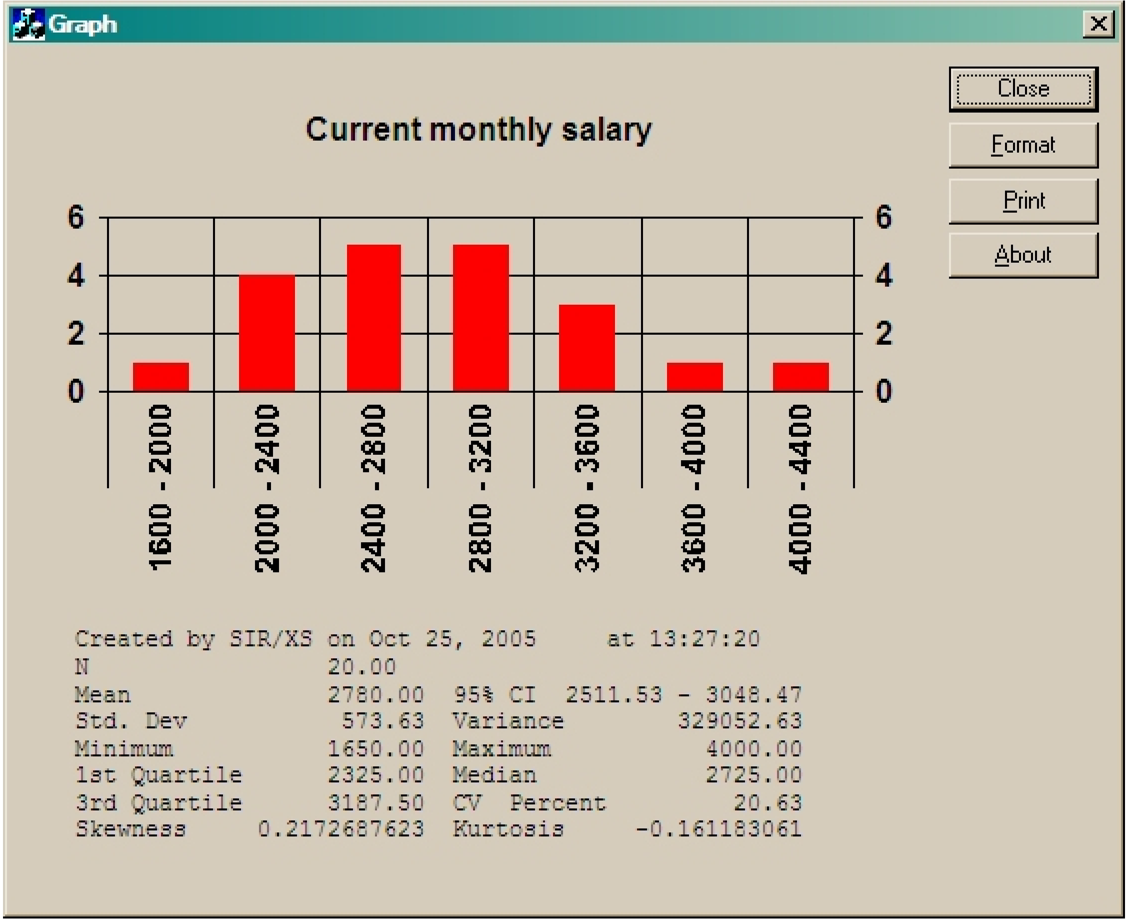
Produce a default analysis on salary.

```
RETRIEVAL
PROCESS CASES
.  PROCESS REC EMPLOYEE
.    GET VARS SALARY
.    PERFORM PROCS
.  END REC
END CASE
DESCRIPTIVE VARIABLE   = SALARY /
            FILENAME   = SALARY.SRG  /
END RETRIEVAL
```

On completion of the program, the file SALARY.SRG contains text which can be viewed with

```
ESCAPE 'sirgraph.exe salary.srg'
```

which looks something like:



```
Created by SIR/XS on Oct 25, 2005     at 13:27:20
N                   20.00
Mean              2780.00  95% CI  2511.53 - 3048.47
Std. Dev           573.63  Variance        329052.63
Minimum           1650.00  Maximum           4000.00
1st Quartile      2325.00  Median            2725.00
3rd Quartile      3187.50  CV  Percent         20.63
Skewness     0.2172687623  Kurtosis      -0.161183061
```

Example 3

Analyse a string variable (name) by letter.

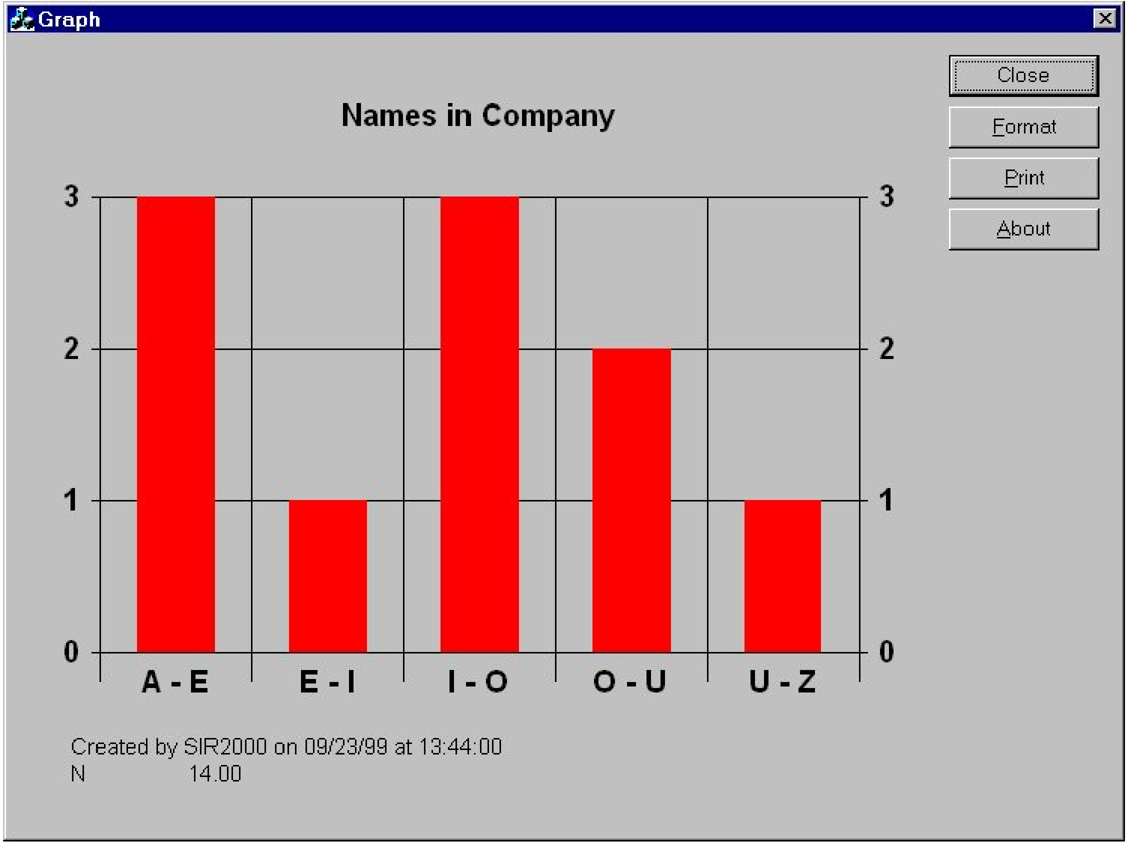
```
RETRIEVAL
STRING*4 SNAME
```

```
PROCESS CASES
.  PROCESS REC EMPLOYEE
.    COMPUTE SNAME = NAME
.    PERFORM PROCS
.  END REC
END CASE
DESCRIPTIVE VARIABLE = SNAME /
            FILENAME = NAMES.SRG  /
            STRING = ('A','E','I','O','U','Z') /
            TITLE       = 'Names in Company' /
END RETRIEVAL
```

On completion of the program, the file NAMES.SRG contains text which can be viewed with

```
ESCAPE 'sirgraph.exe names.srg'
```

which looks something like:



```
Created by SIR2000 on 09/23/99 at 13:44:00
N          14.00
```

Example 4

Produce an analysis on salary in eight $500 bands between 1000 and 5000 and produce the output as html.

```
RETRIEVAL
PROCESS CASES
.  PROCESS REC EMPLOYEE
.    GET VARS SALARY
.    PERFORM PROCS
.  END REC
END CASE
DESCRIPTIVE VARIABLE  = SALARY
         RANGE = (8,1000,5000)
         FILENAME   = SALARY.HTM   HTML
END RETRIEVAL
```

On completion of the program, the file `SALARY.HTM` contains text which can be viewed with any browser which looks something like:

Frequencies Table for Current monthly salary

Values	Frequency
1000 - 1500	1
1500 - 2000	0
2000 - 2500	3
2500 - 3000	5
3000 - 3500	2
3500 - 4000	0
4000 - 4500	1
4500 - 5000	0

Frequency Bar Chart

🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
1000 - 1500	1500 - 2000	2000 - 2500	2500 - 3000	3000 - 3500	3500 - 4000	4000 - 4500	4500 - 5000

Descriptive Statistics

Number of Observations	12.00
Number of Missing Observations	2.00
Mean	2702.83
Standard Deviation	678.84
Variance	460823.61
Minimum	1234.00
Maximum	4000.00
First Quartile	2337.50
Median	2725.00
Third Quartile	2962.50

Mean (95% C.I)	2271.52 - 3134.15
CV Percent	25.12
Skewness	-0.288310643
Kurtosis	1.7548682

DIF Save File

The `DIF SAVE FILE` procedure creates a text file in Data Interchange Format. The DIF file can be used to transfer data to a software package that recognises DIF format.

```
DIF SAVE FILE    FILENAME = filename
   [ VARIABLES = varname [('header')] ...|ALL]
   [ SORT    = [(n)]  variable [(A)|(D)] ,  ....]
   [ BOOLEAN   = ( logical_expression )]
   [ SAMPLE    = fraction]
   [ TITLE     = 'text' | NOTITLE]
   [ NOHEADING ]
```

`FILENAME`	Specifies the filename created by the procedure.
`VARIABLES`	Specifies the procedure variables that are written to the output file. The order in which they are specified is the order in which they appear in the output file. If this option is not specified, the default variables are output. Header text may be specified as text in quotes in parentheses following the variable name. If header text is not specified, the variable name is used. Dummy columns are specified as header text that does not immediately follow a variable name. The `NOHEADING` option suppresses all headers. Multiple strings in quotes in one set of parentheses specify multi-line headers.
`SORT`	Specifies the sequence of the output. **n** is an integer that specifies the maximum number of records to be sorted. The default for this parameter is either the number of records in the database or the value specified in the sortn parameter and need only be specified if the number of records in the procedure table is greater than the default. The procedure table is sorted by the specified variables in variable list order. A variable name followed by (**A**) or (**D**) specifies that for that variable the sort is in **A**scending order (the default) or in **D**escending order.
`BOOLEAN`	Specifies procedure table records used by the procedure. The procedure table records where the logical expression is true are used by the procedure. If this option is not specified, all procedure table records are used.
`SAMPLE`	Specifies that a random sample of the procedure table records are used by the procedure. The **fraction** specifies the percent of records used and is specified as a positive decimal number less than or equal to 1 (one). .25, for example specifies that a 25% sample be used.
`TITLE`	Specifies the title written to the DIF file. If `TITLE` is not specified, a default title is generated.

`NOTITLE`	Suppresses the default title.
`NOHEADING`	Suppresses generation of column headers.

Example 1: Using Defaults

The following creates a DIF file containing each employee's name, social security number and birth date using the variable names as the default column headers. `BIRTHDAY` is converted to a string in a date format before being sent to the procedure.

```
RETRIEVAL
PROCESS CASES
.  PROCESS REC EMPLOYEE
.    GET VARS NAME SSN
.    COMPUTE BDAY = DATEC(BIRTHDAY,'Mmm DD, YYYY')
.    PERFORM PROCS
.  END REC
END CASE
DIF SAVE FILE FILENAME = DIF.OUT /
END RETRIEVAL
```

When the above program is run, the following summary report is displayed:

```
DIF SAVE FILE Summary Report
----------------------------
Title ................. generated
No of header records ... 1
No of data records ..... 20
No of columns .......... 3
```

Example 2: Sorting and Using Column Headers

The following example specifies a title and custom column headers for each variable and the output is sorted by social security number.

```
RETRIEVAL
PROCESS CASES
.  PROCESS REC EMPLOYEE
.    GET VARS NAME SSN
.    COMPUTE BDAY = DATEC(BIRTHDAY,'Mmm DD, YYYY')
.    PERFORM PROCS
.  END REC
END CASE
DIF SAVE FILE FILENAME  = DIF.OUT /
              SORT      = SSN /
              TITLE     = 'Employee Listing' /
              VARIABLES = NAME ('Employee Name')
                          SSN  ('Soc Sec Number')
                          BDAY ('Birthday')
END RETRIEVAL
```

Example 3: Creating Dummy Columns

The following creates two dummy columns, the first called 'Dummy One' and the second (between SSN and BDAY), 'Dummy Two'

```
RETRIEVAL
PROCESS CASES
```

```
.  PROCESS REC EMPLOYEE
.    GET VARS NAME SSN
.    COMPUTE BDAY = DATEC(BIRTHDAY,'Mmm DD, YYYY')
.    PERFORM PROCS
.  END REC
END CASE
DIF SAVE FILE FILENAME  = DIF.OUT /
              SORT      = SSN /
              TITLE     = 'Employee Listing' /
              VARIABLES =      ('Dummy One')
                          NAME ('Employee Name')
                          SSN  ('Soc Sec Number')
                               ('Dummy Two')
                          BDAY ('Birthday')
END RETRIEVAL
```

Frequencies

The `FREQUENCIES` procedure produces frequency counts, histograms and descriptive statistics on numeric variables. Frequency counts show how many times a variable had a particular value, or had a value that fell within a specified range. There are four ways of categorising values:

`INTEGER`	Produces counts for each discrete integer value. Used for categorical variables or other codes.
`GENERAL`	Produces counts for each discrete value. The expected number of counts must be specified on the command. Used for non-integer numbers with a relatively low number of discrete values.
`CONTINUOUS`	Counts the values that fall into each range. Specify a number of ranges within an overall range so that each range is equal. Used for variables with a large range of values.
`INTERVALS`	Similar to `CONTINUOUS` except that each range is specified individually and may be unequal.

```
FREQUENCIES {INTEGER    = {varlist | ALL} (min,max)            |
             GENERAL    = {varlist | ALL} (categories)         |
             CONTINUOUS = {varlist | ALL} (categories,min,max) |
             INTERVALS  = {varlist | ALL} (intervals)}

  [ FILENAME   = filename ]
  [ STATISTICS = keywordlist ]
  [ WEIGHT     = varname ]
  [ TITLE      = 'text'  ]
  [ HISTOGRAM]
  [ ORDER ]
  [ NOLABELS  ]
  [ BOOLEAN    = (logical expression) ]
  [ SAMPLE     = fraction]
```

Specify one of `INTEGER, GENERAL, CONTINUOUS` or `INTERVALS`. This produces a frequency table with the specified number of entries. In addition a summary is printed giving counts of Included Observations (values in the frequency table), Missing Observations (missing and undefined values) and Rejected Observations (values outside the specified ranges).

`INTEGER`	Specify the numbers to count. One entry is produced for each integer value in the range.
`GENERAL`	Specify the number of discrete values to count.
`CONTINUOUS`	Specify an overall range and a number of equal sub-ranges within this. Values that fall outside of the range are omitted from the frequencies table and are not included in the calculations of the statistics. Specify the overall range by **min** and **max** . This is divided into the specified number of equal sized ranges.

`INTERVALS`

Specify a set of values, in increasing order, that represent the boundaries of the ranges in the frequency table. Values below the first and beyond the last value are omitted. For example, to create three ranges for Salary:

```
      FREQUENCIES INTERVALS =
                  SALARY (0,2000,3000,9999)
```

`FILENAME`

Specify the filename created by the procedure.
If multiple `FREQUENCIES` statements are specified and no `FILENAME` clause is specified on a second or subsequent command, the output is written to the file specified on the previous command. If no filename is specified, the output is written to the default output file (the scrolled output buffer in the case of interactive sessions).

`STATISTICS`

Specify the statistics produced. Specify one or more of the following keywords:

`ALL`	produces all the statistics.
`WCOUNT`	weighted count of non-missing values
`MAX`	maximum value
`MIN`	minimum value
`MEAN`	mean value
`STDV`	standard deviation
`SKEW`	skewness
`KURT`	kurtosis
`VAR`	variance
`STDE`	standard error of the mean
`CV`	coefficient of variability
`CI`	95% confidence interval about the mean
`SUM`	sum
`MODE`	mode value
`MED`	median value
`Q25`	first quartile
`Q50`	second quartile (same as MED)
`Q75`	third quartile

If the `STATISTICS` clause is not specified, statistics are not produced.

`TITLE`

Specifies the text on the `FREQUENCIES` report subtitle line. Enclose the text in quotes. If `TITLE` is not used, the subtitle line is left

	blank.
`HISTOGRAM`	Specifies that a histogram is printed in addition to the frequency table. A histogram is a bar chart that displays a frequency distribution the values of a variable.
`WEIGHT`	Specifies the procedure variable used as a weighting factor for the variables in `FREQUENCIES`. Normally, cell counts are incremented by one for each appropriate occurrence. When a weighting value is specified, the cell count is incremented by the value in the specified variable.
`ORDER`	Specifies that the frequency table is printed in increasing order of frequency count (the smallest frequency first), rather than in increasing order of category value that is the default.
`NOLABELS`	Specifies that variable labels are not printed. By default, the `FREQUENCIES` report lists both the variable name and label. Value labels, if they exist, are printed.
`BOOLEAN`	Specifies which procedure table records are used by the procedure. The procedure table records for which the logical expression is true are used by the procedure. If this option is not specified, all procedure table records are used.
`SAMPLE`	Specifies that a random sample of the procedure table records are used by the procedure. The **fraction** specifies the percent of records used and is specified as a positive decimal number less than or equal to 1 (one). .25, for example specifies that a 25% sample be used.

See examples.

Examples

Example 1: Integer Mode with Statistics and Title

Produces a report on Education Level (`EDUC`) that includes all statistics:

```
RETRIEVAL
PROCESS CASES
.  PROCESS REC EMPLOYEE
.    GET VARS EDUC
.    PERFORM PROCS
.  END REC
END CASE
FREQUENCIES INTEGER    = EDUC (1 , 6) /
            FILENAME   = FREQS1.TXT  /
            TITLE      = 'Education Levels in Company' /
            STATISTICS = ALL /
END RETRIEVAL
```

The file FREQS1.TXT contains the following report upon completion of the program.

```
Education Levels in Company

EDUC        Education level

VALUE LABEL                 VALUE       ABSOLUTE   RELATIVE  CUMULATIVE
                                       FREQUENCY  FREQUENCY   REL FREQ
                                                  (PERCENT)   (PERCENT)

Elementary                   1.00           3.00      15.00       15.00
High School                  2.00           2.00      10.00       25.00
Some University              3.00           4.00      20.00       45.00
B.Sc. or B.A.                4.00           6.00      30.00       75.00
M.S.                         5.00           3.00      15.00       90.00
Ph.D.                        6.00           2.00      10.00      100.00
                                      ----------    -------     -------
                            TOTAL          20.00     100.00      100.00

STATISTICS.....

WGT CNT             20.000            MAXIMUM             6.000
MINIMUM              1.000            MEAN                3.500
STD DEV              1.539            SKEWNESS           -0.193
KURTOSIS            -0.690            VARIANCE            2.368
STD ERR              0.344            C.V. PCT           43.971
.95 C.I.             2.780               TO               4.220
SUM                 70.000            MODE                4.000
MEDIAN               4.000            QUARTILE-25         2.500
QUARTILE-75          4.500


VALID OBSERVATIONS              20
MISSING OBSERVATIONS             0
REJECTED OBSERVATIONS            0
```

- -

Example 2: Continuous Mode with Selected Statistics

The following specifies six equal sized categories within the salary range of 1000 to 4000. Note that the Frequencies output does not list the range from 1000 to 1500 because no values in that range were encountered.

```
RETRIEVAL
PROCESS CASES
.  PROCESS REC EMPLOYEE
.    GET VARS SALARY
.    PERFORM PROCS
.  END REC
END CASE
FREQUENCIES CONTINUOUS = SALARY (6 , 1000 , 4000) /
            FILENAME   = FREQS2.TXT  /
            STATISTICS = MEAN MEDIAN STDV MIN MAX /
END RETRIEVAL
```

Output:

```
SALARY      Current monthly salary

LOWER             UPPER    ABSOLUTE         RELATIVE         CUMULATIVE
LIMIT             LIMIT    FREQUENCY        FREQUENCY        REL  FREQ
                                            (PERCENT)        (PERCENT)

1500.00          2000.00      1.00             5.00             5.00
2000.00          2500.00      6.00            30.00            35.00
2500.00          3000.00      6.00            30.00            65.00
3000.00          3500.00      5.00            25.00            90.00
3500.00          4000.00      2.00            10.00           100.00
                         ----------         -------          -------
                   TOTAL     20.00           100.00           100.00

STATISTICS.....

MAXIMUM           4000.000          MINIMUM       1650.000
MEAN              2780.000          STDDEV         573.631
MEDIAN            2750.000



VALID OBSERVATIONS              20
MISSING OBSERVATIONS             0
REJECTED OBSERVATIONS            0
```

Example 3: General Mode with No Statistics

Note that the output of this is identical to that of the first example, specified with the `INTEGER` clause. Given a choice of modes, `INTEGER` is slightly more efficient than `GENERAL`.

```
RETRIEVAL
PROCESS CASES
.  PROCESS REC EMPLOYEE
.    GET VARS EDUC
.    PERFORM PROCS
.  END REC
END CASE
FREQUENCIES GENERAL      = EDUC (6 ) /
            FILENAME     = FREQS3.TXT  /
            TITLE        = 'Education Levels in Company' /
END RETRIEVAL
```

Output:

```
Education Levels in Company

EDUC     Education level

VALUE LABEL       VALUE      ABSOLUTE   RELATIVE  CUMULATIVE
                             FREQUENCY  FREQUENCY   REL FREQ
                             (PERCENT)  (PERCENT)

Elementary         1.00        3.00       15.00       15.00
High School        2.00        2.00       10.00       25.00
Some University    3.00        4.00       20.00       45.00
B.Sc. or B.A.      4.00        6.00       30.00       75.00
M.S.               5.00        3.00       15.00       90.00
Ph.D.              6.00        2.00       10.00      100.00
                           ----------    -------    -------
                  TOTAL       20.00      100.00      100.00

STATISTICS.....



VALID OBSERVATIONS      20
MISSING OBSERVATIONS     0
REJECTED OBSERVATIONS    0
```

Example 4: Interval Mode for Unequal Sized Categories

```
RETRIEVAL
PROCESS CASES
.  PROCESS REC EMPLOYEE
.    GET VARS SALARY
.    PERFORM PROCS
.  END REC
END CASE
FREQUENCIES INTERVALS  =SALARY (1000,2000,3000,3500,4000) /
         FILENAME   = FREQS4.TXT  /
END RETRIEVAL
```

Output:

```
SALARY     Current monthly salary

LOWER     UPPER      ABSOLUTE      RELATIVE         CUMULATIVE
LIMIT     LIMIT     FREQUENCY      FREQUENCY        RELFREQ
                                   (PERCENT)        (PERCENT)

1000.00  2000.00        1.00         5.00              5.00
2000.00  3000.00       12.00        60.00             65.00
3000.00  3500.00        5.00        25.00             90.00
```

```
3500.00  4000.00        2.00        10.00            100.00
                    ----------     -------           -------
         TOTAL         20.00       100.00            100.00
STATISTICS....

VALID OBSERVATIONS                20
MISSING OBSERVATIONS               0
REJECTED OBSERVATIONS              0
```

Example 5: A Histogram

```
RETRIEVAL
PROCESS CASES
.  PROCESS REC EMPLOYEE
.  GET VARS EDUC
.  PERFORM PROCS
.  END REC
END CASE
FREQUENCIES INTEGER      = EDUC(1 , 6) /
            FILENAME     = FREQS5.TXT  /
            HISTOGRAM /
END RETRIEVAL
```

Output:

```
EDUC     Education level

VALUE LABEL       VALUE      ABSOLUTE    RELATIVE      CUMULATIVE
                             FREQUENCY   FREQUENCY      REL FREQ
                             (PERCENT)   (PERCENT)

Elementary         1.00         3.00       15.00           15.00
High School        2.00         2.00       10.00           25.00
Some University    3.00         4.00       20.00           45.00
B.Sc. or B.A.      4.00         6.00       30.00           75.00
M.S.               5.00         3.00       15.00           90.00
Ph.D.              6.00         2.00       10.00          100.00
                          ----------     -------         -------
                     TOTAL    20.00      100.00          100.00

EDUC     Education level

VALUE
            I
       1.00 *******************************
            I Elementary
            I (3.00)  15.00PCT
            I
       2.00 *********************
            I High School
            I (2.00)10.00 PCT
            I
       3.00 *****************************************
            I Some University
            I (4.00)  20.00 PCT
            I
       4.00 *******************************************************
            I B.Sc. or B.A.
            I (6.00)  30.00 PCT
            I
       5.00 *******************************
            I M.S.
            I (3.00)  15.00 PCT
            I
       6.00 *********************
            I Ph.D.
            I (2.00)  10.00PCT
            I
            I
            I....I....I....I....I....I....I....I....I....I....I
FREQUENCY   0   .5  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0

STATISTICS.....



VALID OBSERVATIONS        20
MISSING OBSERVATIONS       0
REJECTED OBSERVATIONS    0
```

GRAPH SAVE FILE

```
GRAPH SAVE FILE OBS = [ var| function(var)]  [ , ... ]
                ROW = var [ 'row title' ]
                          [ (i1 [,i2 [,i3 ]]) ]
                          [ 'value labels' 'xxxx' ... ]
              [ BOOLEAN = (log_expression) ]
              [ COL = var [ 'col title' ]
                          [ (i1 [,i2 [,i3 ]]) ]
                          [ 'value labels' 'xxxx' ... ] ]
              [ FILENAME = filename]
              [ MULTIPLE ]
              [ SAMPLE = sample ]
              [ SUBTITLE = 'subtitle' ]
              [ TITLE = 'title' ]
```

The `GRAPH SAVE FILE` procedure produces a text file suitable for input to the `SIRGRAPH` module. The procedure can run on any supported hardware platform but `SIRGRAPH` itself is Windows based.

The resulting graph can be two-dimensional, stacked two dimensional or three dimensional.
A two dimensional graph has an observation variable to control the magnitude of each graphed item and a row control variable with discrete values that determines the number of categories graphed.
A stacked two dimensional graph has multiple values in each category. These may be separate observation variables or may represent the different values from a second control variable.
A three dimensional graph has a row and a column control variable that determine the number of categories and an observation variable that controls the magnitude of each category.

