Quantum User’s Guide

Volume 3 Advanced Tables

TUM90518U3

SPSS Limited
Maygrove House
67 Maygrove Road
LONDON
NW6 2EG
England

Please address any comments or queries about this manual to the Support Department at the above address, or via e-mail to: support-uk@spssmr.spss.com

Contents

List of figures

About this guide

The Quantum User's Guide is written primarily for Quantum spec writers. It is also a useful reference for Quanvert database administrators and others who prepare data for use with Quanvert or Quanvert Text.

This guide is not intended as a tutorial or teach-yourself document. Instead, it provides a complete and detailed description of the Quantum language and the Quantum programs. However, the guide has been designed with your needs in mind. If you are an experienced user, you will find the Quick Reference boxes at the start of each section helpful as a reminder of syntax. If you are less experienced, you will probably prefer the more detailed explanations and examples in the main body of each section.

The Quantum User's Guide is divided into four volumes, which are described in more detail below. All the volumes contain a comprehensive index that covers all four volumes.

Volume 1, Data editing

Volume 1 of the Quantum User's Guide covers data editing, validation and cleaning:

- Chapters 1 to 3 give you an overview of the language and explain the basic concepts of Quantum spec writing.
- Chapter 4, 'Basic Elements', describes constants, numbers and variables.
- Chapter 5, 'Expressions', describes arithmetic and logical expressions.
- Chapter 6, 'How Quantum reads data', describes types of records, data structure, trailer cards, reserved variables, merging data files and reading non-standard data files.
- Chapter 7, 'Writing out data', describes creating a new data file, copying records to a print file, and writing to a report file.
- Chapter 8, 'Changing the contents of a variable', describes the Quantum assignment statements, adding and deleting codes in a column, forcing single-coded answers, setting a random code in a column, reading numeric codes into an array and clearing variables.
- Chapter 9, 'Flow control', describes the *if* and *else* statements, routing around statements, loops, rejecting records, jumping to the tabulation section and canceling the run.
- Chapter 10, 'Examining records', describes holecounts and frequency distributions.
- Chapter 11, 'Data validation', describes the *require* statement, column and code validation, and validating logical expressions.

- Chapter 12, 'Data correction', describes forced cleaning, on-line data correction, creating clean and dirty data files, correcting data from a corrections file, and missing values in numeric fields.

- Chapter 13, 'Using subroutines in the edit', describes how to call up subroutines, the subroutines in the Quantum library, writing your own subroutines and calling functions from C libraries.

- Chapter 14, 'Creating new variables', describes how to name and define variables in your Quantum spec.

- Chapter 15, 'Data-mapped variables', describes the data-mapped variables feature.

- Chapter 16, 'Running Quantum under Unix and DOS', describes how to compile and run your Quantum program.

Volume 2, Basic tables

Volume 2 of the Quantum User's Guide covers axes and creating basic tables:

- Chapter 1, 'Introduction to the tabulation section', provides an introduction to creating tables in Quantum.

- Chapter 2, 'The hierarchy of the tabulation section', describes the components of a tabulation program, the hierarchies of Quantum, how to define run conditions, the options that are available on the *a*, *sectbeg*, *flt* and *tab* statements, the default options file and some sample tables.

- Chapter 3, 'Introduction to axes', describes how to create an axis, the types of elements within an axis, how to define conditions for an element, the *n* count creating elements, subheadings, netting and axes within axes.

- Chapter 4, 'More about axes', describes the *col*, *val*, *fld* and *bit* statements, filtering within an axis, and options on axis elements.

- Chapter 5, 'Statistical functions and totals', describes totals, averages, means, the standard deviation, standard error and error variance statements and how to create percentiles.

- Chapter 6, 'Using axes as columns', describes special considerations for when axes are used for the columns of a table.

- Chapter 7, 'Creating tables', describes the syntax of the *tab* statement, multidimensional tables, multilingual surveys, combining tables, printing more than one table per page, and suppressing percentages and statistics with small bases.

- Chapter 8, 'Table texts', describes table titles, underlining titles, printing text at the foot of a page, table and page numbers and controlling table justification.

- Chapter 9, 'Filtering groups of tables', describes general filter statements, named filters and nested filter sections.
- Chapter 10, 'Include and substitution', describes filing and retrieving statements, symbolic parameters and grid tables.
- Chapter 11, 'A sample Quantum job', provides an example of a Quantum specification and the tables it produces.
- Appendix A, 'Limits', describes the limits built into Quantum.
- Appendix B, 'Error messages', contains a list of compilation error messages with suggestions as to why you may see them and how to solve the problems which caused them to appear.
- Appendix C, 'Options in the tabulation section', provides a summary of the options available in the tabulation section.