`OBS`	This is required. Specifies the variable that is used to calculate totals, averages or other values that then translates into the magnitude on the graph e.g. the height of individual bars in the chart. The default magnitude for numeric observation variables is the total or `SUM`. The default for string observation variables is `COUNT` that adds 1 to the magnitude for each non-missing value. Specify either the name of a variable or the name of a function with the name of the variable in parentheses. e.g. `OBS=SALARY` or `OBS=COUNT(SALARY)`. The functions are:

- `AVG` - Average value (`AVERAGE` & `MEAN` are synonyms)
- `CNT` - Count of non-missing values (`COUNT` is a synonym)
- `MAX` - Maximum Value (`MAXIMUM` is a synonym)
- `MIN` - Minimum Value (`MINIMUM` is a synonym)
- `PCCOUNT` - Percentage of the column using counts
- `PCSUM` - Percentage of the column using totals
- `PRCOUNT` - Percentage of the row using counts
- `PRSUM` - Percentage of the row using totals
- `PTCOUNT` - Percentage of the total using counts
- `PTSUM` - Percentage of the total using totals
- `SUM` - Sum of values (`TOTAL` is a synonym)

Multiple observation variables can be specified that generate a stacked 2D graph (cannot specify a COL clause with multiple observation variables).

`ROW`

This is required. Specifies a row control variable. The values of this variable determine the number of rows on the graph. The variable can either have discrete values specified as part of the variable definition or you can specify the values in this clause. The variable may be numeric or a short string (up to eight characters).

The default row title is the variable label or variable name if there is no label. Specify a row title in quotes immediately after the variable name if required.

If the variable does not have defined specific values or you want to specify different categories, specify the values in parentheses.
One number, (n), specifies the number of cells that have the first n values from the variable (cannot be used for floating point variables).
Two numbers, (min,max), specify the integer values between min and max are the number of cells (can only be used for integers).
Three numbers, (min,max,interval), specify values between min and max in interval sizes e.g. (1000,6000,500) produces 10 rows. This must be specified for floating point control variables and cannot be used for strings.

The individual row values have labels. By default these are taken from the value labels or categorical string values. Specify value labels on the command to label specified category values or to use instead of the defaults. Enclose each label in quotes.

`BOOLEAN`

Specifies which procedure table records are used by the procedure. The procedure table records for which the logical expression is true are used by the procedure. If this option is not specified, all procedure table records are used.

`COL`	Specifies a column control variable. If specified, must not precede the ROW parameter. Cannot be used with multiple observation variables. The values of this variable determine the number of columns on the graph unless the MULTIPLE option is specified (see below). The variable can either have discrete values specified as part of the variable definition or you can specify the values in this clause. The variable may be numeric or a short string (up to eight characters). The default column title is the variable label or variable name if there is no label. Specify a column title in quotes immediately after the variable name if required. If the variable does not have defined specific values or you want to specify different categories, specify the values in parentheses. One number, (n), specifies the number of cells that have the first n values from the variable (cannot be used for floating point variables). Two numbers, (min,max), specify the integer values between min and max are the number of cells (can only be used for integers). Three numbers, (min,max,interval), specify values between min and max in interval sizes e.g. (1000,6000,500) produces 10 rows. This must be specified for floating point control variables and cannot be used for strings. The individual column values have labels. By default these are taken from the value labels or categorical string values. Specify value labels on the command to label specified category values or to use instead of the defaults. Enclose each label in quotes.
`FILENAME`	Specify the filename created by the procedure. If no `FILENAME` clause is specified, the output is written to a default file `sirgraph.srg`.
`MULTIPLE`	The keyword `MULTIPLE` specifies that a stacked 2D graph using the column variable is produced rather than a 3D graph.
`SAMPLE`	Specifies that a random sample of the procedure table records are used by the procedure. The **fraction** specifies the percent of records used and is specified as a positive decimal number less than or equal to 1 (one). .25, for example specifies that a 25% sample be used.
`SUBTITLE`	Specifies the heading text on the graph footer. Enclose the text in quotes. If `SUBTITLE` is not used, the subtitle is "Created by SIR/XS on DATE at TIME".
`TITLE`	Specifies the text on the graph title. Enclose the text in quotes. If

`TITLE` is not used, the title is the variable label or name when no label.

See examples.

Graph Examples

Example 1

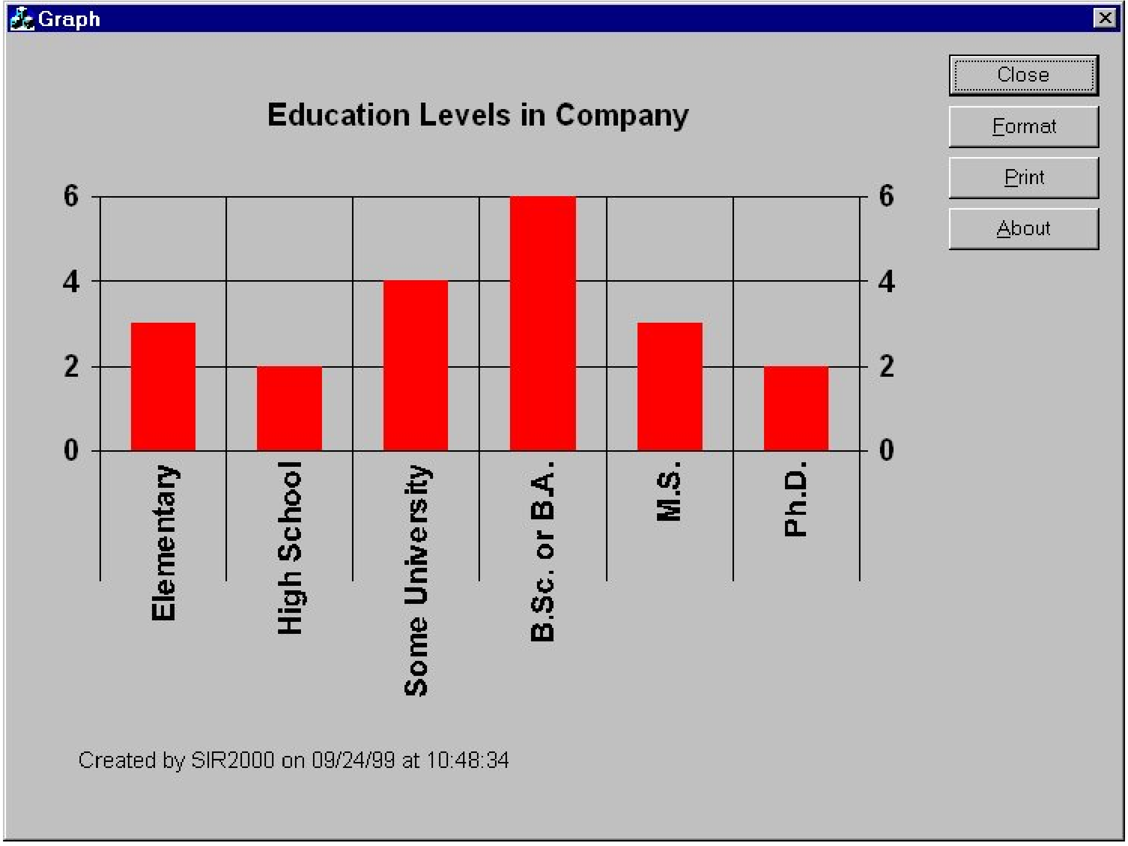
Produces an analysis of the number of employees by Education Level (`EDUC`).

```
RETRIEVAL
PROCESS CASES
.  PROCESS REC EMPLOYEE
.    GET VARS EDUC
.    PERFORM PROCS
.  END REC
END CASE
GRAPH SAVE FILE    OBS   = COUNT(EDUC) /
                   ROW   = EDUC        /
            FILENAME   = GRAPH1.SRG  /
            TITLE      = 'Education Levels in Company'
END RETRIEVAL
```

On completion of the program, the file GRAPH1.SRG contains text that can be viewed with

```
ESCAPE 'sirgraph.exe GRAPH1.srg'
```

which looks something like:



Example 2

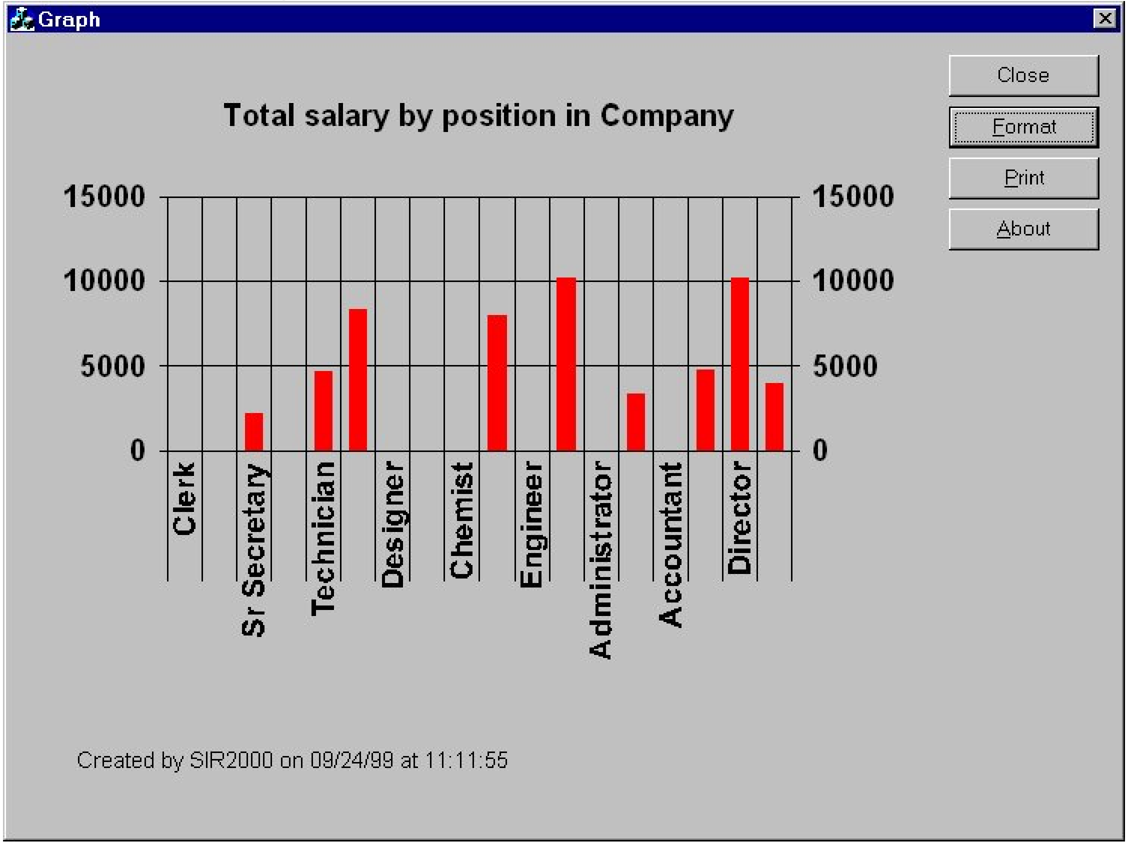
Produces an analysis of salary by current position (`CURRPOS`).

```
RETRIEVAL
PROCESS CASES
.  PROCESS REC EMPLOYEE
.    GET VARS SALARY CURRPOS
.    PERFORM PROCS
.  END REC
END CASE
GRAPH SAVE FILE   OBS  = SALARY /
                  ROW  = CURRPOS        /
            FILENAME   = GRAPH2.SRG  /
            TITLE      = 'Total salary by position in Company'
END RETRIEVAL
```

On completion of the program, the file GRAPH2.SRG contains text that can be viewed with

```
ESCAPE 'sirgraph.exe GRAPH2.srg'
```

which looks something like:



Example 3

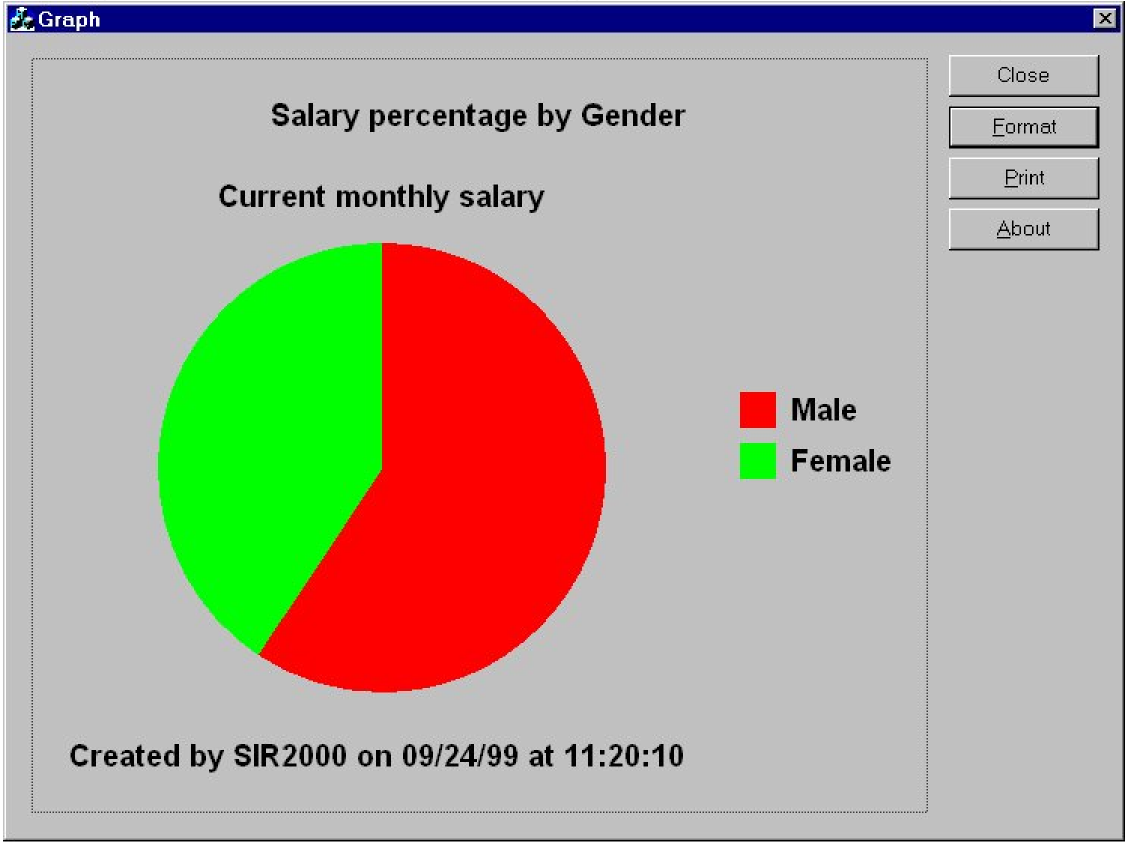
Produces an analysis of percentage of salary by gender.

```
RETRIEVAL
PROCESS CASES
.  PROCESS REC EMPLOYEE
.    GET VARS SALARY GENDER
.    PERFORM PROCS
.  END REC
END CASE
GRAPH SAVE FILE   OBS  = PTSUM(SALARY) /
                  ROW  = GENDER        /
            FILENAME   = GRAPH3.SRG  /
            TITLE      = 'Salary percentage by Gender'
END RETRIEVAL
```

On completion of the program, the file GRAPH2.SRG contains text which can be viewed with

```
ESCAPE 'sirgraph.exe GRAPH3.srg'
```

You might then use some of the options in sirgraph to produce a chart something like:


Graph
Close
Format
Print
About
Salary percentage by Gender
Current monthly salary
🟥 Male
🟩 Female
Created by SIR2000 on 09/24/99 at 11:20:10

Example 4

Produces an analysis of percentage of salary by gender by education.

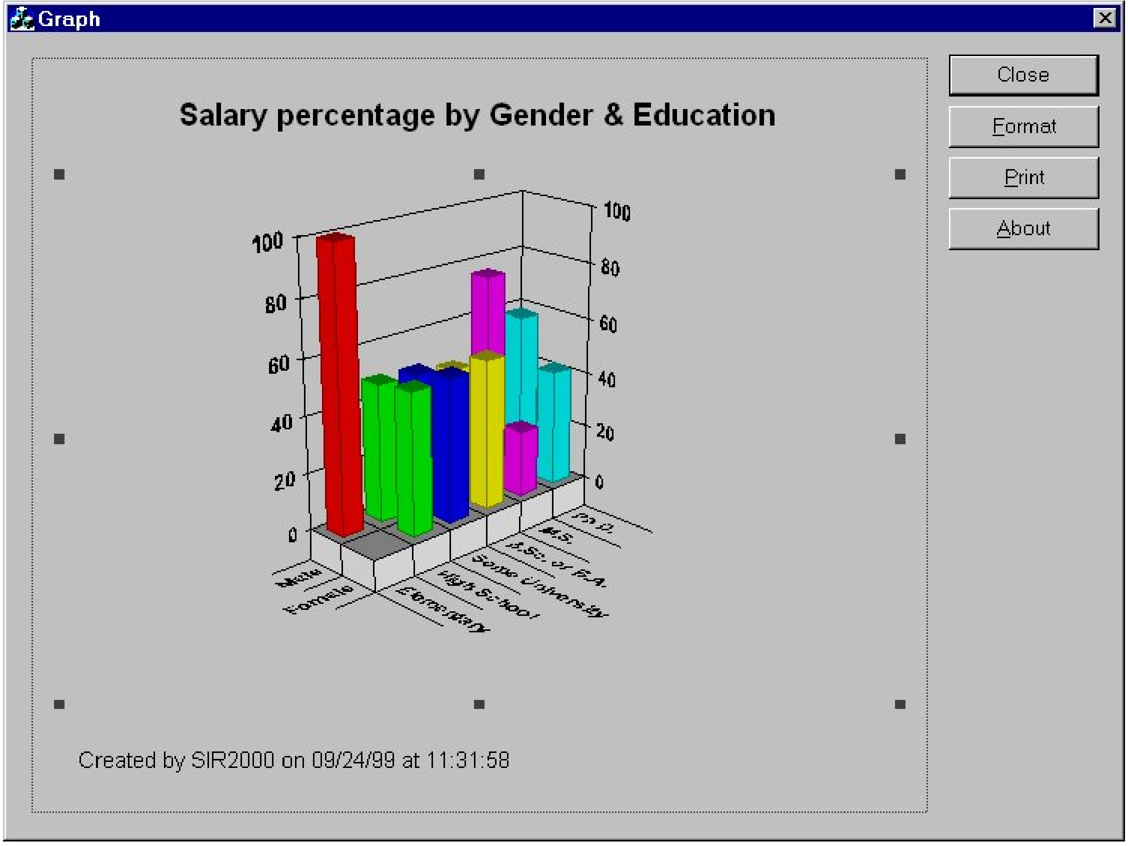
```
RETRIEVAL
PROCESS CASES
.  PROCESS REC EMPLOYEE
.    GET VARS SALARY GENDER EDUC
.    PERFORM PROCS
.  END REC
END CASE
GRAPH SAVE FILE   OBS  = PRSUM(SALARY) /
                  ROW  = GENDER        /
                  COL  = EDUC        /
            FILENAME   = GRAPH4.SRG  /
            TITLE      = 'Salary percentage by Gender & Education'
```

```
END RETRIEVAL
```

On completion of the program, the file GRAPH2.SRG contains text which can be viewed with

```
ESCAPE 'sirgraph.exe GRAPH4.srg'
```

You might then use some of the options in sirgraph to produce a chart something like:



MINITAB Save File

The `MINITAB SAVE FILE` procedure creates a text file in `MINITAB`® external format that contains data and schema information produced in a VisualPQL program. `MINITAB` is a statistical package that runs on a number of different types of computers.

```
MINITAB SAVE FILE    EXPORT = filename
    [ VARIABLES = var_list | ALL  ]
    [ SORT      = [(n)] variable [(A)|(D)], ...]
    [ BOOLEAN   = ( logical_expression )  ]
    [ SAMPLE    = fraction]
```

`EXPORT`	Specifies the filename created by the procedure.
`VARIABLES`	Specifies the procedure variables written to the output file. Specify the variables in the order they are to appear in the output file. If this option is not specified, the default variables are output.
`SORT`	Specifies the sequence of the output. **n** is an integer that specifies the maximum number of records to be sorted. The default for this parameter is either the number of records in the database or the value specified in the sortn parameter and need only be specified if the number of records in the procedure table is greater than the default. The procedure table is sorted by the specified variables in variable list order. A variable name followed by (**A**) or (**D**) specifies that for that variable the sort is in **A**scending order (the default) or in **D**escending order.
`BOOLEAN`	Specifies which procedure table records are used by the procedure. The procedure table records for which the logical expression is true are used by the procedure. If this option is not specified, all procedure table records are used.
`SAMPLE`	Specifies that a random sample of the procedure table records are used by the procedure. The **fraction** specifies the percent of records used and is specified as a positive decimal number less than or equal to 1 (one). .25, for example specifies that a 25% sample be used.

Example

The following example creates a MINITAB file with one record per employee and the variables `ID, GENDER, CURRPOS` and `SALARY`. The records are sorted by Salary in descending order.

```
RETRIEVAL
. PROCESS CASES
.   PROCESS REC EMPLOYEE
.      GET VARS ID GENDER CURRPOS SALARY
.      PERFORM PROCS
.    END REC
. END CASE
MINITAB SAVE FILE EXPORT = MINITAB.DAT /
                  SORT   = SALARY (D)
END RETRIEVAL
```

After the retrieval finishes, the following summary report is displayed.

```
MINITAB PORTABLE WORKSHEET SYNOPSIS
-----------------------------------

WRITTEN TO                'MINITAB.DAT'

NO OF ROWS                20
USER VARIABLES             5

VARIABLES IN SAVE FILE ORDER
----------------------------

ID
GENDER
CURRPOS
SALARY
```

Plot

The `PLOT` procedure generates a two dimensional table containing pairs of values which are the values in those variables for each occurrence in the procedure table. This information is written to a text file together with a set of statistics calculated from the data. This file is then used by the sirgraph module to produce `LINE` or `SCATTER` plots.

The statistics are the results of a simple linear regression of the variables. The following statistics are calculated:
The mean and standard deviation for each variable independently;
The correlation coefficient - r and r squared and the probability of significance of r;
The T-distribution and DF (which is two less than the number of observations).

```
PLOT  {LINE | SCATTER} = var , var
      [ FILENAME = ldi ]
      [ BOOLEAN = (log_expression) ]
      [ SAMPLE = sample ]
      [ SORT = [N] var [A|D], ... ]
      [ SUBTITLE = 'text' ]
      [ TITLE    = 'text' ]
      [ XTITLE   = 'text' ]
      [ YTITLE   = 'text' ]
```

`LINE | SCATTER`	Specify either `LINE` or `SCATTER`. Specify two variables to plot that must both be numeric. The first variable is the X variable and values go across the plot; the second variable is the Y variable and values go up the plot.
`FILENAME`	Specify the filename produced by the procedure. If no filename is specified, the output is written to the default filename sirplot.srg.
`BOOLEAN`	Specifies which procedure table records are used by the procedure. The procedure table records for which the logical expression is true are used by the procedure. If this option is not specified, all procedure table records are used.
`SAMPLE`	Specifies that a random sample of the procedure table records are used by the procedure. The **fraction** specifies the percent of records used and is specified as a positive decimal number less than or equal to 1 (one). .25, for example specifies that a 25% sample be used.
`TITLE`	Specifies the main title displayed at the top of the plot. Enclose the text in quotes. If `TITLE` is omitted, the title is the names of the two

	variables.
`XTITLE`	Specifies a title for the x-axis displayed at the bottom of the plot. There is no default.
`YTITLE`	Specifies a title for the y-axis displayed vertically up the left axis of the plot. There is no default.

Example Scattergram

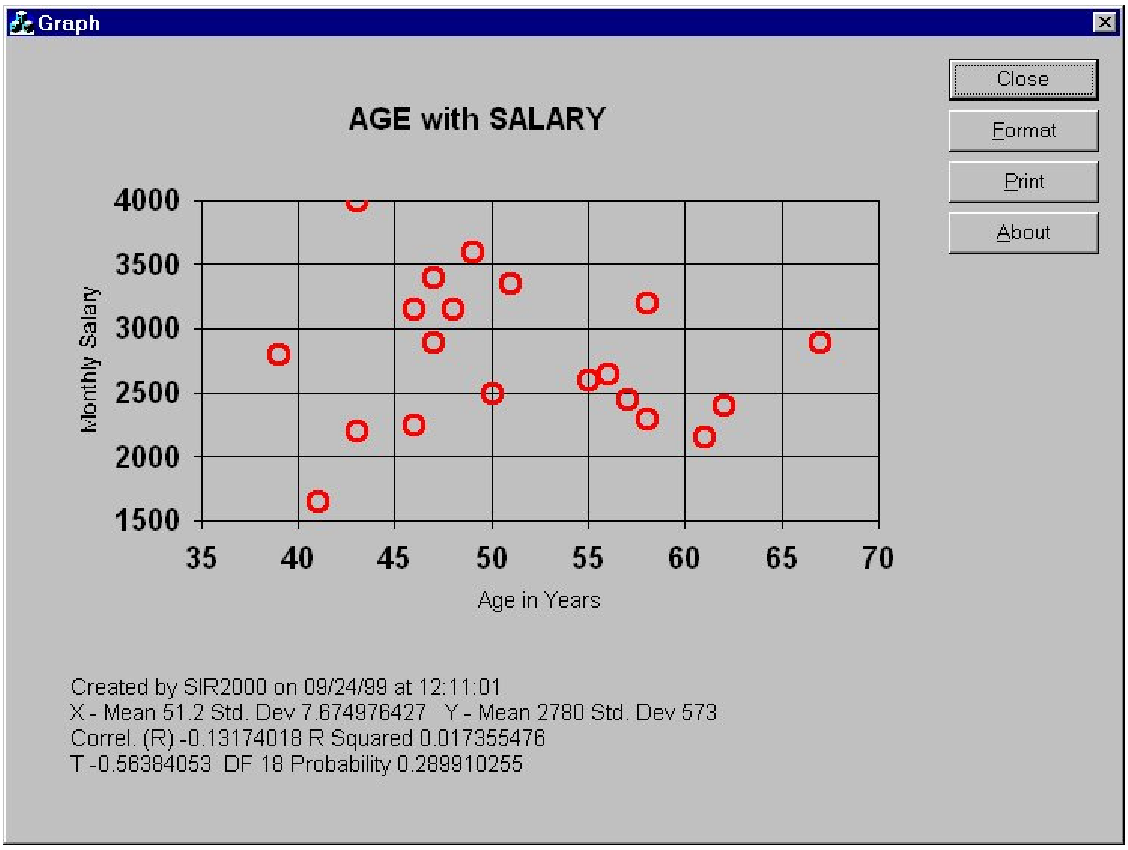
Produce statistics and a scatter plot age and salary.

```
RETRIEVAL
. INTEGER*4 AGE
. PROCESS CASES
.   PROCESS REC 1
.     COMPUTE AGE = (TODAY(0) - BIRTHDAY)/365
.     MOVE VAR SALARY GENDER
.     PERFORM PROCS
.   END PROCESS REC
. END PROCESS CASES
PLOT SCATTER = AGE, SALARY /
     FILENAME    = PLOT1.srg /
     XTITLE      = 'Age in Years'/
     YTITLE      = 'Monthly Salary'/
END RETRIEVAL
```

On completion of the program, the file `PLOT1.SRG` contains text which can be viewed with

```
ESCAPE 'sirgraph.exe plot1.srg'
```

which looks something like:



Created by SIR2000 on 09/24/99 at 12:11:01
X - Mean 51.2 Std. Dev 7.674976427 Y - Mean 2780 Std. Dev 573
Correl. (R) -0.13174018 R Squared 0.017355476
T -0.56384053 DF 18 Probability 0.289910255

Quick Report

Quick Report produces reports with a minimum of specification by the user. Quick Report offers many features including:

- automatic column headings
- automatic data formatting
- automatic totals and subtotals
- automatic layout of the report
- sorting and breakpointing

The REPORT Command

Specify a Quick Report using the `REPORT` command with a `PRINT` clause. A Full Report is specified using the `REPORT` command without the `PRINT` clause.