Volume 3, Advanced tables

Volume 3 of the Quantum User's Guide covers advanced tables and statistics:

- Chapter 1, 'Weighting', describes the weighting methods that you can use in Quantum.
- Chapter 2, 'Row and table manipulation', describes how to create new rows and tables using previously created tables or parts of previously created tables.
- Chapter 3, 'Dealing with hierarchical data', describes how to use analysis levels in Quantum.
- Chapter 4, 'Descriptive statistics', describes the axis-level and table-level statistical tests that are available in Quantum and provides details of the chi-squared tests, non-parametric tests on frequencies and Friedman's two-way analysis of variance.
- Chapter 5, 'Z, T and F tests', describe the Z, T and F tests that are available in Quantum.
- Chapter 6, 'Other tabulation facilities', describes how to include C code and edit statements in the tabulation section and how to sort tables.
- Chapter 7, 'Special T Statistics', describes the special T statistics that are available in Quantum.
- Chapter 8, 'Creating a table of contents', describes how to create a formatted list of the tables that are produced by a Quantum run.
- Chapter 9, 'Laser printed tables with PostScript', describes how to convert the standard tabulation output into a file suitable for printing on a PostScript laser printer.
- Appendix A, 'Options in the tabulation section', provides a summary of the options available in the tabulation section.

Volume 4, Administrative functions

Volume 4 of the Quantum User's Guide covers administrative functions:

- Chapter 1, 'Files used by Quantum', describes files you may need to create in order to use certain Quantum facilities, including the variables file, the levels file, the default options file, the run definitions file, the merges file, the corrections file, the rim weighting parameters file, and the C subroutine code file, aliases for Quantum statements, customized texts, and user-definable limits.

- Chapter 2, 'Files created by Quantum', describes many of the files created during a run and draws your attention to those of particular interest.

- Chapter 3, 'Quantum Utilities', describes how to tidy up after a Quantum run and how to check column and code usage.

- Chapter 4, 'Data conversion programs', describes the q2cda and qv2cda programs that convert tables into comma-delimited ASCII format, the qtspss and nqtspss programs that convert Quantum data into SPSS format, and the qtsas and nqtsas programs that convert Quantum data into SAS format.

- Chapter 5, 'Preparing a study for Quanvert', describes the tasks you need to perform before converting a Quantum spec and data file into a Quanvert database.

- Chapter 6, 'Files for Quanvert users', describes files that are specific to either Quanvert Text or Windows-based Quanvert.

- Chapter 7, 'Creating and maintaining Quanvert databases', describes how to create and maintain Quanvert databases.

- Chapter 8, 'Transferring databases between machines', describes how to transfer databases between machines and the programs provided to help you achieve this.

- Appendix A, 'Limits', lists limits built into Quantum.

- Appendix B, 'Error messages', contains a list of compilation error messages with suggestions as to why you may see them and how to solve the problems that cause them to appear.

- Appendix C, 'Quantum data format', describes the Quantum data format.

- Appendix D, 'Using the extended ASCII character set', explains how you can use Quantum with data that contains characters in the extended ASCII character set.

- Appendix E, 'ASCII to punch code conversion table', provides a table showing ASCII to punch code conversions.

- Appendix F, 'Will this job run on my machine', offers suggestions on how you can check whether a particularly large job will run on your computer.

Symbols and typographical conventions

Words which are keywords in the Quantum language are normally printed in *italics* in the text. In the main description of each keyword, the keyword is shown in **bold** the first time it is mentioned.

When showing the syntax of a statement, as in the Quick Reference sections, all keywords are printed in **bold**. Parameters, such as question texts or responses, whose values are user-defined are shown in *italics*. Optional parameters are enclosed in square brackets, that is, [ ].

All examples are shown in `fixed width type`.

The ✎ symbol marks a note or other point of particular interest.

The ☞ symbol marks a reference for further reading related to the current topic.

Comments

SPSS MR welcomes any comments you may have about this guide, and any suggestions for ways in which it could be improved.

1 Weighting

Sometimes in surveys we treat the respondents as representatives of the total population of which they are a sample. Normally, tables reflect the attitudes of the people interviewed, but we may want the tables to reflect the attitudes of the total population instead, so that it seems as if we had interviewed everyone rather than just a sample of the population. This, of course, assumes that the people interviewed are a truly representative sample.

If we take a sample of 380 from a population of 10,000 middle-aged housewives, and discover that 57 members of this sample buy cheddar cheese, we may want the number of middle-aged housewives who buy cheddar cheese to read 1,500 in our tables, not 57.

Moving from 57 to 1,500 is the fine art of weighting. In this case, each middle-aged housewife has a weight of 10,000/380. Since 57 of them buy cheddar cheese, the number in the cell will be:

$10000 / 380 \times 57 = 1{,}500$

Weighting is also used to correct biases that build up during a survey. For example, when conducting interviews by telephone you may find that 60% of the respondents were women. You may then want to correct this ratio of men to women to make the two groups more evenly balanced.

The basic idea behind weighting is that when someone falls into a given cell (that is, satisfies the conditions for that cell) the number in the cell is not increased by 1; rather, it is increased by 1 multiplied by the individual's weight.