Quick Report vs. SQL

Quick Report produces reports while SQL display simply outputs data in a formatted way. Quick report can thus be used when reports need to be formatted. It can also be used when VisualPQL features such as the computation of new variables or the calculation of statistics are needed. It can also be simpler to use VisualPQL rather than SQL when complex navigation of the database is required.

Quick Report vs. Full Report

Quick Report is used for reports with:

- Fixed headings and footings
- Fixed field columns
- Simple totals and subtotals
- Simple breakpointing and sorting

Full Report gives complete control over the report output. Full Report allows the use of VisualPQL with additional commands for report formatting. Full Report requires that each detail of the report be specified; i.e., there are no default formats, headings or totals.

Consider using the Full Report when:

- The formats for headings change during the report.
- Formatting requirements go beyond a simple column structure.
- Computations are required during report.

- Summary information is required beyond simple subtotals or totals.
- Exact control of the output format is required.

The Structure of a Quick Report

The main components of a report are specified with keywords and options on the `REPORT` command. These control:

- Page Headings and Footings and Column headings.
- Detail lines displaying the data.
- Subtotal lines for sections of the report.
- Total line of grand totals.

Quick Report processes the procedure table records. If the report specifications include `SORT` or `BREAK` options, the procedure table is sorted. As each record is processed, a number of conditions may occur:

Page Break

`REPORT` keeps track of lines on a page. When the current page is full, the page footing is printed and a new page is created. The page counter is incremented, the heading for the new page is written and then column headers are written.

Column Break

`REPORT` tracks any specified *Break Variables*. When the value of a break variable changes, break actions are taken. Break actions include the printing of a subtotal line and possibly triggering a page break.

Detail Line

Detail Line actions are taken for each record in the procedure table after any Page or Column Break actions. Detail line actions include the printing of any specified blank lines above the detail line, the printing of the formatted data line itself and the printing of any specified blank lines below the detail line. Totals and subtotals are maintained.

The printing of detail lines can be suppressed.

End of Report Actions

After the last procedure table record has been dealt with, a final subtotal line (if required), the *Total Line* and page footing for the last page is printed.

Syntax

Quick Report is a single command with options:

```
REPORT FILENAME   = filename
       PRINT      = {variable [(['heading' LABEL] [position]
[format])]...| ALL

 [ BOOLEAN    = (logical_expression)]
 [ BREAK      = [(n)] variable [([A|D][G][C][P]['text'])],...]
 [ FOOTING    = [LEFT | RIGHT | CENTER] (footing) ]
 [ HEADCENTER   ]
 [ HEADING    = [LEFT | RIGHT | CENTER] (heading)  ]
 [ MISSCHAR   = character ]
 [ NOCENTER     ]
 [ NOCOLHEAD    ]
 [ NODETAIL     ]
 [ NOGROUPING   ]
 [ NOPAGEHEAD   ]
 [ NOPAGING     ]
 [ NOSORT       ]
 [ NOSUBTOTALS  ]
 [ NOTOTALS     ]
 [ NOUNDERCOL   ]
 [ NOUNDERHEAD  ]
 [ PAGEWIDTH  = c ]
 [ PAGELENGTH = l ]
 [ PAGESIZE   = l, c ]
 [ PAGELIMIT  = n ]
 [ SAMPLE     = fraction ]
 [ SHOWMISS ]
 [ SORT       = [(n)] variable [(A|D)], ...  ]
 [ SPACING    = SINGLE | DOUBLE | TRIPLE
                 COLHEAD (b,a)
                 DETAIL    (b,a)
                 SUBTOTALS (b,a)
                 TOTALS    (b,a) ]
 [ SUBTOTALS  = {break_var [(subtotal_var, ...)|(subtotal_var)} ...]]
 [ TOTALS     = variable, ......[('total_text')]  ]
 [ UPPERCASE    ]
```

`FILENAME`	Specify the filename produced by the procedure. This is a required clause.	
`PRINT`	Specifies that this is a Quick Report. This is a required clause. Specify the variables required. Each specified variable creates one column from left to right in the order specified.	
	`Heading`	The default heading is the variable name. Specify the keyword `LABEL` to use the variable label.

Specify a text string in quotes to use as the heading. For example:

```
PRINT = POSITION ('Job')
```

If the heading does not fit within the column, it is automatically split into multiple lines. Specify two or more text strings to control multiple heading lines. The character specified between the text strings controls how lines are printed.

Specify a blank to force a new line.

Specify a minus sign to break the heading if it does not fit on a line (conditional break). The general syntax is:

```
PRINT = variable ('text' [-]
'text')
```

Specify a plus sign (+) between two strings to concatenate them. Use this to specify long strings which do not fit on a single line (strings cannot be split across input lines).

`Position`

The default print position for each column is two characters to the right of the end of the previous column. By default, the overall report is left justified on the page.

`nT`

Specifies the absolute (**n**th) print position. For example, 45T specifies that the first position of the column is in print position 45.

`nX`

Specifies that the column is positioned **n** spaces to the right of the end of the previous column. Specify nX (n is a number) to change the position relative to the previous column.

For example, to print division in column 10 and salary four columns to the right:

```
PRINT =  division (10T)
         salary   (4X)
```

`Format`

The default format is taken from the schema. Specify a *format expression* to alter this. The width of a column is calculated from the format expression.

If the data for the column does not fit in the specified width, a numeric column is filled by **X**'s and a character column is truncated to fit.
In the format expressions below, "**w**" is a number specifying the width in columns.

`Aw`	Specifies **A**lphanumeric string format.
`Bw`	Specifies reverse string format. Column headers and data are printed in reverse (**b**ackwards).
`Lw`	Specifies that Value **L**abels are printed instead of the data value. If a value has no defined label, blanks are printed.
`Iw`	Specifies **I**nteger print format. Any decimal portion of the number is ignored.
`Fw.d`	Specifies **F**loating point print format. Use for either floating point or scaled integers with decimal portions. "**d**" is the number of decimal places.
`Ew`	Specifies **E**xponential (scientific).
`DATE`	Specifies a date in the appropriate format. See date formats for a complete description of date format specifications. The width of the column is specified by the characters in the date format. For example: `PRINT = BIRTHDAY (DATE'Wwwwww Mmm DD, YYYY')`
`TIME`	Specifies a time in the appropriate format. See time formats for a complete description of time format specifications. The width of the column is specified by the characters in the time format. For example: `PRINT = TESTTIME (TIME'HH:MM:SS PP')`
`D, C, P`	D puts a dollar sign (**$**) before the numeric value. C separates thousands with the comma character. P puts a percent sign (**%**) after the

numeric value. These can be specified in addition to other numeric format expressions. For example:
`PRINT MONEY (F9.2, D, C)`

`BOOLEAN`	Selects procedure table records. If the logical expression is true for the record, the record is used in the report. The variable names used in the expression must be procedure variables. For example, to select all procedure records where `SALARY` is greater than or equal to 18000: `BOOLEAN = (SALARY GE 18000)`
`BREAK`	Specifies break variables which control subtotaling. Specify the variables in break level order, most significant first.

`n`	Defines the number of procedure records to be sorted. The default is the
`A|D`	`A` specifies sort in ascending order; this is the default. `D` specifies sort in
`G`	Turns on grouping for the variable. Grouping means that repeated values
`C`	Turns on reprinting of column headings when the variable breaks. If
`P`	Starts a new page when this variable breaks. By default, there is no paging

	breaks. By default, there is no paging at break points.
`'text'`	Specifies a string printed on the subtotal line when the variable breaks; the default is the variable name. If the `BREAK` clause is specified without a `SORT` clause, the order of the break variables defines the sorting order for the procedure table records. If a `SORT` clause is specified, its sorting order is used. If the `NOSORT` keyword is specified, no sorting occurs. For example, to sort and break on age in descending order within department, with a new page for each department: `BREAK = DEPARTMENT (P) , AGE(D)`

`SORT`	Specifies the sequence of the output. **n** is an integer that specifies the maximum number of records to be sorted. The default for this parameter is either the number of records in the database or the value specified in the sortn parameter and need only be specified if the number of records in the procedure table is greater than the default. The procedure table is sorted by the specified variables in variable list order. A variable name followed by (**A**) or (**D**) specifies that for that variable the sort is in **A**scending order (the default) or in **D**escending order.

`FOOTING`	`FOOTING` defines one or more lines of text printed at the bottom of each page of the report. There is no default page footing.
	The specification for headings and footings have the same clauses:
	`LEFT|RIGHT|CENTER`
	Left, right or centre justifies the heading or footing. `LEFT` is the default.
	`output spec`
	Specify the text of the heading or footing in parentheses using the same syntax as the WRITE command.
	Three system maintained variables may be used in the specification:
	`DATE`, for current date; `TIME`, for current time; and `PAGE`, for current page number, are available for use in the `HEADING` and `FOOTING` output specifications. For example:
	`FOOTING = CENTER ( '- ' PAGE '-' )`

`HEADCENTER`	Centers column headings within each column. By default, column headings for string variables are left justified and column headings for numeric variables are right justified.

`HEADING`	`HEADING` defines one or more lines of text printed at the top of each page of the report. The default heading is the date, the time and the page number. Suppress the default heading with the `NOHEADING` keyword. If the `NOPAGING` keyword is specified, no headings or footings are produced.
	The specification for headings and footings have the same clauses:
	`LEFT|RIGHT|CENTER`
	Left, right or centre justifies the heading or footing. `LEFT` is the default.
	`output spec`
	Specify the text of the heading or footing in parentheses using the same syntax as the WRITE command.
	Three system maintained variables may be used in the specification:
	`DATE`, for current date; `TIME`, for current time; and `PAGE`, for current page number, are available for use in the `HEADING` and `FOOTING` output specifications. For example:

```
HEADING = ('Employee Report', 2X, DATE,
            60T, 'Page ' PAGE(I3)  ) /
```

`MISSCHAR`	Specifies the character printed for fields containing missing values. The default character is the asterisk (*****). The specified character may be any character (including blank), except the slash (/) and comma (,).
`NOCENTER`	Left justifies the report. By default, when output to a file, the report is centred. (If output to `CONSOL`, the report is left-justified.) Specify `NOCENTER` when positioning individual columns using nT (tab).
`NOCOLHEAD`	Turns off column heading printing at all break points. By default, column headings are printed at break points. If `NOCOLHEAD` is specified, specifying individual column headings on the `BREAK` option, results in a compilation error.
`NODETAIL`	Turns off the printing of detail lines, only subtotal lines are printed. By default, detail lines are printed.
`NOGROUPING`	Turns off grouping for all break variables. By default, grouping is on.
`NOPAGEHEAD`	Turns off printing of page headings. By default, page headings are printed.
`NOPAGING`	Suppresses all paging. By default, page ejects are performed when a page is full.
`NOSORT`	Turns off sorting of procedure records. By default, the records are sorted when a `BREAK` or `SORT` clause is specified.
`NOSUBTOTALS`	Turns off subtotals. By default, subtotals are calculated and printed for numeric variables when a `BREAK` clause is specified.

`NOTOTALS`	Turns off grand totals. By default, grand totals are calculated and printed for numeric variables.
`NOUNDERCOL`	Turns off underlining of the detail line above subtotals. By default, a detail line above a subtotal is underlined.
`NOUNDERHEAD`	Turns off underlining of column headings. By default, column headings are underlined. Underlines by minus (-) printed one line below the text.
`PAGEWIDTH=n`	Sets the number of print characters per line. The default is to make the report as wide as necessary.
`PAGELENGTH=n`	Sets the number of print lines per page. The default is 60 lines per page. The `NOEJECT` keyword suppresses paging entirely.
`PAGESIZE=l,c`	Sets the pagelength and pagewidth in a single statement. l is the number of lines; c is the number of columns. If the required width of a report exceeds the page width, a compilation error message is issued and the program is not executed.
`PAGELIMIT=n`	Sets the maximum number of pages produced. When this limit is reached, the report is terminated.
`SAMPLE`	Specifies that a random sample of the procedure table records are used by the `REPORT`. The **fraction** specifies the percent of records used and is specified as a positive decimal number less than or equal to 1 (one). .25, for example specifies that a 25% sample be used.
`SHOWMISS`	Specifies that a variable's original missing values are printed for fields containing missing values. The default character is the asterisk (*****). Missing values are always excluded from totals - this option only affects printing.
`SORT`	Specifies the sorting order for the report. **n** is an integer that specifies the maximum number of records to be sorted. The default for this parameter is either the number of records in the database or the value specified in the sortn parameter and need only be specified if the number of records in the procedure table is greater than the default. The procedure table is sorted by the specified variables in variable list order. A variable name followed by (**A**) or (**D**) specifies that for that variable the sort is in **A**scending order (the default) or in **D**escending order. For example, to sort on `NAME` in ascending order and `SALARY` in descending order: `SORT = NAME SALARY (D)`
`SPACING`	Controls the spacing between lines of the report.

`SINGLE`	Single spaces detail lines. This is the default.

`DOUBLE`	Double space detail lines. Each detail line is followed by one blank line.
`TRIPLE`	Triple space detail lines. Each detail line is followed by two blank lines.
`COLHEAD`	Specifies the number of blank lines printed before and after column headers.
`DETAIL`	Specifies the number of blank lines printed before and after each detail section.
`SUBTOTALS`	Specifies the number of blank lines printed before and after each subtotal line.
`TOTALS`	Specifies the number of blank lines printed before and after each total line. The default spacing options are:

```
SPACING  =        SINGLE
                  COLHEAD   ( 1
, 0 )
                  DETAIL    ( 0
, 0 )
                  SUBTOTALS ( 0
, 1 )
                  TOTALS    ( 0
, 1 )
```

`SUBTOTALS`

Specifies variables to subtotal. The `BREAK` clause also specifies when subtotals are printed. By default, all numeric variables, that do not appear in the `BREAK` clause, are subtotaled.

Two types of variables can be specified on the subtotal clause. Break variables, i.e. the variables which control the printing of the subtotals, and subtotal variables, i.e. variables which are added up to calculate the subtotals. Differentiate these by the use of parentheses (). Specify break variables outside the parentheses; specify subtotal variables inside the parentheses. Either set of variables may be omitted. This gives rise to three possible formats of the `SUBTOTALS` clause, one with just break variables, the second with just subtotal variables and the third with both:

`SUBTOTALS = break variable, ...`

Specify break variables to determine which breaks cause subtotal printing. By default, all breaks print subtotals.

`SUBTOTALS = (subtotal variable, ...)`

Specify subtotal variables to determine which numeric variables are subtotaled. By default, all numeric variables which are not break variables are subtotaled. Subtotals cannot be calculated for variables which cause breaks. Subtotals are printed at every

	breakpoint. `SUBTOTALS = break variable1 (subtotal variable1, ...)` `break variable2 (subtotal variable2, ...) ...` Specify both break variables and subtotal variables to determine which subtotals print at which breaks. Subtotals in the parenthesised list are printed when the corresponding break variable changes value. For example, subtotals for the variables `GROSS` and `NET` are calculated and printed when `STATE` changes value; subtotals for the variable `TAXES` are calculated and printed when `COUNTY` changes values.

```
SUBTOTALS =      STATE   (GROSS NET)
                 COUNTY  (TAXES)
```

`TOTALS`	Specifies grand totals. By default, grand totals are printed for all numeric variables. The variable list specifies numeric variables which are then totaled. To suppress totals, specify the `NOTOTALS` keyword. The 'total text' string is optional and defines the text printed to the left of the grand totals. If this string is not specified, the default text '`TOTAL`' is printed. For example, to calculate totals for `GROSS` and `NET` and print them with the label 'Grand Totals'. `TOTALS = GROSS NET ('Grand Totals')`
`UPPERCASE`	Converts all text in the report to uppercase. By default, both upper and lower case are produced.

See Examples.

Examples

Example 1: Standard Program

The following VisualPQL program is used for all the examples in this section. The `REPORT` specifications use the procedure table produced by this program.

```
RETRIEVAL
. PROCESS CASES
.   RECORD IS EMPLOYEE
C    Find Last Name
.     COMPUTE REVNAME = TRIMLR(REVERSE (NAME))
.     COMPUTE LEN     = ABS (SRST(REVNAME,' '))
.     COMPUTE LNAME   = REVERSE(SBST(REVNAME,1,LEN-1)
.   END RECORD IS
.   PROCESS REC OCCUP
.     GET VARS POSITION STARTDAT STARTSAL DIVISION
.     PERFORM PROCS       | copy variables to proc table
.   END PROCESS REC
. END CASES
REPORT . . .
 .......
END RETRIEVAL
```

Example 2: A Simple Report

This simple report uses most of the defaults provided by the Quick Report Procedure. The page is made narrower than the default 136 and totals are suppressed.

```
REPORT FILENAME = REPORT1.REP
       PRINT    = LNAME DIVISION POSITION STARTDAT STARTSAL
       PAGESIZE = 60 , 79
       NOTOTALS

Dec 27, 2005      10:07:25                 Page     1

LNAME           DIVISION  POSITION  STARTDAT  STARTSAL
--------------- --------  --------  --------  --------
Jones                  1         4  02 10 80      1500
Jones                  1         5  10 15 81      2000
Arblaster              1         6  01 18 80      2500
Black                  1         9  10 13 79      2750
Black                  1        10  02 18 81      3000
Brown                  1        14  10 13 77      3200
Green                  1        10  11 04 79      3000
Safer                  1         9  05 07 79      2000
Safer                  1        10  03 08 81      2500
West                   1        12  07 10 81      2200
Moore                  1        13  01 01 74      2200
```

Example 3: Value Labels, Headings and Footings

This example specifies that value labels for `DIVISION` and `POSITION` are printed rather than their numeric values. `STARTSAL` is formatted as money with decimal digits, dollar signs and commas to separate thousands. The page number is displayed in the footing and a page header is defined.

```
REPORT FILENAME = REPORT2.REP
       PRINT    = LNAME     (A12)
                  DIVISION (L10)
                  POSITION (L10)
                  STARTDAT
                  STARTSAL ( F10.2 , D , C )
      HEADING  =('Employee Jobs and Starting Salaries' )
      FOOTING  = CENTER ( '- ' PAGE'-' )
      PAGESIZE = 60 , 79
```

```
Employee Jobs and Starting Salaries

NAME          DIVISION   POSITION    STARTDAT    STARTSAL
-----------   --------   ---------   --------   ----------
Jones         Chemical   Laborer     02 10 80   $1,500.00
Jones         Chemical   Technician  10 15 81   $2,000.00
Arblaster     Chemical   SnrTechn    01 18 80   $2,500.00
Black         Chemical   Chemist     10 13 79   $2,750.00
Hiller        Corporate  Sr Engin    01 12 75   $2,600.00
......................
......................
Nugent       Corporate   Sr Accoun   07 03 78   $2,300.00
Neuman       Corporate   Engineer    04 12 79   $2,000.00
Pau          Manufactur  Sr Techn    10 10 78   $2,700.00
Fauntleroy   Manufactur  Sr Secre    06 11 80   $2,000.00
Josephine    Manufactur  Director    01 10 76   $3,000.00
Rabinowitz   Corporate   President   01 01 73   $4,000.00
----------
                                       TOTAL   $75,550.00
- 1 -
```

Example 4: Break Variables and Column Headers

This example creates a report section for each division in the company by *breaking* on division. Each division section begins on a new page. Note the use of an expression in the `HEADING` clause that causes the division value label to appear in the page header. The column headers have also been customised.

```
REPORT FILENAME   = REPORT3.REP
       PRINT    = DIVISION (L13 , 'Division')
                    LNAME    (A13 , 'Employee' 'Name' )
                    POSITION (L16 , 'Job' 'Title' )
                    STARTDAT (DATE 'MM/DD/YY' , 'Job Starting Date')
                    STARTSAL (F10.2 , D , C , 'Starting' 'Salary')
       BREAK    = DIVISION ('Division Totals' , P)
       TOTALS   = STARTSAL ('Company Totals')
       HEADING  = ([TRIM(VALLAB(DIVISION))
                    + ' Division Job History'])
       FOOTING  = CENTER ( '- ' PAGE '-' )
       PAGESIZE = 60 , 79
Chemical Division Job History

                                   Job
          Employee   Job           Starting    Starting
Division  Name       Title         Date        Salary
--------- ---------  ---------     ---------   ---------
Chemical  Jones      Laborer       02/10/80    $1,500.00
          Jones      Technician    10/15/81    $2,000.00
          Arblaster  SrTechnician  01/18/80    $2,500.00
          Black      Chemist       10/13/79    $2,750.00
          Black      Sr Chemist    02/18/81    $3,000.00
          Brown      Sr Administr  10/13/77    $3,200.00
          Green      Sr Chemist    11/04/79    $3,000.00
          Safer      Chemist       05/07/79    $2,000.00
          Safer      Sr Chemist    03/08/81    $2,500.00
          West       Sr Engineer   07/10/81    $2,200.00
          Moore      Administrator 01/01/74    $2,200.00
          Moore      Sr Administr  12/05/75    $2,700.00
          Moore      Director      01/07/77    $3,500.00
                                              ----------
Division Totals                               $33,050.00

- 1 -
................
................
................
Company Totals                              $75,550.00
```

Example 5: Two Break Variables and Suppressed Subtotals

This example breaks on `POSITION` within `DIVISION` but suppress subtotals and repeated column header for `POSITION` to get *grouping*. The page heading is centred.

```
REPORT FILENAME = REPORT4.REP
       PRINT = DIVISION (L10 , 'Division')
               POSITION (L10 , 'Job' 'Title' )
               LNAME(A12 , 'Employee' 'Name' )
               STARTDAT (DATE'MM/DD/YY','Job'-'Starting'-'Date')
               STARTSAL (F10.2 , D , C ,'Starting' 'Salary')
       BREAK = DIVISION ('Division Totals' , P , C)
               POSITION
       SUBTOTALS= DIVISION (STARTSAL)
       TOTALS   = STARTSAL('Company Totals')
       HEADING  = CENTER([TRIM(VALLAB(DIVISION)) +
                 ' Division Job History'])
       FOOTING  = CENTER ( '- ' PAGE '-' )
       PAGESIZE = 60 , 79

                      Chemical Division Job History

                                 Job
            Job        Employee       Starting    Starting
Division    Title      Name           Date        Salary
--------  ----------  ------------   --------    ----------
Chemical  Laborer     Jones          02/10/80    $1,500.00
          Technician  Jones          10/15/81    $2,000.00
                      Arblaster      01/18/80    $2,500.00
          Chemist     Black          10/13/79    $2,750.00
                      Safer          05/07/79    $2,000.00
                      West           02/18/81    $3,000.00
                      Green          11/04/79    $3,000.00
                      Safer          03/08/81    $2,500.00
          Sr Enginee  West           07/10/81    $2,200.00
          Administra  Moore          01/01/74    $2,200.00
          Sr Adminis  Brown          10/13/77    $3,200.00
                      Moore          12/05/75    $2,700.00
          Director    Moore          01/07/77    $3,500.00
Division Totals                                 $33,050.00
                                  - 1 -
...........
...........
 ----------
Company Totals                                  $75,550.00
```

Full Report

The Full Report procedure offers the facilities of VisualPQL within a structure provided by additional report processing commands.

Full Report differs from other procedures in that it is specified with a set of commands rather than a single command. The report specification follows the first part of the program as do the other VisualPQL Procedures.

In contrast to Quick Report, Full Report gives precise control over the program logic and the structure and appearance of the report. Full Report is used when branched reports are required or computations beyond subtotals and totals are needed. It is also used when output formats other than columns are needed and when different sections of the report have different formats.

A Full Report procedure starts with the REPORT command without the `PRINT` option, and ends with an END REPORT command. All commands from `REPORT` to `END REPORT` are a single `REPORT` procedure. A single program may include an unlimited number of `REPORT` procedures. Output from each report procedure is written to a separate file.

As the report executes, each record in the procedure table is processed. The values in any given procedure table record are set at the time the `PERFORM PROCS` command copies the local variables to the procedure table. Values in the procedure table cannot be updated in the procedure. New variables (variables that were not used before the report) can be created and used as required.

If multiple report procedures are specified in one program, the local variables used in one are not available for update in subsequent procedures. In other words, the locally defined variables in the first report become procedure variables in subsequent reports and cannot be modified. If referenced, these contain the last value assigned.

Specify a BEFORE REPORT or AFTER REPORT to create blocks of commands that are executed before or after a report. If any new local variables are required these are typically declared in the `BEFORE REPORT` block using any of the standard PQL variable definition features.

The key structuring in a report is Break Levels. Breaks are triggered by the change in value of a named variable and determine the appropriate processing for that condition. If a break level block is specified without a variable, it is actioned for every record.

Commands are further broken into Action Blocks. The action blocks identify sets of commands executed within a break level for particular conditions such as when the break level initially happens, for every record in the break level and at the end of the break level.

Once the procedure has identified the appropriate block to execute, it executes these standard VisualPQL commands. The primary output for producing a report is the PQL WRITE command. Every detail line that appears in the final report gets there because a `WRITE` command specified it.

Page Breaks

`REPORT` tracks how full a page is and performs page breaks. The PAGE EJECT command also causes a new page.

One type of action block, the PAGE BLOCK specifies the commands to be executed when there is a page break. When a page break occurs, all specified page blocks in all levels are executed.

The HEADING, HEADING BLOCK, FOOTING or FOOTING BLOCK commands specify the headings or footings which are output when a page break occurs. `HEADING BLOCK` or `FOOTING BLOCK` define multiple lines, `HEADING` or `FOOTING` define a single line.

These commands are executed according to the flow of control and may alter the heading, but the output is not written until the page break occurs. If the heading or footing is only defined once, the recommended place for these blocks is in the BEFORE REPORT block of commands.