1.1 Weighting methods

Quantum is sufficiently flexible to allow more than one set of weights for a given set of respondents. Which set is applied is determined by options on the *a*, *sectbeg*, *flt* or *tab* statement or on the statements which create the individual rows or columns of a table. Each set of weights, however, will apply one weight for each respondent. There are two ways of calculating weights:

- The weight for each respondent may be part of the data for that respondent, or it may be calculated in the edit and passed to the tabulation section as a variable.

- The more common method of weighting is to define a set of characteristics and apply specific weights to respondents satisfying those characteristics.

Our example above uses characteristic weighting, where the characteristics are age, sex and working status. Thus, all respondents who are women aged between 45 and 54 and who do not work outside the home receive a weight of 10,000/380.

The characteristics must be such that each record satisfies one unique set. Each respondent falls into one, and only one, set and no respondent is left out. Because of this, you must check all columns containing the characteristics and if necessary, correct any errors. For example, if one characteristic is sex and it is coded in column 6 of card 1, with a code of 1 for male and 2 for female, you must make sure that c106 is single coded with a '1' or a '2' only. It must not be blank, multicoded or otherwise miscoded in any way.

Any respondent who is present in the base of the weighting matrix but not in any other row or column of the matrix will be given a weight of 1.0, and his or her record will be printed in the print file with the message 'unweighted'.

1.2 Types of weighting

Quantum offers factor, target and rim weighting, preweights, postweights, weighting using proportions and weighting to a given total. These are described, with examples, in the sections which follow. The keywords used to write the weighting matrices are described later in this chapter.

Factor weighting

With factor weighting, every record which satisfies a given set of conditions is assigned a specific weight. You would generally use it when the weights are calculated outside of Quantum — for instance, you may be told that all unemployed people in London require a weight of 10.5, whereas unemployed people in the rest of the country need a weight of 7.3.

When Quantum creates the weighted table, it will check which cell of the weighting matrix each respondent belongs in, and will apply the weight associated with that cell before placing the respondent in the table.

You can also use factor weighting, with a factor of 1.0, when you just want to use weights stored in the data or calculated in the edit, without defining any other weights. These weights are defined as preweights.

☞ For an example, see section 1.5, 'Using weights in the record alone'.

Target weighting

Target weights may be used when you know the exact number of respondents you want to appear in each cell of the weighted table. For example, in a table of age by sex, you may know the exact number of men under 21, women under 21, and so on, to appear in the table once it has been weighted. The weights that you define in your matrix are therefore the values to appear in the weighted table rather than the weights to be applied to each respondent of a given age and sex.

When Quantum creates your weighted table, it calculates the weight for an individual respondent by taking the target figure for the appropriate cell in the weight matrix and dividing it by the number of respondents in that cell.

As an example, suppose that you have three groups of people. The first contains 100 people, the second contains 200, and the third contains 300. You know that in the total population, the spread of any 600 respondents across these three groups would be 150, 200 and 250. When Quantum finds someone in the first group it will apply a weight of 1.5 (150/100) in order to obtain the total of 150 respondents in the weighted table. Respondents in the second group will have a weight of 1.0 because the number of respondents in this group matches the value in the weighting matrix for that group. Respondents in the third group will have a weight of 0.83 (250/300) because there are more people in that group than in the corresponding cell of the weighting matrix.

In this example, the number of people in our three groups was the same as the population defined in the weighting matrix. This will not always be the case. Often you will find that the values in the weighting matrix add up to more or less than the number of people you have in your sample. For instance, the spread of the population across your three groups may be 150, 250 and 250, giving a total of 650 respondents. When Quantum balances your sample, it will weight each respondent according to the values in the matrix so that the total number of respondents in your weighted table will be 650, rather than the 600 that were interviewed.

If you decide that you want the total in the weighted table to be the same as the total number of respondents in your sample, you may define this total as part of the weighting matrix using the keyword *total=* which is described below. When Quantum reads this keyword it balances the three groups according to the weights in the matrix and then adjusts all three weights so that the weighted total is 600.

Another variation of target weighting occurs when instead of knowing the actual number of people in each group of the population, you know that each group is a given percentage of the population. For instance, the first group may be 27% of the population, the second may be 48%, and the third may be 25%. In cases like this, you include the keyword *input* (see below) in the weighting matrix with the percentages for each group.

Rim weighting

Rim weighting is used when:

- You want to weight according to various characteristics, but do not know the relationship of the intersection of those characteristics, or

- You do not have enough respondents to fill all the possible cells of the table if you were to weight the data using the multidimensional technique described above.

For example, you may want to weight by age, sex and marital status and may know the weights for each category of those characteristics; for example, people aged 25 to 30, men, single people.

However, you may not know the weights for, say, single men aged between 25 and 30, married women aged between 31 and 40, and so on.