The VisualPQL Subset for Full Report

All VisualPQL commands and functions may be used in full report except those dealing with database and table processing (case, record and row commands) and those using subroutines and subprocedures. The excluded commands are:

- Database commands including PROCESS CASE, CASE IS, PROCESS REC, RECORD IS;
- Tabfile commands including PROCESS ROW, ROW IS;
- GET VARS and
- PUT VARS;
- PERFORM PROCS
- SUBPROCEDURE,
- EXECUTE SUBPROCEDURE and
- EXECUTE SUBROUTINE;
- database functions
- PQL Procedures

Syntax

The syntax for the `REPORT` command is:

```
REPORT    FILENAME  = filename
  [ BOOLEAN  = (logical condition)]
  [ MISSCHAR = char ]
  [ PAGESIZE = lines[,chars]]
  [ SAMPLE   = fraction]
  [ SHOWMISS ]
  [ SORT     = [(n)] varname [A|D]...]
```

`REPORT,` without the `PRINT` option, specifies the full report procedure.

`FILENAME`	Specify the filename produced by the procedure. This is a required clause.
`BOOLEAN`	Selects procedure table records. If the logical expression is true for the record, the record is used in the report. The variable names used in the expression must be procedure variables.
`MISSCHAR`	Specifies the character printed for variables having missing values. The default is an asterisk (*). The specified character may be any character including blank, except the slash (/) or comma (,).
`PAGESIZE`	Sets the page length and page width of the Report output file. The default page size is 60 lines per page and 136 print positions (characters) per line.
`SAMPLE`	Specifies that a random sample of the procedure table records are used by the procedure. The **fraction** specifies the percent of records used and is specified as a positive decimal number less than or equal to 1 (one). .25, for example specifies that a 25% sample be used.
`SHOWMISS`	Specifies that a variable's original missing values are printed for fields containing missing values. The default character is the asterisk (*****). Missing values are always excluded from totals - this option only affects printing.
`SORT`	Specifies the sequence of the output. **n** is an integer that specifies the maximum number of records to be sorted. The default for this parameter is either the number of records in the database or the value specified in the sortn parameter and need only be specified if the number of records in the procedure table is greater than the default. The procedure table is sorted by the specified variables in

variable list order. A variable name followed by (**A**) or (**D**) specifies that for that variable the sort is in **A**scending order (the default) or in **D**escending order.

For example:

```
REPORT FILENAME = REPORT1.LIS /
       SORT       = GENDER /
       PAGESIZE   = 60,132
```

AFTER REPORT

`AFTER REPORT`

Initiates a block of commands executed once at the end of the report procedure. If this command is used, it must be the last report block in the report specification.

`AFTER REPORT` is typically used to print report summary information such as grand totals and other statistics.

It is not recommended practice to reference procedure variables in this block. If procedure variables are referenced, then, if the report contained a `SORT` clause, these contain the last record in the procedure table. If the report does not contain a `SORT`, these contain the last values put in by the main body of the program.

AT END BLOCK

```
AT END BLOCK
```

Initiates a block of commands executed when a break condition is triggered, before the next `INITIAL BLOCK` is executed. It is also executed after the last record has been processed.

When an `AT END BLOCK` is executed, the old procedure table record prior to the break condition is current. It is not recommended practice to reference procedure records in the `AT END BLOCK`. If a value from a procedure variable is needed, compute the value into a report variable in either the `INITIAL BLOCK` or the `DETAIL BLOCK`.

If multiple `AT END BLOCK`s along a report path are triggered by higher level break conditions, all of the `AT END BLOCK`s are executed in reverse order, from highest numbered break level outwards.

BEFORE REPORT

`BEFORE REPORT`

`BEFORE REPORT` initiates a block of commands executed once at the beginning of the report. The block is terminated by the first `BREAK LEVEL` command. If `BEFORE REPORT` is specified, it must be the first command following `REPORT`. The first procedure table record is available in the `BEFORE REPORT` block. `BEFORE REPORT` is used to:

- Declare report variables
- Initialise report variables such as totals
- Define report page headings and footings
- Print a report cover page

For example:

```
REPORT   FILENAME = 'EXAMPLE.REP' PAGESIZE = 66,80
BEFORE REPORT
. STRING * 80 TITLE FOOTLINE HEADLINE
. INTEGER SALTOTAL SALCOUNT
. SET TITLE FOOTLINE HEADLINE ('')
. SET SALTOTAL SALCOUNT ( 0 )
. HEADING BLOCK 2
.   COMPUTE HEADLINE = 'Salary Report'
.   WRITE HEADLINE
.   COMPUTE HEADLINE = DATEC( TODAY(0) , '     MM/DD/YY')
.   WRITE HEADLINE
. END HEADING BLOCK
. FOOTING 37T 'Page ' PAGE
. PAGE EJECT
BREAK LEVEL 1
..........
```

BLANK LINES

```
BLANK LINES n
```

Skips the specified number of blank lines. Blank lines specified by this command do not extend across pages. If the command causes a page break, counters are reset and the new page produced. In contrast, the `WRITE` command produces physical blank lines which do span pages.

BREAK LEVEL

```
BREAK LEVEL {break_level [,break varname ]} |
                 {break_level.identifier (condition)}
```

`BREAK LEVEL` defines a break condition and starts the block of commands executed when the condition is true. End the block of commands with the END BREAK LEVEL command. Every report has at least one break level and can have as many as necessary. Multiple break levels are defined hierarchically and are nested within each other. Each break level is uniquely identified with a number which increases as more deeply nested levels are defined. That is the highest level is 1, the next is 2, etc.

A break at a level causes breaks at all lower levels. The first record triggers the top level break.

It is possible to specify a logical break condition on the command. This creates a report that can have different formats depending on the data values. The CONNECT TO command can be used to execute a lower level break without having to respecify it in every logical branch.

The ON ERROR command is equivalent to a break level command and deals with records not matching any other logical break condition.

For each break level, specify *Action Block(s)* which contain commands that are executed when the break condition is encountered. Each action block command initiates a block of commands that is ended by another action block command or by the end of the break level. If an action block is not specified, commands in the break level are considered to be in a detail block.

The four action blocks are:

INITIAL BLOCK which is executed when the break initially happens.

PAGE BLOCK which is executed when a page break happens.

DETAIL BLOCK which is executed for every record in the break level.

AT END BLOCK which is executed at the end of the break.

Simple Break

A simple break is specified with the following syntax:

```
BREAK LEVEL    level   [,break varname ]
```

A simple break is triggered by a change in the value of the named variable from one procedure table record to the next. For example:

```
REPORT ....    SORT = GENDER AGE
BREAK LEVEL 1, GENDER
BREAK LEVEL 2, AGE
```

Conditional Break

Conditional breaks allow specification of different actions that depend on the value of the break variable. Conditional breaks occur when the value changes to a particular value, as opposed to simple breaks which occur whenever the value changes. This branch of the break level is executed when the condition is true. For example, different report formats might be produced for males as opposed to females.

Other break levels (either simple or conditional) may be defined within conditional break levels. Each procedure table record that matches the specified condition follows the path of break levels nested within it. Typically, this means that a detail block is defined within each path.

Once a conditional break is specified, specify the entire branch, including any more deeply nested break levels, before specifying other conditional breaks at the original level. When specifying conditional break levels, specify a branch for all possibilities.

Use the ON ERROR command to specify the path to take for any unanticipated conditions.

A conditional break has additional syntax. Multiple conditions specify the same level, and the level is further qualified by a *condition identifier*. which is a number following the level, separated by a period. The break level is initiated by the specified condition being met. Specify the condition in parentheses. For example:

```
BREAK LEVEL 1.1 (GENDER = 1)
BREAK LEVEL 1.2 (GENDER = 2)
```

A conditional break creates a branching structure which may have further lower levels. These must have unique level numbers. Instead of additional level numbers, lower level simple breaks can qualify the level number with from one to three characters. These have no meaning other than as a label. The level number determines the level. For example:

```
BREAK LEVEL 1.1 (GENDER = 1)
BREAK LEVEL 2A AGE
BREAK LEVEL 1.2 (GENDER = 2)
BREAK LEVEL 2B AGE
```

Any conditional lower level breaks must use the level.identifier (N.n) syntax.

CONNECT TO

```
CONNECT TO level.condition_ident
```

`CONNECT TO` specifies that a `BREAK LEVEL` in another branch is executed at that point. The `BREAK LEVEL` referenced on the `CONNECT TO` must have been defined previously and must be a lower level i.e have a numerically higher number.

Complex branched reports frequently converge at some lower level, for example, the specifications for level 4 detail blocks might be identical. Respecifying identical blocks in different paths is avoided by using `CONNECT TO`. For example:

```
BREAK LEVEL 1.1 (GENDER = 1)
BREAK LEVEL  2.1 (AGE LT 18)
.....  commands
BREAK LEVEL  2.2 (AGE GE 18)
.....  commands
BREAK LEVEL  1.2 (GENDER = 2)
BREAK LEVEL  2.3 (AGE LT 18)
.....  commands
BREAK LEVEL  2.4 (AGE GE 18)
CONNECT TO   2.2
```

DETAIL BLOCK

```
DETAIL BLOCK
```

Commands in the `DETAIL BLOCK` are executed once for each procedure table record. If the block is within a conditional break level, it is only executed for records which satisfy the condition.

A typical report has one `DETAIL BLOCK` in each branch of the report, though there is no restriction on how many different break levels may contain detail blocks.

END BREAK LEVEL

```
END BREAK LEVEL
```

Defines the end of a break level.

END REPORT

```
END REPORT
```

Defines the end of the report procedure. This is not required and is specified for readability.

FOOTING

```
FOOTING  output_specifications
```

Specifies the text printed at the bottom of each report page. The text is written when the page eject occurs. The syntax of the `FOOTING` command is identical to that of the WRITE and the HEADING command. In addition, report variables and the system maintained variables `PAGE, DATE`, and `TIME` may be used to print the current page, date and time. If multiple `FOOTING` commands are executed, the output from the most recent is written. Do not specify both a `FOOTING` and a `FOOTING BLOCK`. There is no default `FOOTING`. For example:

```
FOOTING 70T 'Page ' PAGE
```

FOOTING BLOCK

```
FOOTING BLOCK n
```

Specifies a block of commands which creates a footing to be output when a page break is encountered. The command must appear within a break level or before report block. The block is terminated with `END FOOTING BLOCK`. The WRITE command specifies the output. Typically used when the footer contains multiple lines, when logical conditions control the footer text and when computations are performed to construct the footer. The maximum number of output lines is specified on the command. If multiple `FOOTING BLOCK` commands are executed, the output from the most recent is written. Do not specify both a `FOOTING BLOCK` and a `FOOTING`. There is no default `FOOTING BLOCK`. For example:

```
FOOTING BLOCK 1
.  IF(PAGE EQ 1) WRITE 33T 'Company Report'
.  IF(PAGE GT 1) WRITE 38T '-' PAGE '-'
END FOOTING BLOCK
```

HEADING

```
HEADING output_specifications
```

Specifies the text printed at the top of each report page. The text is written when the page eject occurs. The syntax of the `HEADING` command is identical to that of the WRITE and the FOOTING command. In addition, report variables and the system maintained variables `PAGE, DATE`, and `TIME` may be used to print the current page, date and time. If multiple `HEADING` commands are executed, the output from the most recent is written. Do not specify both a `HEADING` and a `HEADING BLOCK`. There is no default `HEADING`. For example:

```
'Company Report' 65T DATE(DATE'Mmm DD, YYYY')
```

HEADING BLOCK

`HEADING BLOCK n`

Defines a block of commands executed when a page break is encountered. The command must appear within a break level or before report block. The block is terminated with `END HEADING BLOCK`. The WRITE command specifies the output. Typically used when the header contains multiple lines, when logical conditions control the header and when computations are performed to construct the header text. Specify the maximum number of output lines the block can produce on the command. If multiple `HEADING BLOCK` commands are executed, the output from the most recent is written. Do not specify both a `HEADING` and a `HEADING BLOCK`. There is no default `HEADING BLOCK`. For example:

```
HEADING BLOCK 3
.  COMPUTE HEADLINE = 'Company Report'
.    WRITE HEADLINE
.  COMPUTE HEADLINE = DATEC (TODAY(0) , '     MM-DD-YY'
.    WRITE HEADLINE
.  COMPUTE HEADLINE = 'Division: ' + VALLAB(DIVISION)
.    WRITE HEADLINE
END HEADING BLOCK
```

INITIAL BLOCK

```
INITIAL BLOCK
```

The `INITIAL BLOCK` is executed once each time the break condition is triggered, i.e. when the value of the break variable changes. This block is executed for the first record.

ON ERROR

```
ON ERROR [ level.condition ]
```

The `ON ERROR` command is a special form of the `BREAK LEVEL` command that may be specified once at any conditional break level. It defines actions for conditions not explicitly covered on other `BREAK LEVEL` commands at that level.

A single `ON ERROR` block may be specified without the level identifier and it may be specified at any point. This block is executed any time a procedure table record is not covered by a break condition.

Typically, `ON ERROR` blocks contain code to display error messages and terminate the program (with the `STOP` command).

If there are no `ON ERROR` levels of any kind within a report and an error is detected, i.e., a procedure record is read that does not meet any of the logical conditions at a branching point, the program and report are automatically terminated. Any output written to the report file up to the point of such a termination is preserved.

PAGE BLOCK

```
PAGE BLOCK [ n ]
```

The `PAGE BLOCK` is executed under two conditions. It is executed on every *page eject* and is also executed when the break is triggered.

A number may be specified on the `PAGE BLOCK` command. If the command is executed because of the break and fewer lines than this remain on the current page, a page eject is done. This ensures that there is room for data after printing out column headers.

Note: A page block does not cause a page break, it is executed when a page break occurs. To force a page break, use the `PAGE EJECT` command at a suitable place in another action block.

PAGE EJECT

```
PAGE EJECT n
```

Causes a page break when executed. This can be used to trigger all the page break dependent code in the report. If a number is specified, this conditionally executes the Page Eject if fewer than the specified number of lines remain on the current page. For example, to force a section of the report to begin on a new page, place the `PAGE EJECT` in the `INITIAL BLOCK`.

Examples

The following are examples of full report:

A simple listing

Simple statistics

A simple branched report

Simple breaks

Detail lines and subtotals

Multiple breaks

Branched report with different formats

Underlines

Total data in AFTER REPORT

External files and Edit Buffers

Example 1: A Simple Listing

This example is a very simple full report. It lists the values of the variable `ID`, which is the case identifier in the example database.

```
RETRIEVAL
.  PROCESS CASES
.    GET VARS ID           | put ID in local variables
.    PERFORM PROCS         | copy procedure rec to procedure table
.  END CASE
REPORT FILENAME= 'REPORT1.REP' | name the report file
.  BREAK LEVEL 1               | dummy break
.   DETAIL BLOCK               | for each procedure table record
.   WRITE ID                   | write ID to the report file
.  END BREAK LEVEL
END REPORT
END RETRIEVAL
```

The above program illustrates the basic requirements for a report specification. These requirements are:

- values are put in local variables which are written to the procedure table with a PERFORM PROCS
- the REPORT command specifies an output file
- there is at least one BREAK LEVEL
- the DETAIL BLOCK specifies actions for each procedure table record
- a WRITE command outputs data to the Report file

Though neither the `DETAIL BLOCK` nor the `WRITE` command are strictly required by the VisualPQL compiler, if these are deleted, the program would execute but would write nothing to the file. The report outputs 20 lines, each with an ID number:

```
1
2
.....
20
```

Example 2: Simple Statistics

Suppose a count of the records is required. This requires a minor change to the previous example:

```
RETRIEVAL
.  PROCESS CASES
.    GET VARS ID
.    PERFORM PROCS
.  END CASE
REPORT FILENAME = 'REPORT2.REP'
BEFORE REPORT
.  SET IDCOUNT (0)
.  BREAK LEVEL 1
.    DETAIL BLOCK
.    COMPUTE IDCOUNT = IDCOUNT + 1
.  AFTER REPORT
.  WRITE 'The Count is: ' IDCOUNT
END REPORT
END RETRIEVAL
```

The count is initialised in a `BEFORE REPORT` block.

The detail block is executed once for each procedure table record and is used to increment the count. The `AFTER REPORT` block is executed when the entire procedure table has been processed to write out the totals. This report outputs a single line.

```
The count is:  20
```

Example 3: A Simple Branched Report

This example, and the next, show two different techniques for producing a count by sex. The first example uses a conditional break to construct a report that branches to a different `COMPUTE` statement depending on the value of the variable `GENDER`. Each computation increments a different counter variable.

```
RETRIEVAL
. PROCESS CASES
.   PROCESS REC EMPLOYEE
.     GET VARS GENDER
.     PERFORM PROCS
.   END REC
. END CASE
REPORT FILENAME ='REPORT3.REP'
BEFORE REPORT
.  SET BOYS GIRLS (0)            | initialise counter variables

BREAK LEVEL 1.1 (GENDER EQ 1)    | do this block for men
.  DETAIL BLOCK
.  COMPUTE BOYS = BOYS + 1
END BREAK LEVEL

BREAK LEVEL 1.2 (GENDER EQ 2)    | do this block for women
.  DETAIL BLOCK
.  COMPUTE GIRLS = GIRLS + 1
END BREAK LEVEL

AFTER REPORT| write out the grand totals
.  WRITE 'Male   count is ' BOYS
.  WRITE 'Female count is ' GIRLS
END REPORT
END RETRIEVAL
```

This branches to one of the two conditional break level blocks and increments either the variable `BOYS` or `GIRLS`. This report outputs only two lines.

```
Male   count is 12
Female count is 8
```

Example 4: Simple Breaks

The next example produces the identical report by a different method. The logic of this report is to sort the procedure table by `GENDER` and specify a break level that breaks on `GENDER`. All of the men are first in the table, followed by all of the women. A break is triggered by the first record in the table and another break is triggered when the value of `GENDER` changes from male to female. The initial block is used to initialise a counter and to get the value label for `GENDER`. The counter is incremented in the detail block and the counts are written in the at end block at the end of each group (each gender).

```
RETRIEVAL
PROCESS CASES
.  PROCESS REC EMPLOYEE
.    GET VARS GENDER
.    PERFORM PROCS
.  END REC
END CASE
REPORT FILENAME ='REPORT4.REP'/ SORT = GENDER
BEFORE REPORT
.  STRING * 6 SEX                  | variable for gender label
.  INTEGER    COUNTER              | counter variable

BREAK LEVEL 1 GENDER               | simple break on GENDER
.  INITIAL BLOCK                   | when break occurs
.    SET COUNTER (0)               | initialise counter
.    COMPUTE SEX = VALLAB(GENDER)  | get gender label

.  DETAIL BLOCK                    | for each record
.    COMPUTE COUNTER = COUNTER + 1 | increment counter

.  AT END BLOCK                    | when break is done
.    WRITE SEX ' count is ' COUNTER| output count to report

END BREAK LEVEL
END REPORT
END RETRIEVAL
```

Example 5: Detail Lines and Subtotals

Many reports contain detail lines displaying data from each procedure table record. In this example, the report is broken into two sections, by gender, and reports name, current job position (`CURRPOS`) and salary. Each section appears on a different page and reports average salary for the section. It is formatted as below, where x's stand for data.

```
Male Salary Report                          Page 1
Name                  Job Title             Salary
--------------------  --------------------  ------
xxxxx xxxxxxxxx       xxxxxxxxxxxxxxxxxxx     xxxx
xxxxxxxx xxxxxxxx     xxxxxxxxxxxxxxxxxxx     xxxx
xxxxxx xxxxxxxxx      xxxxxxxxxxxxxxxxxxx     xxxx
                                            ------
Average Male Salary                           xxxx

             - - - - new page - - -
Female Salary Report                             Page 2
Name                  Job Title             Salary
--------------------  --------------------  ------
xxxxx xxxxxxxxx       xxxxxxxxxxxxxxxxxxx     xxxx
xxxxxxxx xxxxxxxx     xxxxxxxxxxxxxxxxxxx     xxxx
xxxxxx xxxxxxxxx      xxxxxxxxxxxxxxxxxxx     xxxx
                                            ------
Average Female Salary                         xxxx
```

The requirements for this report are that the variables `GENDER, NAME, CURRPOS` and `SALARY` are in the procedure table and that the procedure table is sorted by `GENDER`. A break on gender is used to calculate the average salary and to separate the two sections of the report. Each gender section is printed on a new page by specifying a `PAGE EJECT` when the break occurs. Since the heading and average salary line change with each gender, these are also calculated at each break.

To calculate the averages, a sum and a count of the salaries is calculated. These are initialised at gender breaks, and incremented in the detail block for each record. The average is calculated at the end of each gender group in the at end block:

```
RETRIEVAL
PROCESS CASES
. PROCESS REC EMPLOYEE
.  GET VARS GENDER NAME CURRPOS SALARY
.  PERFORM PROCS
. END REC
END CASE

REPORT FILENAME = 'REPORT5.REP'    | specify output file
       SORT     = GENDER           | sort by gender

BEFORE REPORT                      | before we really start
.  INTEGER SALSUM SALCNT AVGSAL    | declare vars for stats
.  STRING * 40 HEADLINE SUBTLINE   | declare string vars
BREAK LEVEL 1 GENDER               | break on gender
. INITIAL BLOCK                    | for each new gender
.  SET SALSUM SALCNT (0)           | initialise sum and count
.  COMPUTE HEADLINE =              | construct header label
                     TRIM(VALLAB(GENDER))+
                     ' Salary Report'
.  COMPUTE SUBTLINE =              | construct subtotal label
                     'Average '+
                     TRIM(VALLAB(GENDER))+
                     ' Salary'
.  HEADING HEADLINE 44T 'Page 'PAGE| define heading
.  PAGE EJECT                      | force a newpage
. PAGE BLOCK                       | at each new page break
.  WRITE 'Name'                    | output column headers
         22T 'Job Title'
         44T 'Salary'
.  WRITE     '--------------------'| output col underlines
         22T '--------------------'
         44T '------'
. DETAIL BLOCK                     | for each table record
.  COMPUTE SALSUM = SALSUM + SALARY| increment salary sum
.  COMPUTE SALCNT = SALCNT + 1     | increment salary count
.  WRITE    NAME(A20)              | output data line
            22T [VALLAB(CURRPOS)](A20)
            44T SALARY(I6)
. AT END BLOCK                     | when gender is done
.  COMPUTE AVGSAL = SALSUM / SALCNT| average salary
.  WRITE 44T '------'              | output subt underline
.  WRITE SUBTLINE 44T AVGSAL (I6)  | output subtotal line

END BREAK LEVEL
END REPORT
END RETRIEVAL
```

Example 6: Multiple Breaks

Suppose the same report is required with a different report section for every combination of Gender and Marital Status giving four sections, married men, single men, married women and single women.

Changing the previous program to accomplish this is trivial. There are four things to do: put the Marital Status (`MARSTAT`) into the procedure table; change the header to include the marital status label; get the same label into the average salary display line; ensure that the report is broken by both Gender and Marital Status.

A new break level on Marital Status is added and Marital Status is included in the sort specification. Placing this break at a higher level than Gender without specifying any action blocks, means that it serves only to trigger the break point actions at the lower level. Exactly the same report is produced as before, except that it is broken every time either Marital Status or Gender changes. Note how few changes have been made to the program. Changed lines are marked with an '*' in the comment area.

```
RETRIEVAL
. PROCESS CASES
.   PROCESS REC EMPLOYEE
.     GET VARS MARSTAT GENDER NAME CURRPOS SALARY  |* add MARSTAT
.     PERFORM PROCS
.   END REC
. END CASE

REPORT FILENAME = 'REPORT6.REP'
           SORT =  MARSTAT GENDER |* add marstat
BEFORE REPORT                     | before we really start
.   INTEGER SALSUM SALCNT AVGSAL  | declare vars for stats
.   STRING * 40 HEADLINE SUBTLINE | declare string vars
BREAK LEVEL 1 MARSTAT             |*break on marstat to trigger
END BREAK LEVEL                   | actions at next break level
BREAK LEVEL 2 GENDER              |*break on gender (note new #)
. INITIAL BLOCK                   | for each new gender/marstat
.  SET SALSUM SALCNT (0)          | initialise sum and count
.  COMPUTE HEADLINE =             |construct header label
           TRIM(VALLAB(MARSTAT))  |*added marstat label
      +' '+TRIM(VALLAB(GENDER))
      +' Salary Report'
. COMPUTE SUBTLINE =              |construct subtotal label
          'Average'+
           TRIM(VALLAB(MARSTAT))  |*added marstat label
      +' '+TRIM(VALLAB(GENDER))+
          ' Salary'
.  HEADING HEADLINE 44T 'Page 'PAGE | define heading
.  PAGE EJECT                     | force a newpage

. PAGE BLOCK                      | at each new page or break
. WRITE     'Name'                 | output column headers
         22T 'JobTitle'
         44T 'Salary'
. WRITE     '--------------------'| outputcol underlines
        22T '--------------------'
        44T '------'
. DETAIL BLOCK                     | for each proc table record
.  COMPUTE SALSUM = SALSUM + SALARY | increment salary sum
.  COMPUTE SALCNT = SALCNT +1      | increment salary count
.  WRITE     NAME(A20)             | outputdata line
         22T [VALLAB(CURRPOS)](A20)
         44T SALARY(I6)
. AT END BLOCK                     | when break group is done
.  COMPUTE AVGSAL =SALSUM / SALCNT| calculate average salary
.  WRITE 44T '------'              | output subt underline
.  WRITE SUBTLINE 44T AVGSAL (I6) | output subtotalline
END BREAK LEVEL
END REPORT
END RETRIEVAL
```

```
Married Male Salary Report                 Page 1
Name                  Job Title              Salary
--------------------  --------------------  ------
John D Jones          Technician              2150
James A Arblaster     Sr Technician           2650
Jack Brown            Sr Administrator        3350
                                            ------
Average Married Male Salary                   2862


Married Female Salary Report               Page 2
Name                  Job Title              Salary
--------------------  --------------------  ------
Carol F Safer         Sr Chemist              1650
Bonnie Rosen          Director                3200
```

…….

…….

Example 7: A Branched Report with Differing Formats

Consider a report that is broken into sections, where the sections contain different data depending on the value of some variable. For example an employment history by employee, with certain data for men and other data for women.

This is a case where a conditional or branched report is required. When reporting females, it follows one path through the report code and when reporting males, it takes another. The layout of the report is as follows:

For Females:

```
Employee Report  date                             Page  x

ID Number      xx
Name           xxxxxxxxxx
Gender         Female
Date of Birth  xxxxxxxxxxxxx

               Title            Salary
               xxxxxxxxxxxx       xxxx
               xxxxxxxxxxxx       xxxx
               xxxxxxxxxxxx       xxxx

               xxxxxxxxxx         xxxx
               xxxxxxxxxx         xxxx
               xxxxxxxxxx         xxxx
```

For Males:

```
Employee Report  date                             Page  x

ID Number      xx
Name           xxxxxxxxxx
Gender         Male
Date of Birth  xxxxxxxxxxxxx

Position  Title                    Date      Salary    Rating
   xx     xxxxxxxxxxxxxxxx         xxxxxxxx    xxxx    xxxxxxxxx
                                   xxxxxxxx    xxxx    xxxxxxxxx

   xx     xxxxxxxxxxxxxxxx         xxxxxxxx    xxxx    xxxxxxxxx
                                   xxxxxxxx    xxxx    xxxxxxxxx
```

Variables are read from three record types in the `COMPANY` database. The report is a complete salary history of each employee, including salaries and dates from two different record types. The initial program puts the values of `STARTSAL` from the `OCCUP` record type and `NEWSAL` from the `REVIEW` record type into a single variable. The same thing happens with `STARTDAT` and `REVDATE`.