On another study, you may need to weight by a large number of characteristics at the same time; for example, sex, age, race, occupation and income. Since each of these characteristics will be broken down into categories, you will require a weighting matrix with many cells. You may not have enough information to write a standard multidimensional weighting matrix which defines weights for the intersection of all these characteristics. However, as long as you have information on each category individually (for example, male, female, 21-24, 25-30, and so on) you will be able to perform the weighting required with rim weights.

Rim weighting is designed to attempt to weight all characteristics at the same time. The accuracy of your weighting will depend on how well your sample matches the known universe. If the sample is a good match, then it is likely that Quantum will generate acceptable weights; if the sample is not a good match it is possible that the weights will look perfectly acceptable when you look at the number of men or the number of married people, but will look totally unacceptable when you look at the number of married men.

As the rim weighting process runs, it tries to distort each variable as little as possible while still trying to attain all the desired proportions among the characteristics. The ***root mean square*** figure which Quantum produces will tell you how much distortion you have introduced, and therefore how reliable your sample is. The larger the number, the more the distortion and therefore the less accurate your sample is. This is discussed in more detail later in this chapter.

Another very powerful facility of rim weighting is the fact that it automatically rescales all the target values to the same base. For instance, suppose you have a sample of 5,000 respondents. Your rim weighting matrix defines:

- A weighted total (table base) of 10,000.
- Weights for age in percentages.
- Weights for sex in target numbers which add up to 758.
- Weights for occupation in numbers which add up to 1134.

Quantum will calculate the weights for these characteristics, using the figures given, and will then adjust them so that the total for the weighted table is 10,000. If you do not define a total, the weights will be adjusted to the total of the first variable defined in the matrix.

As you can see from this simple example, rim weighting can be used when you have weights coming from different sources, and when those weights do not have a common form or total.

Entering weights as proportions (input weighting)

When we were talking about target weighting, we said that sometimes you might not know the actual counts of respondents in a group, even though you may know that the group is a certain percentage or proportion of the total population. For instance, you may know that 60% of the population is women, but you may not know how many women that represents.

When this happens, you can enter the percentages or proportions as the weights for each group, and use the keyword *input* to indicate that these figures should be used as targets. For example, in a table of age by sex you would enter the proportion or percentage that each combination of age and sex is of the total population, and Quantum would calculate what weight to assign to each respondent in each category.

Weighting to a given total

When you define targets which add up to more than the number of respondents in your sample, Quantum will calculate the weights for each respondent such that the total for the weighted table equals the total of the figures in the weighting matrix. You may define your own total figure (usually the number of respondents in your sample) using the keyword **total**=*n*, where *n* is the required weighted total. Quantum will then calculate the weights according to the values in the weighting matrix and will then adjust them to match the total you have defined.

Preweights

Preweights, stored as part of each respondent's data or created during the edit, are applied to individual records before target or factor weighting is applied. When the characteristic weights are targets, the preweights are used in the calculation of the weight for each respondent. For example, suppose that each of our 380 housewives has a preweight in columns 181 to 189 of their data record: one has the value 10 in c(181,189), while for another the weight in that field is 20. If all the rest have a weight of 1, we would appear to have:

$$(10\times1) + (20\times1) + (1\times378) = 408$$

middle-aged housewives instead of the original 380.

To reach our target of 10,000, the weight for each woman would be:

$$10{,}000 \div 408 = 24.51$$

Without preweights, all these women would receive a weight of 26.32.

Preweights are often used in studies which deal with newspaper readership, or the like, where a male adult respondent in a household will be counted as the total number of male adults in the household, on the theory that the other males will probably have the same demographics and similar behavioral patterns. Another use is in political polls, where a respondent is preweighted by

the number of calls it took to reach him. The supposition behind this theory is that the more calls it takes to reach a respondent, the more people there are like him, who are equally hard to reach. The respondent must therefore be preweighted in order to help represent the many like him who were never interviewed.

Postweights

The opposite of preweights are postweights, which are applied after all other weights have been applied, and therefore have no effect on the way in which targets are reached. They are generally used to make a final adjustment to a specific item.

Suppose, for instance, that a survey was conducted in London and Inverness, and 200 respondents were interviewed in each city. The standard weighting might balance each group according to sex and age so that the samples match the patterns of the total populations in those cities. After this is done, you might apply a postweight to adjust the totals for each city into their correct relative proportions, where London has a much larger population than Inverness.

Excluding respondents from the weighting

Although the *a* statement is the first statement in the tabulation section, weights are calculated before any conditions specified on the *a* statement are applied. A similar thing applies to filters defined on *flt, sectbeg* and *tab* statements. Therefore, filters do not exclude respondents from weighting calculations.

If you want to exclude respondents from the weighting, either:

- use *reject;return* in the edit section to reject them from the whole tabulation section, or
- create a cell in the weighting matrix for those respondents and give them a weight of zero.

1.3 Defining weights in a weighting matrix

Quick Reference

To define a weighting matrix, type:

wm*number axis_names*[**;***weight_type*][**;maxwt=***max*][**;minwt=***min*][**;***options*]**;***weight1***;***weight2***;...**

There are two ways of defining characteristic weights. You may either set up a weight matrix that declares the weighting conditions and the weights to be applied, or you may declare the weights in an ordinary axis and then label that axis as a weighting axis. This section explains how to declare weights in a weighting matrix.

✎ Although you may write jobs that have weights declared in weighting matrices and in axes, the syntax for the two methods is not interchangeable. If you want to define weighting based on a combination of age and sex, say, you must either specify it all using a weighting matrix or all using an axis.

When you weight using matrices, each weighting matrix defines a set of conditions and the weights to be applied when a respondent is found having those characteristics. Matrix characteristics are specified in ordinary axes which may be used for other parts of the program, or not, as you choose. In our original example, if a respondent's exact age is stored in c(107,108) and sex is a '1' (Male) or '2' (Female) in c106, the middle-aged women have c106'2' and some arithmetic value between 45 and 54 in c(107,108). When these axes are used for weighting, base statements (*n10*, *n11* and Base on *col*, *val*, *fld*, *bit*), text statements (such as *n03*), statistical rows (such as *n12*) and unweighted elements are ignored.

Weighting matrices are defined on *wm* statements in the following format:

wm*n axis_names*[**;***options*]**;***weights*

where *n* is a unique number by which the matrix can be identified, *axis_names* are the axes defining the characteristics of this matrix, *options* are keywords defining the type of weighting required, and *weights* are the targets, factors or proportions to be used for weighting.

✎ You cannot use a grid axis on a *wm* statement. If you need to define weights specifically for a grid axis, you should define a dummy axis with as many elements as there are cells in the grid axis and use that instead.

☞ For further information, see 'Weighted grids' in chapter 10, 'Include and substitution' in the Quantum User's Guide Volume 2.

The matrix number may be any number between 1 and 9, and as long as no number is repeated, matrices may be numbered in any order.

The following table provides details of the options on the *wm* statement.

Option	Explanation
target	Targets (default).
factor	Factors.
input	Proportions.
rim	Rim weighting.
total=*n*	Cell values are to total to *n*.

Option	Explanation
pre=*var*	Preweighting using the value in variable *var*. This may be a data, integer, real or real-data variable. If the variable contains the value missing_, indicating missing data, the weight for the respondent is assumed to be zero.
post=*var*	Postweighting using the variable *var*. If the variable contains the value missing_, indicating missing data, the weight for the respondent is assumed to be zero.
maxwt=*value*	Maximum weight to be used in tables weighted with this matrix.
minwt=*value*	Minimum weight to be used in tables weighted with this matrix.
varname=*name*	Use this option when you are setting up a database for use with Quanvert to assign a name to the weighting matrix.
anlev=*level_name*	Use this option to cause weights to be applied only at the named level.
c=*firstread*	Use this option in trailer card data, to specify that weights are to be calculated only when the first card of a new record is read.
wmerrors/ nowmerrors	The *wmerrors* option is on by default. It causes the run to stop with an error if certain weight matrix errors are detected. Use *nowmerrors* to switch the option off, so that the run continues with a warning.

☞ For further information about *anlev*= and *c*= options, see 'Defining weighting matrices in hierarchical data' later in this chapter.

For further information about *wmerrors* and *nowmerrors*, see 'Weight matrix errors' later in this chapter.

The following combinations of options are not allowed on *wm* statements:

- Any combination of *target*, *factor* or *rim*
- *input* with *total*.

✎ Quantum does not actually prevent you from putting both *input* and *total* on the same weight matrix. However if you do this, without giving a warning Quantum ignores the one you specified first and uses the other one. For example, Quantum interprets:

```
wm1 sex;input;total=1000;60;40  as  wm1 sex;total=1000;60;40
wm1 sex;total=1000;input;60;40  as  wm1 sex;input;60;40
```

Let's look at a simple matrix defining weights for age and sex. The matrix will consist of targets based on population figures for the area covered by our survey:

Population of Surveyed Area		
	Sex	
Age	Male	Female
18-24	6,200	6,100
25-34	7,600	7,500
35-44	8,200	8,300
45-54	9,600	10,000
55-64	7,100	7,600
65 and Over	4,600	5,900

so we would write:

```
wm1 age sex;target;6200;6100;7600;7500; ... 4600;5900
```

If you prefer, you can enter the weights on separate lines, as the figures are to be printed in the table:

```
wm1 age sex;target;
+6200;6100;
+7600;7500;
     .
+4600;5900
```

In this example, all the weights are whole numbers, but Quantum can cope equally well with weights which are real numbers.

If several consecutive weights are the same, you may save yourself time by writing the weight out once and preceding it by an asterisk and the number of times it is to be repeated. For example:

```
3*10.5
```

tells us that three consecutive cells have a weight of 10.5.