In the report specification, there are three break levels, one to produce a new report section for each employee, another to deal with the different format for men and women and the last to group output lines by position. The first break level is a simple break on the employee `ID` number. The second level is a conditional break on the value of Gender that formats the data differently for each sex.

Note in the format for the men, some detail lines have position data and others do not. The lines that have values for position come from the `OCCUP` record and those that don't come from `REVIEW` records at the position printed in the previous line. This requires different processing depending on which record type the data came from with one of two WRITE commands with different line formats. Following is the code for the report.

```
RETRIEVAL
PROCESS CASES
.  PROCESS REC 1
.  GET VARS ID NAME GENDER BIRTHDAY | rec 1 vars to proc rec
.  END REC

.  PROCESS REC 2
.  GET VARS POSITION                | put position in proc rec
.  GET VARS DATE     SALARY =       | put revdate and startsal
         STARTDAT STARTSAL          | into vars date, salary
.  SET RECTYPE (2)                  | set record typeflag
.  PERFORM PROCS                    | copy rec 2 to proc table

.    PROCESS REC 3 VIA (POSITION)   | get rec 3 using position
.      GET VARS RATING              | put rating in procedure rec
.      GET VARS DATE    SALARY =    | put revdate and newsal
               REVDATE NEWSAL       | into vars date, salary
.      SET RECTYPE (3)              | set record type flag
.      PERFORM PROCS                | copy rec 3 to proc table
.    END REC

.  END REC
END CASES
REPORT FILENAME = REPORT7.REP /SORT = ID GENDER  POSITION DATE
BEFORE REPORT
.  HEADING BLOCK 7
.   WRITE 'Employee Report' 2X DATE 60T 'Page' PAGE(I3)//
.   WRITE 'ID Number'    17T ID
.   WRITE 'Name'17T NAME
.   WRITE 'Gender '      17T [VALLAB(GENDER)]
.   WRITE 'Date of Birth' 17T BIRTHDAY(DATE'Mmm DD, YYYY')
.  END HEADING BLOCK
. BREAK LEVEL 1 ID                  | break on each employee
.  INITIAL BLOCK
.   PAGE EJECT                      | put each employee on a new page
END BREAK LEVEL
BREAK LEVEL 2.1, (GENDER EQ 1)      | take this branch for men
```

```
.  PAGE BLOCK                       | at break or page eject
.   BLANK LINES 2                   | output 2 blank lines
.   WRITE 'Position Title' 32T 'Date' 42T 'Salary' 50T 'Rating'
END BREAK LEVEL
BREAK LEVEL 3A, POSITION            | for every new position
. DETAIL BLOCK                      | for every procedure rec
. IFTHEN (RECTYPE EQ 2)             | do following if rec 2
.    WRITE POSITION(I8),            | output data line with
          10T [VALLAB(POSITION)] (A20)  | position information
          32T DATE(DATE'MM/DD/YY')
          42T SALARY(I6)
          50T 'n/a'                 | no rating data in rec 2
.  ELSEIF (RECTYPE EQ 3)            | do following if rec 3
.    WRITE 32T DATE (DATE'MM/DD/YY')| output data without
           42T SALARY(I6)           | position data
           50T [VALLAB(RATING)]
.  END IF
. AT END BLOCK                      | before next Position
.   BLANK LINES 1                   | output a blank line
END BREAK LEVEL
BREAK LEVEL 2.2, (GENDER EQ 2)      | take this branch for women
.  PAGE BLOCK                       | at break or page eject
.   BLANK LINES 2                   | output 2 blank lines
.   WRITE 20T 'Title' 42T,'Salary'  | output column headers
END BREAK LEVEL
BREAK LEVEL 3B,POSITION             | break on position
. DETAIL BLOCK                      | for each procedure record
.    WRITE 20T [VALLAB(POSITION)](A20) 42T SALARY(I6)
. AT END BLOCK                      | before next position
. BLANK LINES 1                     | output a blank line
END BREAK LEVEL
END REPORT
END RETRIEVAL
```

```
Employee Report  Jan 03, 2006                              Page  1


ID Number       1
Name            John D Jones
Gender          Male
Date of Birth   Jan 08, 1968


Position Title                 Date      Salary  Rating
       4 Laborer               02/04/03    1500  n/a
                               04/05/03    1600  Good
                               06/05/03    1650  Very Good

       5 Technician            10/09/04    2000  n/a
                               12/09/04    2100  Good
                               02/04/05    2150  Very Good

Employee Report  Jan 03, 2006                              Page  2


ID Number       2
```

```
Name            James A Arblaster
Gender          Male
Date of Birth   Dec 02, 1962


Position Title                 Date      Salary  Rating
       6 Sr Technician         01/12/03    2500  n/a
                               03/16/03    2550  Acceptable
                               04/27/03    2600  Good
                               08/08/03    2650  Very Good

Employee Report  Jan 03, 2006                             Page  3


ID Number       3
Name            Mary Black
Gender          Female
Date of Birth   Aug 05, 1973


                   Title                 Salary
                   Chemist                 2750
                   Chemist                 2800
                   Chemist                 2850
                   Chemist                 2900

                   Sr Chemist              3000
                   Sr Chemist              3100
                   Sr Chemist              3150
```

Example 8: Underlines

Reports frequently contain underlined column headers. Typically, these are produced with a WRITE commands that contains the appropriate number of dashes as quoted strings. An alternative is to have a string variable that contains nothing but dashes and to use that variable repeatedly on the `WRITE` statement, controlling the number of dashes with the formatting options. Consider a typical `WRITE` statement with quoted dashes:

```
WRITE '----------' 2X '-----' 2X '------------' 2X '-------'
```

If there is a variable `UL`, filled with dashes, this gives the same output with:

```
WRITE UL(A10) 2X UL(A5) 2X UL(A13) 2X UL(A7)
```

Following is an example of part of a report program using this technique.

```
BEFORE REPORT
.  STRING UL
.  SET UL ('--------------------')

BREAK LEVEL 1
.  PAGE BLOCK
.  WRITE 'Employee Name' 25T 'Job Title'   50T 'Salary'
.  WRITE  UL(A20)        25T  UL(A20)50T  UL(A6)
.  DETAIL BLOCK
.  WRITE     NAME (A20)  25T JOBTITLE(A20) 50T SALARY(I6)
END BREAK LEVEL
```

Example 9: Totals and After Report

Sometimes reports need a data value calculated in the pre-report part of the program, which is needed for every procedure record.

Suppose, for example, that the percent of the total payroll each employee's salary represents is to be written. The calculation of this percentage needs the sum of all the salaries and each employee's salary. Getting the sum of salaries simply requires adding up salaries in the retrieval. The problem is that in the report any given procedure table record only has a partial sum. The basic technique is to store the sum in a new variable (which is not part of the procedure table) at the end of the retrieval section in an `AFTER RETRIEVAL` block and then to access it in the report.

Remember that unless the Procedure Table is sorted, the report operates as `PERFORM PROCS` sends each procedure table record. Since the percent cannot be calculated until all the database records have been processed, delay execution with a dummy sort. Create a dummy variable (`DUMMY`) that always has the value 0 for this purpose. If the report is sorted for other reasons, this is unnecessary.

The following program produces the required results:

```
RETRIEVAL
SET TOTSAL DUMMY (0)                | initialise total salary
PROCESS CASES
.  PROCESS REC EMPLOYEE
.  GET VARS NAME SALARY
.  COMPUTE TOTSAL = TOTSAL + SALARY | increment total salary
.  PERFORM PROCS
.  END REC
END CASE
AFTER RETRIEVAL
SALTOT = TOTSAL                     | put total in variable
REPORT FILENAME = REPORT8.REP /
           SORT = DUMMY             | do a dummy sort to delay
execution

BREAK LEVEL 1
DETAIL BLOCK
COMPUTE PCT = (SALARY/SALTOT) * 100 | calculate percentage
WRITE    NAME   (A20)               | output the data
      2X SALARY (I4)
      2X PCT    ('999.99')
END BREAK LEVEL
```

Example 10: Using External Files and Edit Buffers

Full Report includes the capability of reading from and writing to files and of using edit buffers. In the previous example, it would have been possible to write the sum to a file or edit buffer and then read it back.

If a file is written in the first part of the program and then read from in the report, delay execution of the report with a sort parameter and make sure that the file is closed at the end of the retrieval section and then re-opened before reading it in the report section. Edit buffers have the advantage of being randomly accessed for both read and write operations, though the data can only be retrieved in strings, a single line at a time.

Files may be written from within the report section as well as read. Consider a report from census data in which detail lines are reported for counties with summary data by state as a subtotal line. A summary section might be required which reprints all the subtotal (state) data on a single page at the end of the report. This could be done by storing all the data in a report array and then printing it out again in the `AFTER REPORT` section.

Since the subtotal lines are formatted for its `WRITE` statement, it takes almost no extra effort to write once to the report and a second time to a file. In the `AFTER REPORT` section read the lines of text in the file and print them. The general structure of the report would be:

```
BEFORE REPORT
STRING * 80 FILETEXT
INTEGER     STATE1 TO STATE5            | declare state total vars
OPEN TFILE.TXT WRITE
BREAK LEVEL 1 STATE
.INITIAL BLOCK
.   SET STATE1 TO STATE5(0)             | initialise state totals
.   AT END BLOCK
.   WRITE             STATE1 TO STATE5  | write totals to report
.   WRITE (TFILE.TXT) STATE1 TO STATE5  | write totals to file
END BREAK LEVEL
BREAK LEVEL 2 COUNTY
.  DETAIL BLOCK
.  WRITE COUNTY1 COUNTY2 COUNTY3 COUNTY4 COUNTY5
                                        | increment state totals
                                        | (e.g.  STATE1 = STATE1
+COUNTY1)
END BREAK LEVEL
AFTER REPORT
CLOSE TFILE.TXT                         | close file
OPEN  TFILE.TXT READ                    | open file for read
PAGE EJECT                              | start on a new page
WRITE 30T 'State Summary Data' //       | pagetitle
LOOP                                    | loop thru records
```

```
.  READ (TFILE.TXT,ERR=EOF)FILETEXT(A80)| read a line from file
.  WRITE FILETEXT                       | write line to report
END LOOP
EOF:
END REPORT
```

SAS Save File

SAS® software is a data analysis and reporting tool published by SAS Institute, Inc. of Cary, North Carolina.

The `SAS SAVE FILE` procedure generates files in "exportable" (text) format.

These files include procedure table data and schema information from the PQL program, including value labels, variable labels and missing value indicators.

```
SAS SAVE FILE EXPORT   = filename1 , filename2
    [ BOOLEAN   = (logical_expression ) ]
    [ FORMAT = 'formatprefix']
    [ LRECL = n]
    [ NOLABELS ]
    [ SAMPLE    = fraction ]
    [ SORT      = [ (n) ]  variables [(A)|(D)], ...]
    [ VARIABLES = varlist | ALL]
```

`EXPORT`	Creates two text files. The first is the `SAS` control statement file. It contains a `SAS DATA` step including all appropriate `SAS` commands such as `MISSING, INFILE, INPUT` and `LABEL` and a `PROC FORMAT` step containing the value labels. If the `NOLABELS` clause is specified, the `PROC FORMAT` step is not generated. The second file is a fixed format data file, containing the data from the procedure table. This file is referenced in the `DATA` step of the first file.
`BOOLEAN`	Specifies which procedure table records are used by the procedure. The procedure table records for which the logical expression is true are used by the procedure. If this option is not specified, all procedure table records are used.
`FORMAT`	The names assigned to the value label formats in the `PROC FORMAT` are generated names in the format TnX where 'T' and 'X' are the letters used and n is a sequence number from 1 upwards for each variable requiring a format. Format names for character variables begin with a $ sign. If the SAS analyst combines multiple datasets, the same generated name would be used on each dataset and this would cause conflicts. Specify a different format prefix for each SIR/XS procedure to ensure unique generated format names. The specified prefix is used instead of the default 'T'. The final generated name must not exceed 32 characters.
`LRECL`	An `LRECL` parameter is generated on the `SAS INFILE` command.

	Use the `LRECL` parameter to set this value. It must be greater than 50.
`NOLABELS`	Specifies that variable and value labels are not written to the SAS file. This can save significant processing time if many labels are involved.
`SAMPLE`	Specifies that a random sample of the procedure table records are used by the procedure. The **fraction** specifies the percent of records used and is specified as a positive decimal number less than or equal to 1 (one). .25, for example specifies that a 25% sample be used.
`SORT`	Specifies the sequence of the output. **n** is an integer that specifies the maximum number of records to be sorted. The default for this parameter is either the number of records in the database or the value specified in the sortn parameter and need only be specified if the number of records in the procedure table is greater than the default. The procedure table is sorted by the specified variables in variable list order. A variable name followed by (**A**) or (**D**) specifies that for that variable the sort is in **A**scending order (the default) or in **D**escending order.
`VARIABLES`	Specifies the procedure variables that are written to the output file. The order in which they are specified is the order in which they appear in the output file. This corresponds to the order in which they are declared in the `SAS DATA` step. If this option is not specified, the default variables are output.

A summary report is produced that lists the variables in the files, the name of the files and other information.

Example 1: SAS

The following produces a `SAS` file. The files created by this run are text and can be edited. The program creates a procedure table record for each employee and contains the computed time spent in the current position (`TCURRENT`) and the variables `ID, GENDER, MARSTAT` and `CURRPOS` from the database.

```
RETRIEVAL
INTEGER TCURRENT
PROCESS CASES ALL
.  PROCESS REC 1
.    GET VARS ID GENDER MARSTAT CURRPOS
.    PROCESS REC 2 VIA (CURRPOS)
.         COMPUTE TCURRENT = TODAY(0) - STARTDAT
.    END PROCESS REC
.    PERFORM PROCS
.  END PROCESS REC
END PROCESS CASE
```

```
SAS SAVE FILE EXPORT = SASPROC.TXT , SASDATA.DAT
END RETRIEVAL
```

SAVE TABLE

`SAVE TABLE` creates a table from data in the VisualPQL procedure table. If a table of the same name already exists, it can be replaced. Tables can be accessed by PQL, SQL and FORMS. Tables are stored on tabfiles.

The `SAVE TABLE` procedure creates a table and populates it with data. The definitions for the columns in the table are taken from the schema and the values for each row are taken from the procedure table records. One table row is created for every procedure table record.

The tabfile to contain the table must exist before the procedure is run. The `CREATE TABFILE` command in `SQL` and `DBMS` and the `Create Tabfile...` option from the `Tabfile` menu can be used to create tabfiles.

```
 SAVE TABLE [tabfile_name.]table_name
     [ FILENAME   = fileid ]
     [ VARIABLES  = varlist | ALL  ]
     [ USERS      =  group[/pword][.user[/pword]]... ]
     [ REPLACE ]
     [ SORT       = [ (n) ]  variable [(A)|(D)], ...]
     [ BOOLEAN    = ( logical_expression )  ]
     [ SAMPLE     = fraction ]
```

`tabfile name`	Specify the tabfile to save the table on. If a tabfile name is not specified, the current default tabfile is used. If the tabfile is not connected at the time the VisualPQL program is run, SIR/XS attempts to connect the tabfile. Note that there is no provision to specify an `IDENTIFIED BY` clause so, if a tabfile has groups or users defined, the tabfile must be connected at run time with a `CONNECT TABFILE` command.
`table name`	The table name that is created. This must be specified.
`FILENAME`	Specifies the physical filename of the tabfile. If the physical filename is the same as the tabfile name (appended with '.tbf' or the tabfile is connected, this clause need not be specified.
`VARIABLES`	Specifies the procedure variables that are the columns (variables) of the table. The order in which they are specified is the column order of the table. If this option is not specified, the default variable list is used. All variable schema information is carried over to the table that is created.
`USERS`	Specifies a list of groups or users in groups for tabfiles that have permissions. For the table being created, these groups or users are

	granted all permissions. Other users have no permissions on the table. If the group does not exist, it is created. Passwords may be specified at the group or user level. If the group(.user) already exists, there is no need to specify passwords. If the group(.user) already exists and passwords are specified, these become the current passwords. This is the equivalent of the SQL command `GRANT ALL` for the list of users where the permissions were granted by the group(.user) specified to connect the tabfile. Regardless as to whether this clause is specified or not, the group(.user) specified to connect the tabfile has full permissions on the table. To create a `PUBLIC` table on a tabfile with permissions, specify `USERS = PUBLIC`. (For tabfiles without permissions, all tables are public and the `USER` clause has no affect.) See Permissions for further details on tabfile permissions.
`REPLACE`	The `REPLACE` keyword gives permission to overwrite an existing table of the same name if it exists. If the option is omitted and the table exists, the program terminates with an error message.
`SORT`	Specifies the order in which the procedure table records are sorted and written to the tabfile table. **n** specifies the maximum number of records to be sorted. The default for this parameter is either the number of records in the database or the value specified in the sortn parameter and need only be specified if the number of records in the procedure table is greater than the default. The procedure table is sorted by the specified variables in variable list order. A variable name followed by (**A**) or (**D**) specifies that for that variable the sort is in **A**scending order (the default) or in **D**escending order.
`BOOLEAN`	Specifies which procedure table records are used by the procedure. The procedure table records for which the logical expression is true are used by the procedure. If this option is not specified, all procedure table records are used.
`SAMPLE`	Specifies that a random sample of the procedure table records are used by the procedure. The **fraction** specifies the percent of records used and is specified as a positive decimal number less than or equal to 1 (one). .25, for example specifies that a 25% sample be used.

Example: Creating a Database Subset

Since it is very simple to create table variables with the same definitions as record variables in the database, tables can be used as a database subset. The following program creates a table that is identical to a database record type.

```
RETRIEVAL
```

```
PROCESS CASES
.  PROCESS REC 1
.    GET VARS ALL
.    PERFORM PROCS
.  END REC
END CASE
SAVE TABLE  REC1BKUP
END RETRIEVAL
```

```
*** Table replaced ......... REC1BKUP
*** on tabfile ............. SQLTAB
*** no of rows ............. 20
*** no of columns .......... 4
*** variable names ......... 1. NAME
                             2. GENDER
                             3. MARSTAT
                             4. SSN
                             ........
```

After the retrieval is run, the procedure records are passed to the `SAVE TABLE` procedure. The default tabfile `SQLTAB` is used. A new table called `REC1BKUP` is created. (The `REPLACE` option is used to overwrite the old table if it exists.) After the retrieval is finished, a summary report is produced showing the tabfile written to, the number of rows and columns written, and the variable names written to the table.

SIR Save File

The `SIR SAVE FILE` procedure creates a sequential file in the same format as that produced by the `UNLOAD` and `SIR SUBSET` utilities. Reload this file to create a new SIR/XS database or use the `SIR MERGE` utility to merge this file into an existing database.

The output records all have the same format and record type. Variables carry with them all variable schema information, including valid values, variable ranges, missing values, variable labels and value labels.

`SIR SAVE FILE` does not produce an `INPUT FORMAT` or `DATA LIST` with input columns. Using this procedure is equivalent to having defined a record schema using the variable declarations of internal storage types and sizes. Once the new database has been reloaded, the schema may be modified to specify input columns for Batch Data Input use.

Many of the options on this command specify schema information for the database that is created when the file is reloaded.

Note that the `SIR SAVE FILE` procedure cannot be run from a `PROGRAM`. It must be run from a database `RETRIEVAL`.

```
SIR SAVE FILE
         { {DATABASE  = database_name | FILENAME = filename } |
            DATABASE = database_name   FILENAME  = filename  }
            RECTYPE = recnum [, recname]

     [ {CASEID   = varname [(D | A)] | NOCASEID }]
     [ VARIABLES = var_list ]
     [ SORTIDS   = variable [(D | A)] ...  ]
     [ NOFCASES  = number  ]
     [ RECSCASE  = number  ]
     [ MAXTYPES  = number  ]
     [ MAXRECS   = number  ]
     [ MAXKEYSZ  = number  ]
     [ SAMPLE    = fraction]
     [ BOOLEAN   = (logical_expression) ]
     [ NOLABELS ]
```

`DATABASE`	Specify either the name of the new database or a filename or both. If not specified, the name defaults to the attribute name assigned to the file. The name must comply with the rules for naming a SIR/XS database. The password for the new database is the same as the password of the original database.
`FILENAME`	Specify the name of the output file. If not specified, a file with the same name as the `DATABASE` clause is written.

RECTYPE	Specifies the record number of the records when reloaded in the new database. This clause is required. The record name is optional, if it is omitted, a default name in the form RECn is assigned.
VARIABLES	Specifies the procedure variables to form the new database record type. The order in which they are specified becomes the data list order.
CASEID	Specifies the procedure variable used as the Case Id in the new database. If the `CASEID` clause is omitted, the case id of the original database is used for the new database. The original case id must be included as a procedure variable. To alter the default sort sequence, specify **D** for Descending or **A** for Ascending in parentheses following the variable name.
NOCASEID	Specifies that a caseless database is created.
SORTIDS	Specifies the *Key Fields* of the new database record type. By default, the records are sorted in **A**scending order by Key Field (Sort ID) value. To alter the default sort sequence, specify **D** for Descending or **A** for Ascending in parentheses following the variable name.
BOOLEAN	Specifies which procedure table records are used by the procedure. The procedure table records for which the logical expression is true are used by the procedure. If this option is not specified, all procedure table records are used.
SAMPLE	Specifies that a random sample of the procedure table records are used by the procedure. The **fraction** specifies the percent of records used and is specified as a positive decimal number less than or equal to 1 (one). .25, for example specifies that a 25% sample be used.
MAXKEYSZ	Specifies the maximum key size for the new database. This clause is the equivalent of the database parameter `MAX KEY SIZE`. If this clause is omitted, the value from the original database or the size implied by this record type (if it is larger) is used.
MAXRECS	Specifies the maximum number of records per case of any record type in the new database. This clause is the equivalent of the database parameter `MAX REC COUNT`. If this clause is omitted, the value from the original database is used.
MAXTYPES	Specifies the maximum number of record types in the new database. This clause is the equivalent of the database parameter `MAX REC TYPES`. If this clause is omitted, the `MAX REC TYPES` of the original database is used.
NOFCASES	Specifies the maximum number of cases in the new database. This clause is the equivalent of the `N OF CASES` database parameter. If this clause is omitted, the `N OF CASES` specification from the

	original database is used.
`RECSCASE`	Specifies the average number of records per case in the new database. This clause is the equivalent of the `RECS PER CASE` database parameter. If this clause is omitted, the `RECS PER CASE` specification from the original database is used.
`NOLABELS`	Specifies that variable and value labels for the variables used in the procedure are not transferred to the new database.

Example

The requirement is to extract from an educational database, a new database where a case includes all students in a grade level and record type 10 contains essential information on each student.

```
SIR SAVE FILE    DATABASE  = EDUCAT /
                 FILENAME  = SAVE /
                 CASEID    = GRADE /
                 RECTYPE   = 10 , STUDENTS/
                 SORTIDS   = STUDNO /
                 VARIABLES = STUDNO , GRADE , AGE , SEX ,IQ
```

SPREAD SHEET

The `SPREAD SHEET` procedure displays summary data in a format similar to that used by spreadsheets. This procedure does not produce an intermediate file, it writes directly to the SIR/XS spreadsheet interface.

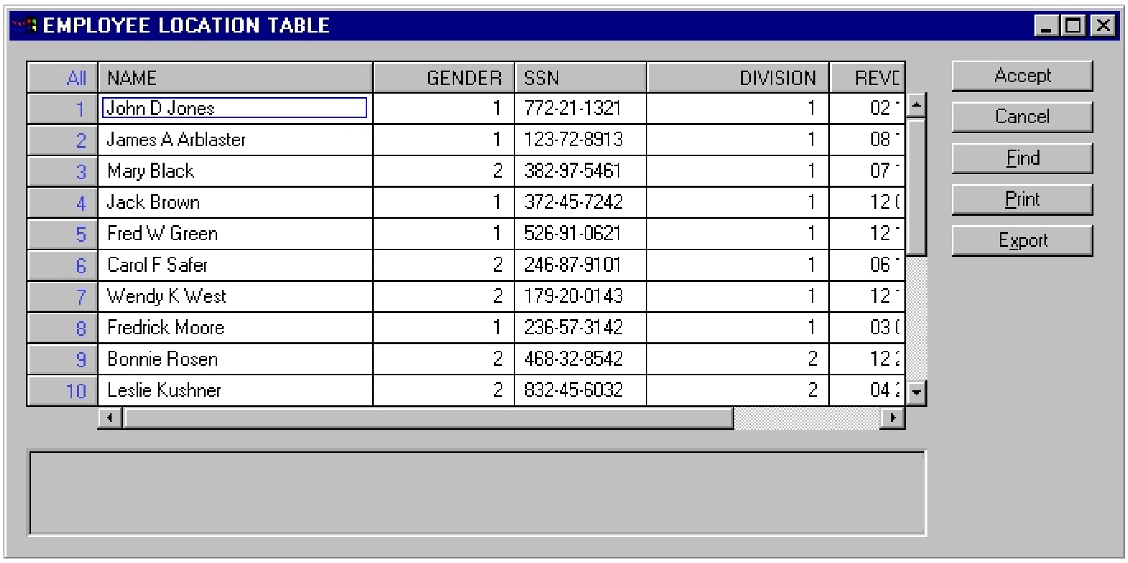
```
SPREAD SHEET
         [ VARIABLES = varlist ]
         [ SORT      = [(n)]variable [(A)|(D)], ...]
         [ BOOLEAN   = ( logical_expression ) ]
         [ SAMPLE    = fraction]
         [ TITLE     = 'spreadsheet title']
```

`VARIABLES`	Specifies the procedure variables to display. Specify the variables in the order in which they are to appear. If this option is not specified or the keyword `ALL` is specified, the default variables are output.
`SORT`	Specifies the sequence of the output. therefore the spreadsheet rows are sorted.**n** is an integer that specifies the maximum number of records to be sorted. The default for this parameter is either the number of records in the database or the value specified in the sortn parameter and need only be specified if the number of records in the procedure table is greater than the default. The procedure table is sorted by the specified variables in variable list order. A variable name followed by (**A**) or (**D**) specifies that for that variable the sort is in **A**scending order (the default) or in **D**escending order.
`BOOLEAN`	Specifies which procedure table records are used by the procedure. The procedure table records for which the logical expression is true are used by the procedure. If this option is not specified, all procedure table records are used.
`SAMPLE`	Specifies that a random sample of the procedure table records are used by the procedure. The **fraction** specifies the percent of records used and is specified as a positive decimal number less than or equal to 1 (one). .25, for example specifies that a 25% sample be used.
`TITLE`	Specifies a title for the spreadsheet.