If your tables are to be correct, it is imperative that you enter the axis names and the weights in the correct order. Axes are entered as they are for *tab* statements; that is, higher dimension axes followed by column axis followed by row axis.

Weights are entered on a row by row basis, working from left to right along the row. As you can see by comparing the numbers on the *wm* statement with those in the chart above, the first two numbers are the weights for men aged 18 to 24 and women aged 18 to 24, in that order. Note that there is no need to keep weights for different characteristics on different lines; just string them one after the other separated by semicolons on the same line. If you run out of room, continue on the next line remembering to start the line with a plus sign to tell Quantum it is a continuation.

Suppose, now, that the chart looked like this:

```
                Population of Surveyed Area

                              Age
           18-24  25-34  35-44  45-54  55-54    65+

Male       6,200  7,600  8,200  9,600  7,100  4,600
Female     6,100  7,500  8,300 10,000  7,600  5,900
```

The *wm* statement would then be written as:

```
wm1 sex age;target;6200;7600;8200; ... 7600;5900
```

If you like, think of weighting as creating a table in which you not only specify the axes to create the cell-conditions but also define the numbers to go into those cells. When a table is created that uses these weights, the program will first check which cell of the table the respondent belongs in, then it will look to see which cell of the weighting matrix refers to him or her. Finally, Quantum reads the appropriate weight from the matrix and increments the cell count in the table by this value instead of by 1.0.

If we want to use both preweights and targets, we would write:

```
wm4 work age sex;pre=c(181,189);target;1200;2400;1400; ...
```

The preweight read from c(181,189) will be applied before the targets listed on the *wm* statement.

Weight matrix errors

Quantum performs validity checks on weight matrices and provides information when certain weight matrix errors are detected. When one of these errors is detected, a message is displayed and, by default, the Quantum run stops. You can see further details of the error in the weighting report file.

If you would prefer the run not to stop when one of these errors is detected, you can use the *nowmerrors* keyword on the *a* statement or the *wm* statement. When you do this, the run continues with a warning. However, you can still find the additional information in the weighting report file.

Non-zero target weights and zero cases

It is incorrect to specify a non-zero target weight for a cell that has no cases. When Quantum encounters this error, it stops the run with a message of the form:

```
Error: Weight matrix 5, cell 3: target 5.000 given, no cases found
For details, see the Quantum weighting report file (weightrp)
```

If the *nowmerrors* keyword is in force, the same message is issued except that the word 'Error' is replaced by the word 'Warning' and the run continues.

Non-zero rim weights and zero cases

It is incorrect to specify a non-zero target weight for an element in a rim weighting dimension when there are no cases in that element. When Quantum encounters this error, it stops the run with a message of the form:

```
Error: Weight matrix 3, dimension 2, elem 3: target 4.800 given, no cases found
For details, see the Quantum weighting report file (weightrp)
```

If the *nowmerrors* keyword is in force, the same message is issued except that the word 'Error' is replaced by the word 'Warning' and the run continues.

The message is repeated in the weighting report file immediately after the line reporting the element in error, for example:

```
     INPUT          INPUT           PROJECTED      PROJECTED
    FREQUENCY      PERCENT          FREQUENCY      PERCENT
   -----------   -----------       -----------   -----------
      24.000        50.00             28.800        60.00
      24.000        50.00             19.200        40.00
   -----------   -----------       -----------   -----------
      48.000       100.00             48.000       100.00

      16.000        33.33             14.400        30.00
      16.000        33.33             19.200        40.00
       0.000         0.00              4.800        10.00
Error: Weight matrix 3, dimen 2, elem 3: target 4.800 given, no cases found.
```

Zero target with non-zero cases

It is not an error to specify a zero target for a target weighting cell or for an element in a rim weighting dimension when there are cases in that cell or element. However, when Quantum encounters either of these situations, it provides a warning message in case you specified this in error. The warning messages are of a similar form to the error messages described for when a non-zero target is found for a cell or element that contains no cases, and the messages are repeated in the weighting report file. However, Quantum does not stop the run regardless of whether *nowmerrors* has been specified or not.

☞ For further information about the weighting report file, see section 2.9, 'The weighting report file' in the Quantum User's guide Volume 4.

Defining weighting matrices in hierarchical data

Weights are calculated and applied each time a record is read in or a *process* statement is executed. However, when trailer cards are read in one at a time, the weight is calculated as if each trailer card were a new record, which can lead to incorrect weights being used. You therefore need to do one of the following:

- Use the *anlev=* option to specify the level each weighting matrix applies to.
- Use the *c=* option to limit the cards that contribute to the weighting calculations and the number of times each weight is applied.

anlev=

In hierarchical data, you can use the *anlev=* option on the *wm* statement to cause weights to be applied only at a named level. For example, the statement:

```
wm2 age hhld;anlev=person; .....
```

calculates weights at the person level rather than at household level.