Example

```
RETRIEVAL
. PROCESS CASES
.   GET VARS ID
.   PROCESS RECORD 1
.     GET VARS NAME GENDER SSN CURRPOS
.     PROCESS RECORD 2 WITH (CURRPOS)
.       GET VARS DIVISION
.       PERFORM PROCS
.       EXIT RECORD
.     END RECORD
.   END RECORD
. END CASE
SPREAD SHEET  VARIABLES = NAME GENDER SSN DIVISION
              TITLE     = "EMPLOYEE LOCATION TABLE"
END RETRIEVAL
```

The summary data is displayed as a spreadsheet which looks something like:



All	NAME	GENDER	SSN	DIVISION	REV
1	John D Jones	1	772-21-1321	1	02
2	James A Arblaster	1	123-72-8913	1	08
3	Mary Black	2	382-97-5461	1	07
4	Jack Brown	1	372-45-7242	1	12
5	Fred W Green	1	526-91-0621	1	12
6	Carol F Safer	2	246-87-9101	1	06
7	Wendy K West	2	179-20-0143	1	12
8	Fredrick Moore	1	236-57-3142	1	03
9	Bonnie Rosen	2	468-32-8542	2	12
10	Leslie Kushner	2	832-45-6032	2	04

You can view, edit and save this data as text, html or in EXCEL® format.

SPSS Save File

SPSS®, the *Statistical Package for the Social Sciences*, is a multi-purpose data analysis and reporting tool. The `SPSS SAVE FILE` procedure provides an interface to this product. (`SPSSX` is a synonym for `SPSS`.)

`SPSS SAVE FILE` creates a portable file that can be accessed by SPSS. One SPSS case is created for each procedure table record generated by the VisualPQL program. In addition to the data, the dataset contains value and variable labels and missing value indicators.

```
SPSS SAVE FILE FILENAME = filename
     [ BOOLEAN = ( logical_expression ) ]
     [ NOLABELS ]
     [ PORTABLE ]
     [ SAMPLE    = fraction]
     [ SHORTNAME]
     [ SORT      = [ (n) ] variable [(A)|(D)] , ...]
     [ VARIABLES = varlist | ALL  ]
     [ WEIGHT    =  varname  ]
```

`FILENAME`	Specifies the filename created by the procedure.
`BOOLEAN`	Specifies which procedure table records are used by the procedure. The procedure table records for which the logical expression is true are used by the procedure. If this option is not specified, all procedure table records are used.
`NOLABELS`	Specifies that variable and value labels are not copied to the `SPSS SAVE FILE`. This can save time if many labels are involved.
`PORTABLE`	Specifies that the output file is in portable file format. By default, the output file is in system file format.
`SAMPLE`	Specifies that a random sample of the procedure table records are used by the procedure. The **fraction** specifies the percent of records used and is specified as a positive decimal number less than or equal to 1 (one). .25, for example specifies that a 25% sample be used.
`SHORTNAME`	Specifies that a maximum of eight characters is used for variable names output to SPSS. Truncation occurs after eight characters. If any names are truncated, these are reported on the summary report. Note. If variables have SIR/XS non-standard names, these may not be acceptable SPSS names. SIR/XS does not check this and does not change names. If a truncated name or non-standard name is not the required SPSS name, specify an alternative name that is a valid

	SPSS name on the `VARIABLES` clause. e.g.
	`VARIABLES = Long_Name_Salary AS SALARY or`
	`VARIABLES = {Special Name Salary} 'Salary'`
	This option also truncates variable labels to 40 characters.

`SORT`	Specifies the sequence of the output. **n** is an integer that specifies the maximum number of records to be sorted. The default for this parameter is either the number of records in the database or the value specified in the sortn parameter and need only be specified if the number of records in the procedure table is greater than the default. The procedure table is sorted by the specified variables in variable list order. A variable name followed by (**A**) or (**D**) specifies that for that variable the sort is in **A**scending order (the default) or in **D**escending order.
`VARIABLES`	Specifies the procedure variables to write to the output file. The order in which variables are specified is the order in which they appear in the output file. If this option is not specified, the default variables are output.
`WEIGHT`	Specifies the variable that is the `CASWGT` variable which provides a permanent weighting of the cases in the SPSS file. The default `WEIGHT` is 1.

Notes on SPSS Files

- The basic numeric unit is an 8 byte real.
- The file uses base 30 arithmetic to represent all numbers.
- Strings are represented in 8 byte units. A short string is one that is less than or equal to 8 characters. For string variables that are greater than 8 characters, dummy variables are generated for every 8 characters.
- Undefined string variables are represented by blanks.
- Undefined numeric values and blank missing numeric values are represented by the SPSS system missing value (see `SYSMIS` in SPSS).
- SPSS missing value ranges cannot be used.

Example

```
RETRIEVAL
PROCESS CASES ALL
AUTOSET
PROCESS REC EMPLOYEE
. GET VARS ALL
. PROCESS REC OCCUP WITH (CURRPOS)
.   GET VARS ALL
. END PROCESS REC
. PERFORM PROCS
END PROCESS REC
END PROCESS CASE

SPSS SAVE FILE
       FILENAME  = 'EMPLOYEE.DAT' /
       VARIABLES = ALL
END RETRIEVAL
```

When a VisualPQL program with an SPSS interface procedure is run, a summary report is produced and has the following form:

```
SPSS SAVE FILE - System file
----------------------------

Written to:             'C:\sir2004\alpha\employee.dat'
Number of records:      20
User variables:         14

Variables in save file order
----------------------------

NAME                              GENDER
MARSTAT                           SSN
BIRTHDAY                          EDUC
NDEPENDS                          CURRPOS
SALARY                            CURRDATE
POSITION                          STARTDAT
STARTSAL                          DIVISION

End of SPSS report
------------------
```

SYSTAT Save File

The `SYSTAT SAVE FILE` procedure creates a binary file in `SYSTAT`® internal format that contains data and schema information produced in a VisualPQL program. `SYSTAT` is a statistical package that runs on PCs.

```
SYSTAT SAVE FILE    FILENAME = filename
   [ VARIABLES = var_list | ALL  ]
   [ SORT      = [(n)] variable [(A)|(D)], ...]
   [ BOOLEAN   = ( logical_expression )  ]
   [ SAMPLE    = fraction]
```

`FILENAME`	Specifies the filename created by the procedure.
`VARIABLES`	Specifies the procedure variables that are written to the output file. The order in which variables are specified is the order in which they appear in the output file. If this option is not specified, the default variables are output.
`SORT`	Specifies the sequence of the output. **n** is an integer that specifies the maximum number of records to be sorted. The default for this parameter is either the number of records in the database or the value specified in the sortn parameter and need only be specified if the number of records in the procedure table is greater than the default. The procedure table is sorted by the specified variables in variable list order. A variable name followed by (**A**) or (**D**) specifies that for that variable the sort is in **A**scending order (the default) or in **D**escending order.
`BOOLEAN`	Specifies which procedure table records are used by the procedure. The procedure table records for which the logical expression is true are used by the procedure. If this option is not specified, all procedure table records are used.
`SAMPLE`	Specifies that a random sample of the procedure table records are used by the procedure. The **fraction** specifies the percent of records used and is specified as a positive decimal number less than or equal to 1 (one). .25, for example specifies that a 25% sample be used.

- *Note:* All numeric variables are written as 8 byte reals. All string variables are written as 12 character strings. Long strings are truncated and short strings are blank filled.

Example

The following example creates a Systat binary file with one record per employee and the variables `ID, GENDER, CURRPOS` and `SALARY`. The records are sorted by Salary in descending order.

```
RETRIEVAL
PROCESS CASES
.  PROCESS REC EMPLOYEE
.  GET VARS ID GENDER CURRPOS SALARY
.  PERFORM PROCS
.  END REC
END CASE
SYSTAT SAVE FILE FILENAME = SYSTAT.DAT /
                 SORT     = SALARY (D)
END RETRIEVAL
```

After the retrieval finishes, the following summary report is displayed.

```
SYSTAT SAVE FILE SUMMARY REPORT
-------------------------------

SYSTAT FILENAME .......SYSTAT.DAT

NO OF DATA RECORDS ..... 20
NO OF COLUMNS .......... 4

COLUMN NAME (TYPE)

ID (REAL)           GENDER (REAL)       CURRPOS(REAL)
SALARY (REAL)

SYSTAT SAVE FILE COMPLETE
```

Tabulate

The TABULATE procedure selects and summarises data. It then produces cross-tabulations with summary data, statistics or percentages. See syntax. A wide variety of cross-tabulations can be produced:

- One, two and three dimensional tables can be produced; the first dimension being columns across a page, the second being rows down a page and the third being different sections. These three dimensions are known as headers (for columns), stubs (for rows) and wafers (for sections).
- Control variables, with discrete values, and observation variables, with continuous values, can be used.
- Totals, means, minimums, maximums, standard deviations, medians, quantiles and percentages can be specified with flexibility in choosing percent 'base' cells, for various combinations of wafer, row and column percentages.
- Wafer, stub and header labeling can be controlled with use of variable labels and value labels. There is automatic segmentation of labels and adjustment of table size parameters to fit. Tables too large to fit on a single physical page are broken across pages and page headings can be specified.
- Output can be in standard format or produced as *html* suitable for viewing and printing by software that uses that format such as browsers or Microsoft Word®.

Syntax

```
TABULATE
     FILENAME = filename
     HEADER   = (expression)
    [STUB     = (expression)]
    [WAFER    = (expression)]
```

Record Filtering

```
  BOOLEAN     = (boolean expression)
  SAMPLE      = fraction
  WEIGHT      = varname
```

Cell Statistics

```
  TOTAL      = varname ['label']...
  COUNT      = varname ['label']...
  CSS        = varname ['label']...
  CV         = varname ['label']...
  CVERR      = varname ['label']...
  MAXIMUM    = varname ['label']...
  MEAN       = varname ['label']...
  MEDIAN     = varname ['label']...
  MINIMUM    = varname ['label']...
  MISSING    = varname ['label']...
  NORMALIZED = varname ['label'](n1)['label1'],(n2)['label2']...
  PERCENT    = varname ['label']...
  QUANTILE   = varname ['label'] n ['label1','label2'...]...
  QUANTILE   = varname ['label'](n1)['label1'],(n2)['label2']...
  RANGE      = varname (lo,hi) ['label']...
  STDERR     = varname ['label']...
  STDEV      = varname ['label']...
  TSTATISTIC = varname ['label']...
  USS        = varname ['label']...
  ISDNUMBER = n
```

Data Print Formatting

```
  PRINTFORMATS= varlist (option)...
```

Page Formatting

```
  PAGETITLE   = 'string'
  PAGELENGTH  = n
```

```
PAGEWIDTH   = n
COLLAPSE
```

Header Formatting

```
HEADERWIDTH       = n
HEADERINDENTATION = n
HEADERDIVIDER     = 'character'
NODIVIDERS
NOHEADERCENTER
```

Stub Formatting

```
STUBTITLE       = option 'string'
STUBWIDTH       = n
STUBINDENTATION = n
STUBCONTINUATION= n
STUBHEIGHT      = n
STUBFILLER      = 'character'
STUBDIVIDER     = 'character'
```

Wafer Formatting

```
WAFERTITLE        = option 'string'
NOWAFERTITLE
WAFERCENTER
WAFERSTART        = n
WAFERINDENTATION  = n
WAFERCONTINUATION = n
WAFERHEIGHT       = n
```

Other Options

```
HTML
NOBOTTOMBORDER
NOLEFTBORDER
NORIGHTBORDER
NOTES= 'string'
FOOTNOTES= 'string'
WAFERNOTES= 'string'
NOZEROS
ZEROCHARACTER= 'character'
NOEMPTYROWS
DELETEMISSING
UPPERCASE
SPANNERS
DUMMY  = varname 'label' ...
LANGUAGE = DUTCH | GERMAN
```

The `FILENAME` and the header expression are the only required clauses.

`FILENAME`	Specify the filename created by the procedure. If a filename is not specified on a subsequent `TABULATE` procedure, the output is appended to the last `TABULATE` output file. If the filename is `CGI` then, if the procedure is run when SIR/XS is operating from the common gateway interface, the output is returned to the user's browser. When run locally, a filename of `CGI` results in the file `sircgi.htm`. It is normal to specify the `HTML` keyword when using `CGI` as the filename.

expressions

The three possible expressions (`HEADER`, `STUB` & `WAFER`) define the data in one of the dimensions of the table. Each expression consists of from one to n variables and/or statistical keywords all enclosed in brackets. If multiple variables are specified on a single expression they are related by the keywords `BY` or `THEN`.

Percentages can also be used in expressions.

The type of variable, Variable Modifiers, and statistics all effect the Cell contents.

`HEADER = (expression)`	Defines the *columns* of the tabulation. This clause is required.
`STUB = (expression)`	Defines the *rows* of the tabulation.
`WAFER = (expression)`	Defines the *sections* of the tabulation. Sections are a number of two dimensional sections specified by the stub and header.
BY	Nest a variable within another thus producing combinations of the variable values within one dimension of the table. The asterisk (*) symbol may be used instead of the keyword.
THEN	Appends a second variable or statistical keyword to the set of variables displayed within one dimension of the table. The plus (+) symbol may be used instead of the keyword.
BY and THEN	Can be used in combination to create complex structures within a single dimension or expression.

Errors may occur, not only if the table is misspecified, but also if it cannot fit the output page.

Expressions

The `TABULATE` command has the general form:

```
TABULATE  HEADER = (expression) [STUB = (expression) [WAFER =
(expression)]]
```

A table consists of up to three components or dimensions: *header*, *stub* and *wafer*. Each dimension is defined by an expression. Each expression is identical in format but applies to a different dimension of the table. These expressions name the variables to tabulate and their relation to each other in the table. The different keywords (`HEADER`, `STUB` & `WAFER`) define the dimension being specified. The `HEADER` is required; `WAFER` can only be specified if a `STUB` is also specified. The sequence of the three expressions in the command is irrelevant.

The header defines variables whose values correspond to columns; the stub defines variables whose values correspond to rows. The rows and columns define a two-dimensional array called the wafer. The wafer is a higher level categorisation resulting in multiple individual row/column combinations. The table description consists of one to three expressions.

Define an expression with variable names and keywords. At the most basic an expression consists of a single variable name. Multiple variables or statistics may be defined in a single expression separated with the keywords `BY` and `THEN`. `BY` specifies a level of nesting within a single dimension; `THEN` specifies multiple variables within a single dimension.

The structure of the table is defined by the values of the control variables specified in the various expressions. The values in the variables named in these expressions are used as the labels for the particular dimension being specified. The number of categories of any control variable is taken from the schema definition which contains information on variable ranges, valid values and value labels.

The body of the table consists of *Cells.* The content of a cell is a summary of data which applies to that particular combination of variables in the expressions. This might be a count of occurrences of data, a sum of values, a percentage or some other statistic.

If the command has a single header expression, a table containing a single value for each column is produced. That is one line of data is produced with one entry per column. For example:

```
TABULATE  HEADER = (REGION)
TABULATE  HEADER = (AGE BY SEX)
TABULATE  HEADER = (SALARY BY (MIN THEN MEAN THEN MAX))
```

If the tabulate statement contains a `STUB` expression, then a two-dimensional table is produced. The stub defines the rows of the table going down the page. The header defines the columns going across the page. For example,

```
TABULATE HEADER = (RACE)  STUB = (REGION)
TABULATE HEADER = (RACE THEN SEX) STUB = (REGION)
TABULATE HEADER = (SALARY BY (MEAN THEN MEDIAN)) STUB = (REGION BY
RACE)
```

By specifying variables in two expressions, a cross-tabulation is produced with one cell for each value of each variable. For example, if a cross tabulation for two variables is specified, where each variable has two values, four cells are produced. The more values, the more cells.

Nesting variables with the `BY` operator also produces a cross-tabulation within a single dimension. If the same variables are used as a Stub and header or as a header with a `BY` clause, the same cells are produced but the table has a different shape.

If the tabulate statement contains a `WAFER` expression, a set of two-dimensional tables is produced. There must be a stub expression for a `WAFER` expression to be valid. When more than one wafer is generated a table of contents is also produced. If a particular wafer contains no data, it is not printed and is marked deleted in the table of contents. If only one wafer is produced, it is printed even if it is empty. For example:

```
TABULATE HEADER = (SEX)           STUB = (AGE)         WAFER = (REGION)
TABULATE HEADER = (SEX)           STUB = (AGE)         WAFER = (REGION
BY RACE)
TABULATE HEADER = (AGE BY REGION) STUB = (RACE BY SEX) WAFER = (COUNT
THEN PCT)
```

BY

Specifying BY between two variables nests each of the values of the second variable in each of the values of the first variable. If a second BY is specified, a third level of nesting takes place. BY defines a structure within a particular dimension. Each variable specifies a further level of nesting. The number of cells produced is the product of the number of values for each variable. There is no limit to the number of levels allowed within one expression. The asterisk (*) symbol can be used instead of BY.

Example: BY

HEADER = (AGE BY INCOME)

Young		Old	
Rich	Poor	Rich	Poor
6	4	8	3

HEADER = (SEX BY AGE BY INCOME)

Male				Female			
Young		Old		Young		Old	
Rich	Poor	Rich	Poor	Rich	Poor	Rich	Poor
4	1	6	2	2	3	2	1

THEN

Specifying `THEN` between two variables or two clauses in effect concatenates two independent tables with all values of the first table followed by all values of the table. The plus (+) symbol can be used instead of `THEN` .

Example: THEN

The following header expression results in the independent tabulation of individual occurrences of the two values for each of the two variables:

```
HEADER = (AGE THEN SEX)
```

Young	Old	Male	Female
10	11	13	8

When two variables are joined by `THEN`, each value of each variable defines a cell. The number of cells produced is the sum of the number of values for each variable.

Combining BY and THEN

`BY` takes precedence over `THEN` when the two operators are used in combination in an expression. Use parentheses to specify a different precedence.

Example: BY and THEN

The following examples show various combinations of the same variables in a header expression. In the first example, since nesting takes precedence, `SEX` is nested with `INCOME`, and the result of that nesting operation is concatenated with `AGE`.

```
HEADER = (AGE THEN INCOME BY SEX)
```

Young	Old	Rich		Poor	
		Male	Female	Male	Female
10	11	10	4	3	4

The next example concatenates `AGE` and `INCOME` and nests `SEX` within both `AGE` and `INCOME`, using parentheses to cause the concatenation operator to be applied first.

```
HEADER = ((AGE THEN INCOME) BY SEX)
```

Young		Old		Young		Old	
Male	Female	Male	Female	Male	Female	Male	Female
5	5	8	3	10	4	3	4

Other combinations might be:

```
HEADER = (AGE BY INCOME THEN SEX)
```

Young		Old		Male	Female
Rich	Poor	Rich	Poor		
6	4	8	3	13	8

```
HEADER = (AGE BY (INCOME THEN SEX))
```

Young				Old			
Rich	Poor	Male	Female	Rich	Poor	Male	Female
6	4	5	5	8	3	8	3

Nesting and concatenation can also be used in the stub and wafer expressions.

Variable Types

Tabulate uses numeric variables only. This includes categorical variables, which are numeric codes representing discrete string values. In a VisualPQL program, string variables can be recoded into numeric categorical values. There are two types of numeric variables; variables with discrete categories, known as `CONTROL` variables and variables with continuous values, known as `OBSERVATION` variables.

Tabulate determines the tabulation type of a variable from the information in the variable schema according to the following rules:

- Variables that are `CATEGORICAL,` or have `VALID VALUES` or have `VALUE LABELS` are control variables. All other variables are observation variables.
- Database and table variables may be explicitly declared as `CONTROL VARS` or `OBSERVATION VARS` in the data dictionary.
- Variables implicitly defined with `GET VARS` maintain their tabulation type.
- `The CONTROL VARS` and `OBSERVATION VARS` commands in a VisualPQL program assign an explicit tabulation type to local variables created in the program.
- The tabulation type of variables can be modified in the tabulate statement.
- Variables declared as `CONTROL` variables **must** have ranges specified.

Most variables are either control or observation. For example, salary would be an observation variable and department would be a control variable. However, some variables may be reasonably used in either way. A variable containing number of persons per family might be used as a control variable in one table but might be aggregated in another table along with a count of families, in order to get the average persons per family. Such a variable can be defined as observation type in the VisualPQL program or database and then reclassified in the `TABULATE` statement as a control variable.

If a control variable contains missing values, the record is normally excluded from the body of the tabulation. Categories can be generated to show cells for missing control variables. If an observation variable contains missing values, the record is normally included but the variable is not computed into the cell contents. This can give rise to situations where overall totals do not agree with counts in the body of the table. Specify the `DELETEMISSING` keyword to exclude records where any cross-tabulated variable has a missing or undefined value.

Cell Contents

Different types of variables combine to give different cell contents and a different table.

Control variables provide classifications which determine the structure of the table and the presence of a particular cell. Observation variables give the result that appears in the cell.

When two control variables are cross tabulated or nested, the content of the cell is a count of the number of occurrences of data which match that combination of values.

When an observation variable is cross tabulated or nested with a control variable, the **total** (sum) for the observation variable is the default. That is, the structure of the table is unaltered by the inclusion of an observation variable but, instead of a count of data appearing in the cell contents, a total of the observation variable is calculated and an appropriate header label printed. Other statistics can be specified for observation variables.

Two observation variables cannot be cross tabulated or nested within each other and therefore observation variables may be specified in one and only one expression (dimension) of the `TABULATE` statement.

Different types of information are reported in the cell contents for Control and Observation variables. For Control variables, which have discrete values, counts or percentages are reported. For Observation variables, which have continuous values, a variety of cell statistics are reported, the default being a sum of the values. For any cell that cross tabulates a control and an observation variable, the observation variable statistics are reported.

Example: Observation Variables

In the following three examples, the observation variable `SALARY` is included in the expression. The number in any given cell is the sum of the `SALARY` values from each record having the combination of values for the control variables for that particular cell. The placement of the observation variable in the header expression determines the placement of the label, but does not alter the content of the table. The observation variable does not alter the number of cells. Observation variables and counts may be specified in a single expression in various ways:

```
HEADER = (SALARY BY AGE BY SEX) STUB = (REGION)
```

	SALARY	
	Young	Old

	Male	Female	Male	Female
North..........	2800	5650	20100	3200
South..........	2200	11700	4950	5000

`HEADER = (SEX THEN (SALARY BY SEX) STUB = (AGE)`

	Male	Female	Current Monthly Salary	
			Male	Female
Young...	5	5	13050	14450
Old.....	8	3	21900	8200

`HEADER = (AGE BY (SEX THEN SALARY)) STUB = (REGION)`

	Young			Old		
	Male	Female	Current Monthly Salary	Male	Female	Current Monthly Salary
North....	3	2	13600	6	1	20150
South....	2	3	13900	2	2	9950

Variable Modifiers

Modifiers change the attributes of a variable in an expression. These have the form:

```
variable[.modifier][.modifier]...
```

Several modifiers can be appended to a single variable. The modifiers and their abbreviations are:

```
.CONTROL   (.C)
.OBS       (.O)
.FIRST     (.F)
.TOTAL     (.T)
.MISSING   (.M)
.UNDEFINED (.U)
.SPAN      (.S)
.NOSPAN    (.NOS)
```

The `.CONTROL` and `.OBS` modifiers alter the default tabulation type of a variable (i.e. control or observation). The `.CONTROL` modifier may only be specified for an observation variable if valid ranges have been specified for it.

The `.FIRST` modifier is used in conjunction with the percent base marker as described in the section on percentages. It specifies that the first occurrence of the percent base marker is used.

The `.TOTAL` modifier automatically concatenates a `TOTAL` control variable as described in the section on `TOTAL`. The `TOTAL` is displayed **before** the variable being modified.

When a control variable has a missing or undefined value, it is normally excluded from the detail cells in the table. Specify the `.MISSING` or `.UNDEFINED` modifiers to generate additional cells in the table. The `.UNDEFINED` modifier groups all missing and undefined values into a single cell. The `.MISSING` modifier displays undefined and each defined missing value as separate cells. These modifiers override any `DELETEMISSING` specification for that variable.

Example: Undefined Modifier

```
HEADER = (AGE.T.U)
```

TOTAL	Young	Old	AGE= UNDEFINED
22	10	11	1

By default, the rows and columns of a table are labeled with value labels (if they exist) or with the "variable=value" notation. The `.SPAN` modifier generates a second level of heading for a variable. This heading is either the variable label (if it exists) or the variable name. This new heading "spans" all the categories of the variable, hence the name `.SPAN`.

Example: SPAN Modifier

The next two tables illustrate the effect of `.SPAN`. The only difference between them is the `.SPAN` modifier in the second table.

```
HEADER = (AGE BY SEX) STUB = (REGION)
```

	Young		Old	
	Male	Female	Male	Female
North..........	3	2	6	1
South..........	2	3	2	2

```
HEADER = (AGE.S BY SEX.S) STUB = (REGION.S)
```

	AGE			
	Young		Old	
	Gender			
	Male	Female	Male	Female
REGION				
North..........	3	2	6	1
South..........	2	3	2	2

The `SPANNERS` option can be used to turn on spanning labels for all variables. The `.NOSPAN` modifier turns off spanning labels for a particular variable when the `SPANNERS` option is in effect.

Statistics

Other than TOTAL, statistics are used with observation variables. Statistics are specified with the `BY` keyword to associate them to an observation variable.

There are two methods of specifying statistics.

> The first method is to specify a pseudo-variable in the expression and to use one of the statistical option clauses to specify the statistic and label to produce in the form:
>
> `keyword = pseudo-variable 'label'`
>
> The 'label' is optional but the variable name is required.
>
> The second method is to use a statistical keyword directly. In effect, these are pre-defined, convenient pseudo-variables with standard labels. (This method cannot be used with `NORMALIZED, PERCENT, QUANTILE` and `RANGE`, since additional data is required on the specification.)

The statistical clauses are:

`COUNT`	A count of records in this category.
`CSS`	Corrected Sum of Squares, where the cell = sum of squares - ((sum**2)/n).
`CV`	Coefficient of Variance, where the cell = (standard deviation/mean) * 100.
`CVERR`	Coefficient of Error, where the cell = (standard error/mean) * 100.
`MAXIMUM`	Maximum value of the variable.
`MEAN`	Mean value of the variable.
`MEDIAN`	Median value of the variable.
`MINIMUM`	Minimum value of the variable.
`MISSING`	Count of missing values in the variable.
`NORMALIZED*`	Normalized quantiles (by median value) where cell = (quantile/median) * 100.
`PERCENT*`	Percentage, where cell = (cell/base) * 100.

`QUANTILE*`	Quantile, equal or unequal quantiles can be specified.
`RANGE*`	Range produces the range of the variable expressed as the difference between the lowest and the highest values encountered. The format of the range clause is: `RANGE = variable (lo,hi) ['label']` The specification of the lo and hi values defines the outer limits of any values to be included in the calculation. This can be used, for example, to exclude negative or zero values or very high numbers used as some coding convention.
`STDERR`	Standard Error where cell = standard deviation/sqrt(n).
`STDEV`	Standard Deviation about the mean.
`TOTAL`	Count of records. When cross tabulated with an observation variable, gives sum of values of the observation variable.
`TSTATISTIC`	T-Statistic where cell = mean/standard error.
`USS`	Uncorrected Sum of Squares where cell = sum of squares.

***** Cannot be used as keywords directly in an expression.

Example Statistical Clauses

The `TOTAL` clause specifies that the pseudo-variable N is a total, with the label 'Valid Cases'. The `MEAN` and `STDEV` clauses assign labels for these pseudo-variables to 'X Bar' and 'Sigma'.

```
TABULATE HEADER = (N THEN SALARY BY (MEAN THEN STDEV)) STUB = (SEX)
         TOTAL  = N 'Valid Cases'
         MEAN   = MEAN 'X Bar'
         STDEV  = STDEV 'Sigma'
```

	Valid Cases	Current monthly salary	
		X Bar	Sigma
Male...........	13	2688	486
Female.........	8	2831	746

The pseudo-variables created can be used anywhere that is valid. If the pseudo-variable is a `TOTAL` it is a control variable, otherwise it is an observation variable. Multiple pseudo-variables can be created and assigned to the same keyword. In the next example, both `N` and `SUM` are totals, `N` is the count of records and `SUM` is the total `SALARY`.

```
TABULATE HEADER = (N THEN SALARY BY (SUM THEN MEAN THEN STDEV)) STUB =
(SEX)
         TOTAL = N 'Valid Cases'  SUM 'Sum'
         MEAN  = MEAN 'X Bar'
```

```
          STDEV = STDEV 'Sigma'
```

	Valid Cases	Current monthly salary		
		Sum	X Bar	Sigma
Male........... Female.........	13 8	34950 22650	2688 2831	486 746

Example: Statistical keywords

```
HEADER = (SALARY BY (MIN THEN MAX)) STUB = (SEX)
```

	Current monthly salary	
	MINIMUM	MAXIMUM
Male........... Female.........	2000 1650	3600 4000

Keywords must be nested within an observation variable. Keywords may not be nested within each other; they may be concatenated. For example, `SALARY BY MEAN BY STDEV` is not allowed; `SALARY BY (MEAN THEN STDEV)` is allowed.

Accuracy and ISDNUMBER

Medians and quantiles are estimates of the true values. The accuracy of the estimates is governed by a value called the interval size designator or `ISD` number . This number is initially set to 4 which means that the maximum relative error in a median or quantile estimate should not exceed 12.5%. The value is usually more accurate if the values are distributed smoothly. Set the ISD number explicitly by using the clause:

```
ISDNUMBER = n /
```

that sets the `ISD` number to **n**. Increasing the ISD number by 1 cuts the maximum relative error in half while doubling the internal storage required to compute the medians or quantiles. The `ISDNUMBER` clause must precede the clauses it is to affect. For example,

```
TABULATE HEADER    = (SALARY BY (MEAN THEN Q)) STUB = (TOTAL THEN
REGION)
         ISDNUMBER = 5
         QUANTILES = Q 'Quantiles' 4
```

Total

`TOTAL` can be specified as an independent control variable which is a count of records. It is specified in exactly the same way as other statistics (using either pseudo-variables or a keyword). However, since it is a control variable rather than an observation variable, it operates in a different manner.

Example: TOTAL

```
HEADER = (AGE THEN TOTAL)
```

Young	Old	TOTAL
10	11	22

The control variable, `AGE`, with two classifications is concatenated with `TOTAL`. The total column equals the count of **all** occurrences at the appropriate level of nesting. (Note that records with undefined values have still been counted.)

```
HEADER = (SALARY THEN TOTAL)
```

Current monthly salary	TOTAL
57600	22

`SALARY` is an observation variable and thus the sum is displayed by default. The keyword `TOTAL` is a control variable and displays a count. Cross tabulating or nesting `TOTAL` with either control or observation variables is allowed since it is a control variable. As per the standard rules for cell contents, if `TOTAL` is nested with a control variable, a count is produced; if it is nested with an observation variable, a sum of the observation variable is produced.

Example: TOTAL with BY and THEN

The following header expression uses the observation variable `SALARY` and the control variables `AGE` and `SEX`:

```
HEADER = (SALARY BY AGE BY (TOTAL THEN SEX))
```

Current Monthly Salary	
Young	Old

TOTAL	Male	Female	TOTAL	Male	Female
27500	13050	14450	30100	21900	8200

Since `TOTAL` is nested within an observation variable, this column contains the total `SALARY` for each `AGE` group.

TOTAL shorthand notation

There is a shorthand notation for `TOTAL`. Appending a single T to the variable name separated by a period specifies that a `TOTAL` is produced **before** the variable. The following two expressions are identical:

```
TOTAL THEN AGE
AGE.T
```

A tabulate statement can be shortened as follows:

```
TABULATE  HEADER = (SALARY BY AGE BY (TOTAL THEN SEX))
TABULATE  HEADER = (SALARY BY AGE BY SEX.T)
```

Normalized and Quantiles

Normalized and Quantile specifications are very similar to other statistics but have to supply some additional information. This can take one of two forms:

In the first form, the quantity "n" specifies the number of equal-sized quantiles to be produced. Of these, n-1 are printed. Thus for n=4, `TABULATE` prints output for the 25%, 50% and 75% quantiles. In the second form a number of non-equal sized quantiles are produced and each n specifies the quantiles to produce. For example, specifying Ns of 15 and 85 produces the 15th and 85th percentage quantiles. Both forms allow specification of labels. In the first form, when labels are used for individual quantiles, the commas separating 'label1', 'label2', etc. are required.

```
1)  keyword = variable ['label'] n ['label1', ...]
2)  keyword = variable ['label'] (n1) ['label'] (n2)...
```

Example: Quantiles

```
TABULATE HEADER = (SALARY BY (MEAN THEN Q)) STUB = (REGION.T)
         QUANTILES = Q 'Quantiles' 4
```

	Current monthly salary			
		Quantiles		
	MEAN	QUANTILE-=25	QUANTILE-=50	QUANTILE-=75
TOTAL.......	2743	2311	2678	3222
North........	2813	2238	2825	3315
South........	2650	2369	2538	2913

A specification in the first form , with multiple labels:

```
TABULATE  HEADER    = (SALARY BY (MEAN THEN Q)) STUB = (REGION.T)
          QUANTILES = Q 'Quantiles' 4 '25%', 'Median', '75%'
```

	Current monthly salary			
		Quantiles		
	MEAN	25%	Median	75%
TOTAL........	2743	2311	2678	3222
North........	2813	2238	2825	3315
South........	2650	2369	2538	2913

A specification in the second form:

```
TABULATE HEADER    = (SALARY BY (MEAN THEN Q)) STUB = (REGION.T)
         QUANTILES = D 'Deciles' (10) '10th' (50) '50th' (90) '90th'
```

	Current monthly salary			
	MEAN	Deciles		
		10th	50th	90th
TOTAL........	2743	2051	2678	3407
North........	2813	1973	2825	3647
South........	2650	2267	2538	3265

Percentages

`TABULATE` can calculate and display percents. Tables can be specified which display just percentages or which also include the original values.

Tables can be specified which contain row percentages, column percentages or overall percentages. Multiple percentages can be displayed in a single table. A single table can have percentages computed from more than one base value. However a table cannot have any cell which is an intersect of two percentage calculations. This means that all percentage cells must appear in a single expression; there cannot be percentages in both a row and column expression.

To display percentages, specify where the percents appear and the base of each percent calculation. Use the `PERCENT` option to define a pseudo-variable and then reference this in an expression to display the percent. Specify the base variable to use to calculate the percentage by appending a percentage symbol (%) and the same pseudo-variable name to a base variable. The only valid base variables are `TOTAL` control variables, typically cross tabulated against the observation variable that is the percentage variable.

Shorthand Notation for Base Marker Specification

If a percent base marker is appended to a variable, it specifies that a `TOTAL` precedes the variable and that this is the base. For example, the following expressions are equivalent:

```
TOTAL%X THEN REGION
REGION%X
```

This shorthand notation can simplify the tabulate expressions. For example, both the following statements are equivalent:

```
TABULATE HEADER = (TOTAL%X THEN REGION) STUB = (X BY (TOTAL%X THEN
AGE))
TABULATE HEADER = (REGION%X)            STUB = (X BY AGE%X)
```

The table output is the same in both cases. With the shorthand form, the label of the created `TOTAL` category is always "`TOTAL`".

A brief review of .T and %X notation: An expression such as `AGE.T`, produces three rows or columns - `TOTAL,Young` and `Old`. An expression such as `REGION%X`, produces three categories - `TOTAL`, `North` and `South`. It also specifies the `TOTAL` column as the percent base. The sub-expression, `X.T`, is the same as `TOTAL THEN X` and produces two cells for each category.

The basics of producing percentages in tables are covered in simple percentages. You can also get percentages with observation variables and tables with values and percentages. Tables can have percentages for row totals and can be displayed in different wafers. There are numerous examples of different tables using percentages in percentage examples.

Percentages with Counts

The simplest percentage tables are produced with control variables and cell contents are therefore counts. Consider the following standard table:

```
HEADER = (TOTAL THEN REGION) STUB = (TOTAL THEN AGE)
```

	TOTAL	North	South
TOTAL.......... Young.......... Old............	22 10 11	12 5 7	9 5 4

To add percentages to this table, define a pseudo-variable, say X, and reference this on `PERCENT` clause, for example:

```
PERCENT = X 'Percent'
```

Choose the base row or column for the percentage computations. Base cells are indicated within an expression by appending a percentage symbol (%) and the pseudo-variable to the base variable. For **row percentages**, there must be a **total column** for the base or 100% column. In the example, the `TOTAL` variable in the header expression is the total column. Therefore, append `%X` to `TOTAL` to make it the base for percentage calculations.

Decide where to place the percent variable X, ensuring that it is nested correctly. Specifying it in the header expression or in the stub expression only affects the labeling of the table.

Row Percents

```
TABULATE HEADER  = (X BY (TOTAL%X THEN REGION)) STUB = (TOTAL THEN AGE)
         PERCENT =  X 'Percent'
```

	Percent		
	TOTAL	North	South
TOTAL.......... Young.......... Old............	100 100 100	55 50 64	41 50 36

The total (100%) and regional percentages for each age group (row) are displayed in the table. The table contains only percentages because the percent variable, X, is at the highest level of the header expression nesting. The remainder of the header, `(TOTAL THEN REGION)`, is nested within X. The percent marker, `%X,` specifies the header column `TOTAL` as the 'set' of base cells for this table's percent calculations. Each cell value for `TOTAL` is the base for the percents in its row. For example, the calculations for the first row of the table are:

```
TOTAL  = 2200/2200 * 100 = 100
         North  = 1200/2200 * 100 = 55
         South  =  800/2200 * 100 =  41
```

Note: The above illustrates the effect of missing values in a control variable. Some records had missing values in `REGION` and thus the columns do not reflect these records. If required, specify the `MISSING` modifier on `REGION` to generate an extra column for those records.

Moving X to the stub expression changes the labeling of the table but not its contents:

```
TABULATE HEADER = (TOTAL%X THEN REGION) STUB = (X BY (TOTAL THEN AGE))
         PERCENT = X 'Percent'
```

	TOTAL	North	South
Percent			
TOTAL..........	100	55	41
Young..........	100	50	50
Old...........	100	64	36

Column Percents

To display column percentages instead of row percentages, use a **total row** to be the base or 100% row.

In the example, move the base marker, `%X`, from `TOTAL` in the header and append it to `TOTAL` in the stub:

```
TABULATE HEADER = (X BY (TOTAL THEN REGION)) STUB = (TOTAL%X THEN AGE)
         PERCENT = X 'Percent'
```

	Percent		
	TOTAL	North	South
TOTAL..........	100	100	100

Young.......... Old............	45 50	42 58	56 44

As in the preceding table, the percent variable, `X`, is nested with all cells of the table so that all cells contain percents. In this table, however, the percent marker, `%X`, specifies the `TOTAL` row in the stub expression as the set of base cells for this table's percent calculations. For example, the calculations for the first column of the table are:

```
          TOTAL             = 2200/2200 * 100 = 100
          Young             = 600/2200 * 100 =  45
          Old               = 1400/2200 * 100 =  50
```

Base Markers in Both the Header and the Stub

- If the percent base marker is appended to a total column in the header expression, row percentages are produced.
- If the percent base marker is appended to a total row in the stub expression, column percentages are produced.

To get overall or grand total percentages, place the percent base markers in both the stub and the header. By convention, the cell (or cells) at the intersection of the base row and column are considered the base cells for the table.

```
TABULATE  HEADER = (X BY (TOTAL%X THEN REGION)) STUB = (TOTAL%X THEN
AGE)
          PERCENT = X 'Percent'
```

	Percent		
	TOTAL	North	South
TOTAL.......... Young.......... Old............	100 45 50	55 23 32	41 23 18

In this table, all percentages are measured using the first cell of the table as the base. This cell is at the intersection of the stub `TOTAL` row and the header `TOTAL` column.

Percentages with Observation Variables

Percentages of observation variables can be produced but at least one control variable must be nested with the percent variable.

Example: Percentages for Observation Variables

The following shows salary by age group:

```
TABULATE  HEADER = (SALARY BY (TOTAL THEN AGE))
```

Monthly salary		
TOTAL	Young	Old
57600	27500	30100

The corresponding percentage table is:

```
TABULATE  HEADER = (X BY SALARY BY (TOTAL%X THEN AGE))
PERCENT = X 'Percent' /
```

Percent		
Monthly salary		
TOTAL	Young	Old
100	48	52

Moving the clause `(TOTAL%X THEN AGE)` to the front of the expression, results in the same data in the table with different labeling:

```
TABULATE HEADER = ((TOTAL%X THEN AGE) BY X BY SALARY)
         PERCENT = X 'Percent'
```

TOTAL	Young	Old
Percent	Percent	
Monthly salary	Monthly salary	Monthly salary
100	48	52

Original Values and Percents

Original values can be included as well as the percents. In the examples that follow, all of the manipulations are performed in the header expression. However, they could equally be done in stub or wafer expressions.

The following has alternating columns of original values and percents. Note only the cells nested within the percent variable X contain percents.

```
TABULATE HEADER  = (REGION%X BY X.T) STUB = (AGE.T)
         PERCENT =  X 'Pct' /
```

	TOTAL		North		South	
	TOTAL	Pct	TOTAL	Pct	TOTAL	Pct
TOTAL...... Young...... Old........	22 10 11	100 100 100	12 5 7	55 50 64	9 5 4	41 50 36

Instead of presenting the data in alternating columns, it could be grouped into separate blocks of frequencies and percents by inverting the order of the header expression.

```
TABULATE HEADER  = (X.T BY REGION%X) STUB = (AGE.T)
         PERCENT =  X 'Pct' /
```

	TOTAL			Pct		
	TOTAL	North	South	TOTAL	North	South
TOTAL..... Young..... Old.......	22 10 11	12 5 7	9 5 4	100 100 100	55 50 64	41 50 36

Percents for Row Totals.

```
TABULATE HEADER  = (TOTAL THEN (X BY REGION%X)) STUB = (AGE.T)
         PERCENT =  X 'Pct' /
```

	TOTAL	Pct		
		TOTAL	North	South
TOTAL.......... Young.......... Old............	22 10 11	100 100 100	55 50 64	41 50 36

An analogous set of tables can be designed for percentages and totals of an observation variable. The following table displays alternating columns of total salary and percentage.

```
TABULATE HEADER  = (SALARY BY SEX%X BY X.T STUB) = (AGE.T)
         PERCENT =  X 'Pct'
```

	Current monthly salary					
	TOTAL		Male		Female	
	TOTAL	Pct	TOTAL	Pct	TOTAL	Pct
TOTAL..... Young..... Old.......	57600 27500 30100	100 100 100	34950 13050 21900	61 47 73	22650 14450 8200	39 53 27

This table groups all salary columns followed by all the percent columns.

```
TABULATE HEADER  = (SALARY BY X.T BY SEX%X) STUB = (AGE.T)
         PERCENT =  X 'Pct'
```

	Current monthly salary					
	TOTAL			Pct		
	TOTAL	Male	Female	TOTAL	Male	Female
TOTAL..... Young..... Old.......	57600 27500 30100	34950 13050 21900	22650 14450 8200	100 100 100	61 47 73	39 53 27

The following shows total salary for both sexes and percentage by sex.

```
TABULATE HEADER  = (SALARY BY (TOTAL THEN (X BY SEX%X))) STUB = (AGE.T)
         PERCENT =  X 'Pct'
```

	Current monthly salary			
	TOTAL	Pct		
		TOTAL	Male	Female
TOTAL.....	57600	100	61	39
Young.....	27500	100	47	53
Old.......	30100	100	73	27

Percentages in Wafers

It is possible to display frequencies and percentages in different wafers of the same table. For example:

```
TABULATE HEADER  = (REGION%X) STUB = (AGE%X) WAFER = (X.T)
         PERCENT =  X 'Percent'
```

The wafer expression is `X.T` which stands for `TOTAL THEN X`. Thus the table consists of two wafers, the first wafer contains totals and the second contains percentages. Each wafer is a two-dimensional table of `AGE` (in the stub) by `REGION` (in the header). Since the base marker, `%X`, appears in both the stub and the header, the percentage wafer contains overall percentages. The following two wafers are the output of the previous tabulate statement:

```
HEADER  = (REGION%X) STUB = (AGE%X) WAFER = (X.T)
Table of Contents:
TOTAL........................................  page 1
Percent......................................  page 2

HEADER  = (REGION%X) STUB = (AGE%X) WAFER = (X.T)
TOTAL..............
```

	TOTAL	North	South
TOTAL..........	22	12	9
Young..........	10	5	5
Old...........	11	7	4

(On next page)

```
HEADER = (REGION%X) STUB = (AGE%X) WAFER = (X.T)
Percent............
```

	TOTAL	North	South
TOTAL..........	100	55	41
Young..........	45	23	23
Old...........	50	32	18

Example percentage tables

The following examples illustrate some of the different ways to display percentages. The examples are all based on the following table of frequency counts.

```
TABULATE HEADER = (REGION.T) STUB = (AGE.T BY SEX.T)
```

	TOTAL	North	South
TOTAL			
TOTAL...........	22	12	9
Male............	13	9	4
Female..........	8	3	5
Young			
TOTAL...........	10	5	5
Male............	5	3	2
Female..........	5	2	3
Old			
TOTAL...........	11	7	4
Male............	8	6	2
Female..........	3	1	2

Base marker appended to AGE

This table contains nine base (100%) cells. The `TOTAL` for `AGE` is the base cell (`AGE%X`). Since `SEX.T` is nested within `AGE`, there are three base cell rows, and since the header contains three columns (`REGION.T`) there is a total of nine base cells (three columns times three rows).

```
TABULATE HEADER = (X BY REGION.T)  STUB = (AGE%X BY SEX.T)  PERCENT = X
'Percent'
```

	Percent		
	TOTAL	North	South
TOTAL			
TOTAL..........	100	100	100
Male...........	100	100	100
Female.........	100	100	100
Young	45	42	56
TOTAL..........	38	33	50
Male...........	63	67	60
Female.........	50	58	44
Old	62	67	50
TOTAL..........	38	33	40
Male...........			

Female.........			

Base marker appended to SEX

Note that the nine base cells now occur where `TOTAL` for `SEX` appears.

```
TABULATE HEADER = (X BY REGION.T) STUB = (AGE BY SEX%X) PERCENT = X
'Percent'
```

	Percent		
	TOTAL	North	South
Young			
TOTAL..........	100	100	100
Male...........	50	60	40
Female.........	50	40	60
Old			
TOTAL..........	100	100	100
Male...........	73	86	50
Female.........	27	14	50

Use of FIRST

In the previous table, base rows appeared at **every** occurrence of the `TOTAL` variable for `SEX`. The next table uses `.FIRST` to restrict the base row to the **first** occurrence of `TOTAL` for `SEX`.

```
TABULATE HEADER  = (X BY REGION.T) STUB = (AGE BY SEX%X.F)
         PERCENT =  X 'Percent'
```

	Percent		
	TOTAL	North	South
Young			
TOTAL..........	100	100	100
Male...........	50	60	40
Female.........	50	40	60
Old			
TOTAL....... ...	110	140	80
Male...........	80	120	40
Female.........	30	20	40

Multiple Base Markers

The next three tables are similar to the first three with the addition of a second base marker appended to `REGION` in the header. The base cells occur at the intersections of the base rows and columns.

```
TABULATE HEADER  = (X BY REGION%X) STUB = (AGE%X BY SEX.T)
         PERCENT =  X 'Percent'
```

	Percent		
	TOTAL	North	South
TOTAL			
TOTAL..........	100	55	41
Male...........	100	69	31
Female.........	100	38	63
Young			
TOTAL.....	45	23	23
Male...........	38	23	15
Female.........	63	25	38
Old			
TOTAL..........	50	32	18
Male...........	62	46	15
Female.........	38	13	25

```
TABULATE HEADER  = (X BY REGION%X) STUB = (AGE BY SEX%X)
         PERCENT = X 'Percent'
```

	Percent		
	TOTAL	North	South
Young			
TOTAL..........	100	50	50
Male...........	50	30	20
Female.........	50	20	30
Old			
TOTAL....... ...	100	64	36
Male...........	73	55	18
Female.........	27	9	18

```
TABULATE HEADER  = (X BY REGION%X) STUB = (AGE BY SEX%X.F)
         PERCENT = X 'Percent'
```

	Percent		
	TOTAL	North	South
Young			
TOTAL..........	100	50	50
Male...........	50	30	20
Female.........	50	20	30
Old			

TOTAL....... ...	110	70	40
Male...........	80	60	20
Female.........	30	10	20

Multiple Percent Variables Within a Table

More than one percent distribution can be displayed in a single table by specifying more than one percent variable. For example, the following two variables could be specified representing row and column percentages:

```
PERCENT=  PR 'Row Pct' PC 'Col Pct' /
```

Since `PR` represents row percentages, the base marker associated with `PR` must be in the header expression. Similarly, since `PC`, represents column percentages, the base marker associated with `PC` must be in the stub expression. In the example below, the results are presented in two tables concatenated together. The first table contains row percentages, the second contains column percentages.

```
TABULATE HEADER  = ((PR + PC) BY REGION%PR) STUB = (AGE%PC)
         PERCENT = PR 'Row Pct' PC 'Col Pct'
```

	Row Pct			Col Pct		
	TOTAL	North	South	TOTAL	North	South
TOTAL.....	100	55	41	100	100	100
Young.....	100	50	50	45	42	56
Old.......	100	64	36	50	58	44

Multiple percents in Rows

The next example presents the same data in a more conventional form. Notice (`PR + PC`) is moved to the stub expression.

```
TABULATE HEADER  = (REGION%PR) STUB = (AGE%PC BY (PR + PC))
         PERCENT = PR 'Row Pct' /
         PERCENT = PC 'Col Pct' /
```

	TOTAL	North	South
TOTAL			
Row Pct.........	100	55	41
Col Pct.........	100	100	100
Young			

Row Pct.........	100	50	50
Col Pct.........	45	42	56
Old			
Row Pct.........	100	64	36
Col Pct.........	50	58	44

Example: Overall Percents

Overall percentages can be included in the same table by adding a third percent variable, `PT`. Since `PT` is to represent overall percentages, append the `PT` base marker to both the stub and the header variables.

```
TABULATE HEADER =  (REGION%PR%PT) STUB = (AGE%PC%PT BY (PR + PC + PT))
         PERCENT = PR 'Row Pct'
         PERCENT = PC 'Col Pct'
         PERCENT = PT 'Tot Pct'
```

	TOTAL	North	South
TOTAL			
Row Pct.........	100	55	41
Col Pct.........	100	100	100
Tot Pct.........	100	55	41
Young			
Row Pct.........	100	50	50
Col Pct.........	45	42	56
Tot Pct.........	45	23	23
Old			
Row Pct.........	100	64	36
Col Pct.........	50	58	44
Tot Pct.........	50	32	18

Counts and Multiple Percents

Counts can be included by adding the `TOTAL` keyword:

```
TABULATE HEADER = (REGION%PR%PT) STUB = (AGE%PC%PT BY (TOTAL + PR + PC
+ PT))
         TOTAL = TOTAL 'Count'
         PERCENT = PR 'Row Pct'
         PERCENT = PC'Col Pct'
         PERCENT = PT 'Tot Pct'
```

	TOTAL	North	South
TOTAL			
Count...........	22	12	9
Row Pct.........	100	55	41

Col Pct.........	100	100	100
Tot Pct.........	100	55	41
Young			
Count...........	10	5	5
Row Pct.........	100	50	50
Col Pct.........	45	42	56
Tot Pct.........	45	23	23
Old			
Count...........	11	7	4
Row Pct.........	100	64	36
Col Pct.........	50	58	44
Tot Pct.........	50	32	18

Percentages in both Columns and Rows

It is not possible to produce a table which has a column of percentages on the total and a row of percentages on the same total. The solution to this is to produce a second table on the same file with a single row - the final percentages.

Suppose the table is a breakdown of males and females under and over forty. The basic table (without percentages) is easily produced with a table statement such as:

```
TABULATE HEADER = (AGEGROUP.T)  STUB = (GENDER.T)
```

However it is impossible to produce both sets of percentages in the same table. The required output is the following:

	Total Young and Old	Under 40	40+	% BY AGE
Total Males and Females	20	6	14	100%
Male	12	2	10	60%
Female	8	4	4	40%
% BY GENDER	100%	30%	70%	

It is simple to add percentages to either the row totals, giving percentages of males and females or the column totals giving percentages of age groups but not both at once. The solution is to produce two tables in the one run:

```
TABULATE HEADER =(TOTY THEN AGEGROUP THEN X) STUB = (TOTX%X THEN
GENDER)
          FILENAME = A
          TOTAL = TOTX 'Total Males and Females'
```

```
         PERCENT = X '% BY AGE'
TABULATE HEADER = (TOTY%Y THEN AGEGROUP) STUB = (Y)
         TOTAL = TOTY 'Total Young and Old'
         PERCENT = Y '% BY GENDER'
```

Note that the second tabulate does not have a `FILE` clause and thus writes the output to the same file. This second tabulate produces one row which is the required set of final percentages.