It is necessary to weight a table using a weight matrix at the appropriate level, so you generally need to create a weight matrix for each level of data that is present. For example, in a survey that has three levels, hhold, person and trip:

```
wm1 waccom1;input;anlev=hhold;70;30
wm2 waccom2;input;anlev=person;70;30
wm3 waccom3;input;anlev=trip;70;30

tab accom region;anlev=hhold;wm=1
tab sex age;anlev=person;wm=2
tab mode destin;anlev=trip;wm=3

l waccom1;anlev=hhold
col 106;House;Flat

l waccom2;anlev=person
col 106;House;Flat

l waccom3;anlev=trip
col 106;House;Flat
```

c=firstread

As an alternative to using *anlev=* to specify the level the weighting matrix applies to, you can use the *c=* option on the *wm* statement to limit the cards that contribute to the weighting calculations and the number of times each weight is applied. It takes the form:

```
wm1 sex hhold;c=firstread; .....
```

This causes weights to be calculated whenever the first card of a new record is read. If the current card is not the first card of a new record, Quantum applies the last weights it calculated, that is, those calculated when the first card in this record was read.

✎ *c=* on a *wm* statement does **not** define a condition which respondents must have in order to be weighted with this matrix.

Multidimensional weight matrices

A Quantum run may contain up to nine weighting matrices (wm1 to wm9), each of which may name up to nine axes defining the conditions of that matrix.

Do not be put off by the prospect of multidimensional weight matrices: they are exactly the same as multidimensional tables. The last two axes named on the *wm* statement are the rows and columns of the table, and weights are entered with reference to the last axis.

For example, we might have:

```
wm3 work age sex;target;1200;2400;1400;2300; ....
```

where 1,200 is the target for working men aged 18-24, 2,400 is the target for working women of the same age, 1,400 is the target for working men in the age group 25-34, and so on. Not until all weights have been defined for people who work do we come onto those for people who do not work. Remember that base-creating elements are ignored.

1.4 Weighting information in axes

Quick Reference

To define a weighting target for an element, place the option:

wttarget=*number*

on the *n01* that creates the element.

To define a weighting factor for an element, place the option:

wtfactor=*number*

on the *n01* that creates the element.

An alternative to defining weight matrices is to declare weights in the axes themselves. This does not prevent you using the axes as the rows, columns or higher dimensions of tables, nor does it affect the appearance of those tables or their cell values.

✎ Elements must be specified using *n* statements since this facility does not work with *col*, *val*, *fld* or *bit* statements.

Rim weights are not supported by this method; you must specify them using a weight matrix.

To define weighting targets for the elements of an axis, add the option:

wttarget=*number*

to the *n01* statements that create the elements, where *number* is the number of respondents you want Quantum to show in the element.

The example below shows how to define targets based on sex. When you weight tables by sex, Quantum will count the number of men in the data and will calculate a weight such that the number of men matches the target for men.

```
l sex
n01Male;c=c156'1';wttarget=485
n01Female;c=c156'2';wttarget=515
n01Not answered;c=c156n'12'
```

The 'Not answered' element has no target defined so it is ignored for weighting purposes. This means that records in that element are unweighted. If you do not already have an axis whose elements define the weighting characteristics you want to use, just create the axis but do not use it on a *tab* statement.

You define factors in the same way except that you use the keyword:

wtfactor=*number*

Any elements in a weighting axis that do not contain either *wttarget* or *wtfactor* are ignored for weighting purposes.

Numbering weighting axes

Each weighting axis must have a number by which Quantum can refer to it. Type the statement:

wm*number*=*axis_name*

at the top of the tabulation section under the *a* statement. If weighting is defined in the sex axis, you could write the *wm* statement as:

```
wm1=sex
```

If you use the weighting axis as the rows, columns or higher dimension of an unweighted table the weighting specifications are ignored. For example:

```
wm1=sex
tab sex region
tab sex region;wm=1
l sex
n10Base
n01Male;c=c110'1';wtfactor=48
n01Female;c=c110'2';wtfactor=52
l region
col 123;Base;North;South;East;West
```

This specification produces two tables. The first is unweighted so the weighting information in the sex axis is ignored:

	Base	North	South	East	West
Base	17	5	4	4	4
Male	10	4	1	2	3
Female	7	1	3	2	1

The second table has the same rows and columns but the cell values are weighted using the weights in the sex axis:

	Base	North	South	East	West
Base	844	244	204	200	196
Male	480	192	48	96	144
Female	364	52	156	104	52

Quantum does not accept a weighting axis as the rows or columns of the table if the table itself is weighted using a different axis, as in the example shown here:

```
wm1=sex
tab region brand;wm=1
l region
n10Base
n01North;c=c115'1';wttarget=100
n01South;c=c115'2';wttarget=110
n01East;c=c115'3';wttarget=120
n01West;c=c115'4';wttarget=115
```

To alert you to this error Quantum issues the message 'weight line needs one target or factor' for each element in the row or column axis (in this example, for each region).