Record Filtering

```
BOOLEAN  = (logical expression)
SAMPLE   = fraction
WEIGHT   = varname
```

`BOOLEAN`	Specifies which procedure table records are used by the procedure. The procedure table records for which the logical expression is true are used by the procedure. If this option is not specified, all procedure table records are used.
`SAMPLE`	Specifies that a random sample of the procedure table records are used by the procedure. The **fraction** specifies the percent of records used and is specified as a positive decimal number less than or equal to 1 (one). .25, for example specifies that a 25% sample be used.
`WEIGHT`	Specifies an integer procedure variable used to weight the frequency counts and aggregations. By default, each record adds a count of 1 to frequency counts. Specifying a weight variable, adds the value of that variable rather than 1. For example, if `WEIGHT = FAMSIZE` were specified, then, in a table of `RACE` by `REGION`, a procedure table record would contribute a count of 5 to the `RACE=1`, `REGION=2` cell of the table if it contained the data: `RACE=1   REGION=2   FAMSIZE =5`

Print Formatting

```
PRINTFORMATS = var list (options,...) .../
```

By default, cell contents are printed as integers. The `PRINTFORMATS` option is used to alter the defaults. The options and their abbreviations are:

`COMMAS (C)`	Separates groups of 3 digits by commas.
`DECIMAL=n (n)`	Specifies the number of decimal places to be printed. A decimal point is only printed when the number of decimal places is non-zero.
`DOLLAR  (D)`	Places a floating dollar sign before the number.
`PERCENT  (P)`	Places a trailing percent sign after the number.

By default the `DOLLAR, PERCENT` and `COMMAS` options are not in effect and `DECIMAL=` 0 (i.e. numbers are printed as integers).

Use of the PRINTFORMATS clause.

```
TABULATE HEADER = (SALARY BY (TOTAL THEN (X BY SEX%X))) STUB = (AGE.T)
          PERCENT = X 'Pct' /
          PRINTFORMATS = SALARY (D,C) X (P,2) /
```

	Current monthly salary			
	TOTAL	Pct		
		TOTAL	Male	Female
TOTAL..........	$57,600	100.00%	60.68%	39.32%
Young..........	$27,500	100.00%	47.45%	52.55%
Old............	$30,100	100.00%	72.76%	27.24%

`TOTAL` under `SALARY` is printed with dollar signs and commas. However, `TOTAL` under `PCT` is printed with a percent sign and two decimal places because of the **precedence rules** for print formats.

Precedence rules for print formats

- Except for percents, variables that have no assigned `PRINTFORMAT` are not considered in the precedence analysis.
- `PRINTFORMATS` for percentages have the highest precedence. If a percent variable is not assigned an explicit `PRINTFORMAT`, it is printed as an integer.

- In an expression containing an observation variable nested with a control variable, the observation variable's `PRINTFORMAT` takes precedence.
- When a statistical keyword (e.g. `MEAN, MAX`, etc.) is nested with an observation variable, the observation variable's `PRINTFORMAT` is used. When nesting variables, the `PRINTFORMAT` of the variable lower in the nesting takes precedence.

For example:

```
TABULATE HEADER  = (SALARY BY REGION BY (MIN THEN MAX))
         MINIMUM = MIN 'Lowest'
         MAXIMUM = MAX 'Highest'
         PRINTFORMATS = SALARY (D) MIN (2)MAX (2)
```

The salaries are listed with 2 decimal points and no dollar sign. To print the salaries with dollar signs and no decimal points the expression is:

```
TABULATE HEADER = (REGION BY (MIN THEN MAX) BY SALARY)
```

- When an expression contains two or more nested control variables, the `PRINTFORMAT` of the lowest variable in the nesting takes precedence.
- For the purposes of `PRINTFORMAT` precedence, the header expression has highest precedence followed by the stub and then the wafer.

Labels

When control variable values are printed, they are labeled by their value labels. If there are no value labels, the name of the variable and the individual value is displayed.

Observation variables are displayed in a summary cell labeled with the variable label. If there is no variable label, the variable name is used.

When not enough room is provided in a table for label or title information, the label or title is broken into two or more segments so that it can fit in the allotted space. The `VALUE LABELS` command defines value labels for missing values including `BLANK` and `UNDEFINED`.

Page Formatting

```
PAGETITLE = 'string'['string'['string']]
PAGELENGTH = n
PAGEWIDTH = n
COLLAPSE
```

The following options alter the default formatting of the page.

`PAGETITLE`	Specifies the title printed left justified at the top of the page. By default, the page title is the string "SIR/XS Tabulate Procedure" and the page number is printed at the top right-hand side of the page. The page title may consist of up to 3 lines. The first line contains the page number. The second and third lines are printed if there are wafers and/or if the wafers must be broken into chunks (i.e. the wafer does not fit on a single page).
`PAGELENGTH`	The maximum number of lines that can appear on a page. Specifying `PAGELENGTH = NOEJECT` turns off paging. The default `PAGELENGTH` value is the current output file page length. This must include space for: the page title and page number line(s), the wafer title, wafer label, and the blank lines between, the table (wafer) itself and any `WAFERNOTES` specified.
`PAGEWIDTH`	Controls the width of the printed page. By default, tables are printed centred within `PAGEWIDTH` print positions. The default value is the current output file page width.
`COLLAPSE`	If a wafer exceeds either `PAGELENGTH` or `PAGEWIDTH`, an attempt is made to break the wafer into multiple 'chunks', each printed on a separate page. The chunks appear in the output in order from left to right, then top to bottom. The wafer and stub labels appear as needed. The `COLLAPSE` option allows chunks of broken up wafers to be printed on the same page. No more than 2 chunks are printed per page. The default is not to collapse chunks into a single page.

Header Formatting

```
HEADERWIDTH = n
HEADERINDENTATION = n
HEADERDIVIDER = 'character'
NODIVIDERS
NOHEADERCENTER
```

The following options alter the default formatting of the column header.

`HEADERWIDTH`	Specifies the width to be used in printing each column of the table. This width does not include the column divider. The default header width is 10 print positions. If `HEADERWIDTH` is set too small, there may be insufficient room to print a number in the table, in which case the field is filled with `X`s. Also, if `HEADERWIDTH` is too small, there may not be enough room for the header labels to be printed and they may be severely segmented. Conversely, if `HEADERWIDTH` is set too large, the table header may contain too many print positions for one page. When this occurs, the page is broken into chunks. Chunks are printed on successive pages left to right, top to bottom.
`HEADERINDENTATION`	Specifies the number of print lines for each level of the header nesting. The default is 3 and it cannot be set smaller.
`HEADERDIVIDER`	Specifies the character used as the column divider. The default is a vertical bar (|). This character is used for both the label area of the header as well as between the columns of data.
`NODIVIDERS`	Sets the divider character between columns of data to blank. It does not affect the label area of the header.
`NOHEADERCENTER`	Left justifies header labels in a column. By default header labels are centred in the column.

Stub Formatting

```
STUBTITLE = option 'string'
STUBWIDTH = n
STUBINDENTATION = n
STUBCONTINUATION = n
STUBHEIGHT = n
STUBFILLER = 'character'
STUBDIVIDER = 'character'
```

The following options alter the default formatting of the stub:

`STUBTITLE`	Specifies a title which appears in the boxed in area to the left of the header and directly above the stub labels. The **options** allowed are `CENTER`, `LEFT` and `RIGHT` and control the justification of the stub label. If the **option** is omitted, it is assumed to be `CENTER`. If the stub title is too long, it is automatically segmented over two or more lines. There is no default stub title.
`STUBWIDTH`	Specifies the number of print positions for stub labels. The default is 20 print positions. If "too much" stub label segmentation occurs, `TABULATE` aborts with a **`WAFER/STUB OUTPUT FORMATTING ERROR.`** The solution is to increase the `STUBWIDTH`.
`STUBINDENTATION`	Specifies the indentation for each level of nesting. The default is 2 print positions.
`STUBCONTINUATION`	Specifies the indentation for continuation lines. A label which exceeds the stub width is segmented into two or more lines. Continuation lines are indented by the parameter. The default indentation is 3 print positions.
`STUBHEIGHT`	Specifies the number of print lines used for each stub label. The default is 1 line. Label segmentation works independently from the `STUBHEIGHT` parameter and, therefore, some stub labels may occupy more than `STUBHEIGHT` lines.
`STUBFILLER`	Stub labels (on lines which contain data) are filled with the character to the full width of the stub. The default stub filler is a period (.).
`STUBDIVIDER`	Specifies the horizontal divider character. The default is the dash (-).

Stub Formatting

```
TABULATE STUB = (AGE BY SEX) HEADER = (REGION.T)
         STUBTITLE = 'Age by Sex'
         STUBWIDTH = 20
```

Age by Sex	TOTAL	North	South
Young			
Male............	5	3	2
Female..........	5	2	3
Old			
Male............	8	6	2
Female..........	3	1	2

```
TABULATE STUB = (AGE BY SEX) HEADER = (REGION.T
         STUBFILLER = ' '
```

	TOTAL	North	South
Young			
Male	5	3	2
Female	5	2	3
Old			
Male	8	6	2
Female	3	1	2

Wafer Formatting

```
WAFERTITLE = option 'string'
NOWAFERTITLE
WAFERCENTER
WAFERSTART = n
WAFERINDENTATION = n
WAFERCONTINUATION = n
WAFERHEIGHT = n
```

When a tabulate statement contains a wafer expression, each wafer produced contains a wafer label in the upper left corner below the wafer title. If there is no wafer expression in the tabulate statement, a wafer label is not printed.

Wafer labels are formed in the same way that header and stub labels are produced.

`WAFERTITLE`	Specifies a wafer title printed above each wafer. By default, the tabulate statement is printed as the wafer title, centred and one line above the wafer (table). The **option** may be CENTER, LEFT or `RIGHT` to position the wafer title over the wafer.
`NOWAFERTITLE`	Suppresses printing of the wafer title.
`WAFERCENTER`	Centres the wafer on the page. By default wafers are left justified on the page.
`WAFERSTART`	Specifies the starting print position for the wafer. Does not apply when `WAFERCENTER` is specified.
`WAFERINDENTATION`	Specifies the indentation used for each level of nesting in the wafer title. The default is 2 print positions.
`WAFERCONTINUATION`	Specifies the indentation for continuation lines when a wafer label is broken up into multiple lines. The default is 3 print positions.
`WAFERHEIGHT`	Specifies the number of print lines for the wafer label. The default is 1. This height is automatically adjusted if a label does not fit in the width of the wafer title and has to be segmented.

The number of print positions available for printing the wafer label is the same as the width of the stub titles. If the wafer title does not fit, `TABULATE` reports a **`WAFER/STUB OUTPUT FORMATTING ERROR`**. The solution is to increase the size of `STUBWIDTH`.

Wafers

The following table would consist of four wafers labeled as follows:

```
TABULATE  WAFER = (AGE BY REGION) STUB = (SEX) HEADER = (RACE)
         Under 21
          North.......

         Under 21
          South.......

         21 and Over
           North.......

         21 and Over
           South.......
```

Other Options

```
HTML

NOBOTTOMBORDER

NOLEFTBORDER

NORIGHTBORDER

NOTES     = 'string'
FOOTNOTES = 'string'

WAFERNOTES = 'string'

NOZEROS

ZEROCHARACTER = 'character'

NOEMPTYROWS

DELETEMISSING

UPPERCASE

SPANNERS

DUMMY

  = variable 'label' ...
LANGUAGE
```

HTML

Specifies that the output file contains data in html format for viewing by a browser or other package which expects this format.

When output is in html format, then the print formatting, page formatting and many of the general output control options can be specified but have no effect. There is no control over the detailed appearance of the table. The general shape of the table is dictated by the table expressions and the specifics of the table appearance depend on the software used to view and/or print the resulting html.

Border Generation

By default all borders are generated. The `NOBOTTOMBORDER`, `NOLEFTBORDER` and `NORIGHTBORDER` options alter the production of the left, right and bottom borders of each wafer. They can be used in conjunction with the `HEADERDIVIDER` and `STUBDIVIDER`.

Footnotes

There are no default footnotes. `NOTES` specifies text printed at the end of the Table of Contents which is produced if there are multiple wafers or a `NOTES` option specified. `FOOTNOTES` specifies text printed on a new page following all wafers. `WAFERNOTES` specifies text printed at the end of each wafer or chunk of a wafer.

Zero Printing

By default zeros are printed as a 0 (or as specified in the `PRINTFORMAT` statement). Specify `NOZEROS` to print zeros as dashes (-). Specify `ZEROCHARACTER` to print another character for zero cells.

Suppression of Empty Rows and Wafers

An empty row or wafer is one that contains zero in all cells. By default empty rows are printed. In a table with a single wafer or with no wafer expression at all, the table is printed whether it is empty or not. If multiple wafers are produced, empty wafers are automatically deleted. The deletion is noted in the table of contents. Empty columns cannot be deleted. `NOEMPTYROWS` suppresses the printing of empty rows.

Exclusion of Records with Missing Values

Specify `DELETEMISSING` to exclude records where any of the cross tabulation variables have missing or undefined values in a variable.

Converting Text to Uppercase

Specify `UPPERCASE` to convert text to uppercase. By default, both upper and lower case are used.

Spanner Labels

Specify the `SPANNERS` option to produce spanner labels for all control variables. Specify spanner labels on a per variable basis with the `.SPAN` modifier. When the `SPANNERS` option is in effect, suppress spanner labels for an individual variable with the `.NOSPAN` modifier.

Dummy Spanner Labels

Specify the `DUMMY` clause to create label information that is not related to any specific variable. The `DUMMY` clause creates a dummy variable whose sole purpose is to carry label information into the table. Dummy spanner labels can be used to span concatenated expressions.

Example: Dummy Spanner Labels

```
TABULATE HEADER = (J BY (SEX THEN REGION))
         DUMMY  = J 'Sex and region Data'
```

	Sex and region Data			
	Male	Female	North	South
Dummy..........	13	8	12	9

Language

If a language is specified, the effect is to produce translated versions of the following words or phrases:
SIR/XS Tabulate Procedure, Footnotes, Notes,Page, empty/deleted, Table of Contents, cont'd, of, Wafer, and, Chunk.
Recognised language keywords are HEBREW, DUTCH and GERMAN.

String Specifications

Whenever an option includes the specification of strings or labels, a string can be specified in multiple segments using the following format:

```
'segment' option 'segment'
```

Option can be a plus (+), blank or minus(-); the different characters specify where strings can be broken across lines:

Plus (+) simply concatenates two segments; blank forces a line break between the two segments; minus (-) specifies optional segmentation as per the following rules.

Label Segmentation

If a label does not fit, it is broken into segments by splitting the text at an appropriate breakpoint. The breakpoint is chosen by searching the label from right to left looking for one of the following conditions taken in sequence:

- Forced break.
- Conditional break.
- Space, hyphen, or underscore.
- Vowels. If a vowel is found, the break occurs after the vowel, with that segment followed by a hyphen when printed.

If no breakpoint is found, the label is truncated at a point which allows a hyphen to be printed.

After the first line segment has been formed, all the remaining characters of the original label to the right of the break character are considered next. If the remaining segment fits in the allocated space, it is printed, otherwise, the first **n** characters of this portion of the label (where **n** is the space available) are also scanned from right to left and segmented as described above. This process continues until the entire label has been printed.

Error Processing

There are two stages at which errors in the `TABULATE` command are detected, during compilation and during execution. Errors detected during compilation cause error messages to be displayed and the VisualPQL program is not executed. Errors detected during execution cause the program to terminate.

If during compilation, a `CELL CONTENTS ERROR` error occurs, check that:

- Not more than 1 observation variable or statistic has been nested into the same table cell.
- That there is an observation variable nested with each cell that contains statistics such as mean and median.
- The base marker(s) have been specified for a percentage table.

The table is formatted after the program is run, and some problems are diagnosed at this stage. The error `WAFER/STUB OUTPUT FORMATTING ERROR` implies that there is a problem inserting the wafer and/or stub labels within their allocated space in each wafer. Usually there are too many levels of nesting, within the wafer or stub, for the width of the `STUBWIDTH`.

If there is not enough room on a page to print an entire row or column, `TABULATE` breaks the wafer into + "chunks". If a `BY` clause has been specified, all of the rows or columns beneath the highest level in a nesting must fit on a single page, otherwise a `CHUNK FORMATTING ERROR` is reported. This implies that a wafer is too large to print on a single page and that it cannot be broken at any 'clean' place to produce multiple 'chunks' of output. A wafer can be broken into chunks only along the highest variable nested within the stub or header. The solutions are to:

- Reorganise the `TABULATE` statement, or
- Change the size formatting options to fit more of each wafer onto each page.

WRITE RECORDS

The `WRITE RECORDS` procedure produces a fixed format data file and serves as a general purpose interface to other programs. The WRITE command in VisualPQL also creates output files. The `WRITE RECORDS` procedure allows an output file to be written in a different sequence from the input data.

```
WRITE RECORDS  FORMAT    = ( format_specifications )
     [ FILENAME   = filename ]
     [ VARIABLES  = varlist | ALL ]
     [ LRECL      = number ]
     [ MISSCHAR   = char  ]
     [ SHOWMISS ]
     [ UPPERCASE     ]
     [ SORT       = [(n)] variable  [(A)|(D)] , ...]
     [ BOOLEAN    = ( logical_expression )]
     [ SAMPLE     = fraction ]
```

`FORMAT`

The `FORMAT` clause describes the format of the variables output to the file. Every output variable requires a format specification. The order of the `FORMAT` clause specifications matches the order of variables on the `VARIABLES` clause. The format specification list is enclosed in parentheses.

```
FORMAT =  (var1_fmt,var2_fmt,var3_fmt) /
VARIABLES = VAR1 VAR2 VAR3
```

The output record length may not exceed the value of `LRECL` (Logical Record Length), the default for which is 80 characters. Numbers in the output record are right justified within the field size, strings are left justified. Data padding and spacing is with blanks.

The following `FORMAT` specifications may be used:

`Fw.d`	**F**loating point field, **w** characters wide, **d** characters to the right of the decimal point. The decimal point takes up one of the **w** characters.
`Iw`	**I**nteger field, **w** characters wide, no decimal point.
`Aw`	**A**lphanumeric (string) field, **w** characters wide.
`Ew.d`	**E**xponential (scientific) notation **w** characters wide, **d** characters to the right of the decimal point. The decimal point takes up one of the **w**

	characters.
`Bw`	String, printed **B**ackwards (reversed), right-justified and blank filled on the left.
`nX`	Skip **n** characters on the output record.
`Tn`	**T**ab to column **n** before writing the next variable. Tabbing may only be done to the right.
`'text'`	The specified string enclosed in quotes is written to the output record.

If several variables have the same format, a repeat count can be used, as in 5F4.1 or 3I6. Parentheses can be used to specify groups of repeating format specifications as in 3(1X,I4,2A10). Parenthetical expressions can be nested up to 10 levels deep. For example:

```
VARIABLES = TEST1 TEST2 TEST3 /
FORMAT    = 3(I3)

VARIABLES = NAME1 TEST1 NAME2 TEST2 /
FORMAT    = 2(A15 , I3)
```

If the number of variables in the variable list exceeds the number of format fields, the excess variables are output using the format over again from the last nested left parenthesis. Each time the end of the format is encountered, a new line is started in the output file.

`FILENAME`	Specify the filename created by the procedure. If this is not specified, the default output file is used in batch or the scrolled output buffer is written to in an interactive session.
`VARIABLES`	Specifies the procedure variables written to the output data file. The order in which variables are specified is the order in which they appear in the output file. If this option is not specified, the default variable list is used.
`LRECL`	Specifies the logical record length written to the output file produced by `WRITE RECORDS`. The default is 80 characters. The format clause must not specify lines that exceed `LRECL` characters.
`MISSCHAR`	Specifies the character used as the Missing or Undefined value indicator. The default is a blank. For example, to have all missing values represented by asterisks, specify: `MISSCHAR = *`
`SHOWMISS`	Specifies that a variable's original missing values are printed for fields containing missing values. The default character is blank. Missing values are always excluded from totals - this option only affects printing.

`UPPERCASE`	Specifies that string values are converted to uppercase.
`SORT`	Specifies the sequence of the output. **n** is an integer that specifies the maximum number of records to be sorted. The default for this parameter is either the number of records in the database or the value specified in the sortn parameter and need only be specified if the number of records in the procedure table is greater than the default. The procedure table is sorted by the specified variables in variable list order. A variable name followed by (**A**) or (**D**) specifies that for that variable the sort is in **A**scending order (the default) or in **D**escending order.
`BOOLEAN`	Specifies which procedure table records are used by the procedure. The procedure table records for which the logical expression is true are used by the procedure. If this option is not specified, all procedure table records are used.
`SAMPLE`	Specifies that a random sample of the procedure table records are used by the procedure. The **fraction** specifies the percent of records used and is specified as a positive decimal number less than or equal to 1 (one). .25, for example specifies that a 25% sample be used.

Example

In this example, `WRITE RECORDS` creates a file named `SCHOOL` having 2 lines for each procedure table record.

```
WRITE RECORDS FILENAME = SCHOOL
              VARIABLES= CASENO NAME SCHOOLNO SCHLNAME
                         AGE GRADE SEX RESP1 TO RESP5
              FORMAT   = (2(I6,A20),2I2,I1,/,5I1)
```

The first line of each pair contains the variables `CASENO` through `SEX` in the format (I6,A20, I6, A20, I2, I2, I1). The second line contains the variables `RESP1` to `RESP5` in the format (I1, I1, I1, I1, I1).

XML SAVE FILE

```
XML SAVE FILE
  FILENAME = filename
  [BOOLEAN = (logical expression)]
  [MISSCHAR = character]
  [SHOWMISS]
  [SAMPLE   = fraction ]
  [SORT    = [(n)]  variable [(A)|(D)] ,  ....]

  [ROOT       = 'string']
   BREAK      = break_variable (TAG = 'string',
                               ATTRIBUTES = (varname (format) ),...),
                               ELEMENTS   = (varname (format) ),...)),
               break_variable ..........
  [DTD        [= filename]]
  [SCHEMA     [= filename]]
```

The `XML SAVE FILE` procedure produces a text file which is an Extensible Markup Language, abbreviated XML, document. The XML File is a text file and consists of an hierarchical set of tags which enclose lower levels of tags and, at some point, enclose data. It resembles HTML, only it uses tags defined by users. Many products can now deal with XML based files. (Note. Statements in this document do not purport to describe XML or make comprehensive statements about the language but some explanation is necessary to describe the clauses on the procedure command. There are published standards for XML for those interested.)

An example bit of XML from within a document might be:

```
<company>
   <person>
       <name>John D Jones</name>
       <salary>2150</salary>
       <birthday>1986</birthday>
   </person>
   <person>
       <name>James A Arblaster</name>
       <salary>1500</salary>
       <birthday>1981</birthday>
   </person>
</company>
```

XML has its own standard for names (which is different to SIR/XS) and any names that are generated by this procedure should meet this standard. XML names begin with alphabetic character (or underscore _) and should not start with `XML`. They are case sensitive and allow letters, numbers plus some special characters but no spaces.

`FILENAME`	Specify the filename created by the procedure. If you do not supply a file extension, then .xml is added to the filename as a suffix.
`BOOLEAN`	Specifies which procedure table records are used by the procedure. The procedure table records for which the logical expression is true are used by the procedure. If this option is not specified, all procedure table records are used.
`MISSCHAR`	Specifies the character used as the Missing or Undefined value indicator. The default is a blank. For example, to have all missing values represented by asterisks, specify: `MISSCHAR = *`
`SAMPLE`	Specifies that a random sample of the procedure table records are used by the procedure. The **fraction** specifies the percent of records used and is specified as a positive decimal number less than or equal to 1 (one). .25, for example specifies that a 25% sample be used.
`SHOWMISS`	Specifies that a variable's original missing values are printed for fields containing missing values. The default character is blank. Missing values are always excluded from totals - this option only affects printing.
`SORT`	Specifies the sequence of the output. **n** is an integer that specifies the maximum number of records to be sorted. The default is either the number of records in the database or the value specified in the sortn parameter and need only be specified if the number of records in the procedure table is greater than the default. The procedure table is sorted by the specified variables in variable list order. A variable name followed by (**A**) or (**D**) specifies that for that variable the sort is in **A**scending order (the default) or in **D**escending order.
`ROOT`	The XML file consists of a well formed hierarchy and the `ROOT` is the top-level outermost component of this. This defaults to `SIR_XS_ROOT` if not specified. Specify a valid XML name as the

root that the processing application expects.

`BREAK` The `BREAK` clause determines the hierarchy of the XML document. Each variable listed on the clause means one further level of nesting. The first variable is the outer level. By default the variable name is used as the tag. Specify a `TAG =` to override this.
There are two ways in which data can be included in an hierarchical level. There can be a number of individual data `ELEMENTS` or a set of `ATTRIBUTES` can be specified. Both of these name data variables but they appear in a different way in the output. Elements appear as individually tagged items whereas attributes appear within the start tag. For example, if there are three data variables for a person `Name, Salary, Birthday` then using elements, the output looks like:

```
   <person>
   <NAME>John D Jones</NAME>
   <SALARY>2150</SALARY>
   <BIRTHDAY>01 15 78</BIRTHDAY>
   </person>
   <person>
   <NAME>James A Arblaster</NAME>
   <SALARY>2650</SALARY>
   <BIRTHDAY>12 07 82</BIRTHDAY>
   </person>
```

Using attributes the output looks like:

```
   <person NAME="John D Jones" SALARY="2150"
BIRTHDAY="01 15 78">
   </person>
   <person NAME="James A Arblaster" SALARY="2650"
BIRTHDAY="12 07 82">
   </person>
```

If designing an XML application from scratch, this may be a matter of style and choice. If supplying a file to an existing application, then it is a matter of matching a specification.

The name of the element or attribute is the variable name. If this does not match the tag required, you can alter this as per the standard method for specifying variable lists in procedures (e.g. `S(1) AS SALARY` or `S(1) 'SALARY'`). Specify any formatting to be applied to the data as per the normal formats specified on a `WRITE` command.

`DTD` Additional files may be produced which describe the XML file written. You can specify a DocumentType Definition or DTD. XML provides an application independent way of sharing data. With a DTD, independent groups of people can agree to use a common DTD for interchanging data. Your application can use a standard DTD to verify that data that you receive from the outside

	world is valid. If you specify the `DTD` keyword, the default filename is the name of the main XML file produced with the extension .dtd
`SCHEMA`	XML Schema is an XML based alternative to DTD. You can produce an XSD file which describes the XML. If you specify the `SCHEMA` keyword, the default filename is the name of the main XML file produced with the extension .xsd. Specifying either Schema or DTD changes the header information written to the main XML file and so informs other processes that a descriptive file exists. You may find other applications that process the XML need a certain style of descriptive file.

Example

```
RETRIEVAL /PROGRESS
. PROCESS CASES ALL
.   get vars id
.   PROCESS RECORD EMPLOYEE
.     GET VARS NAME BIRTHDAY SALARY
.     PERFORM PROCS
.   END PROCESS RECORD
. END PROCESS CASES
XML  SAVE FILE
     FILENAME = "c:\sirxs\alpha\XML2.XML"
     ROOT = 'company'
     BREAK = ID (TAG = 'person' ATTRIBUTES = (name salary birthday))
     SORT  = ID
     schema

END RETRIEVAL
```