Preweights, postweights, proportions (input) and weighting to a given total

Preweights and postweights, weighting using proportions, and weighting to a given total are all requested using keywords on the *l* statement. The following table provides details of the keywords.

Keyword	Explanation
pre=*var*	Names a field or variable containing a preweight that is to be applied to the record before weighting defined in the elements to which the record belongs.
post=*var*	Names a field or variable containing a postweight that is to be applied to the record after the weighting defined in the elements to which the record belongs.
total=*n*	Defines the total value for all cells in the table. If the weights you have defined produce a weighted total that is greater than the value defined with *total=*, Quantum will reduce the weights proportionally so that the weighted total is the same as the *total=* value.
input	Indicates that the weights in the axis are to be read as proportions rather than as weights to be applied. When Quantum calculates weights it will calculate them such that the number of respondents in each element are in the given proportions.

1.5 Using weights in the record alone

Sometimes you will be dealing with data in which all weighting information is already in the record or where the weights are all calculated in the edit. The only way to weight using information from the data or edit is to use preweights because otherwise Quantum expects to read the weights from the *wm* statement. However, preweights cannot be used by themselves, so we need to set up a dummy weighting matrix as shown here. Using a weighting matrix you would write:

```
a;op=12;dsp;wm=1
wm1 axdum;pre=wtvar;factor;1
          .....
l axdum
n01
```

If you are declaring weighting in the axes themselves you would write:

```
a;op=12;dsp;wm=1
wm1=axdum
           .....
l axdum;pre=wtvar
n01;wtfactor=1
```

The weight is read from the given variable (in this case wtvar) and treated as a preweight. Since preweights must be used with targets, factors or proportions, we define a factor of 1 which will not alter the value of the preweight when the two are multiplied.

In the weight matrix version, the dummy axis, axdum, contains a single *n01* statement with no conditions to correspond to the single factor in the matrix.

If the value in wtvar is 5, the final weight for the respondent will be 5 (5×1=5).

1.6 Minimum and maximum weights

Quick Reference.

To specify the maximum and/or minimum weights you will accept, place the keywords:

maxwt=*number* and/or **minwt=***number*

either on the *wm* statement if you are using a weight matrix, or on the *l* statement or on individual elements if you are declaring weights in axes.

The options *maxwt=* and *minwt=* allow you to define the maximum and/or minimum weights that can be applied in tables using a specific weighting matrix or axis. The values you specify may be integers or reals.

If you are using weighting matrices you place these keywords on the *wm* statement; if you are specifying weights in axes you place these keywords on the *l* statement or on individual elements. If you use the same keyword with different values on an element and on the *l* statement, the setting on the element overrides the setting on the *l* statement.

When you specify *maxwt=* and/or *minwt=*, Quantum tries to ensure that the maximum and/or minimum weights used in the table match your specifications. Quantum performs the weighting calculations and adjustments as follows:

1. Calculate the weight for each cell of the table.
2. Examine each weight and compare it against the minimum and/or maximum values defined. If a weight is less than the minimum value it is set to the minimum value, if it is larger than the maximum value it is set to the maximum value. If no adjustment is necessary the weighting calculation is complete.
3. If adjustments were necessary, Quantum calculates the total obtained using the modified weights and compares it with the total obtained using the unmodified weights. If the totals are different, Quantum attempts to correct for this by adjusting the weights which were not set to *maxwt* or *minwt* and then returns to step 2.

   If all weights are set to *maxwt* or *minwt* so that no correction is possible, Quantum uses the unmodified values.

All adjustments made with this type of weighting are recorded in the weighting report file.

☞ For further information about the weighting report file, see section 2.9, 'The weighting report file' in the Quantum User's guide Volume 4.

You can use this facility with target, factor, input or rim weighting. Preweights or postweights are not affected by adjustments made by this stage of the weighting process.

✎ Use this facility with care. If minimum or maximum adjustment takes place it is possible that the targets or proportions defined in the matrix will not be met.

1.7 Rim weighting

✎ Rim weighting can only be specified in a weighting matrix, not in an axis.

In all our examples so far, we have known the total number of people in our population who have two or more characteristics in common — one from each axis. For example, in the weighting matrix for age and sex we knew the target number of men who are aged 65 or over.

Suppose, however, we don't have these figures. We know only that our universe of 1,000 people can be described as follows:

470 men and 530 women

220 people aged 18-24, 200 people aged 25-44,
310 people aged 45-64, and 270 people aged 65 and over

200 people live in the north, 380 people live in the south,
150 people live in the east, and 270 people live in the west

We don't know, for instance, how many men there are aged 65 and over, and how many of them live in the north, so we cannot create a standard weighting matrix using region, age and sex as characteristics. Instead we use **rim** weighting which permits us to weight on these three conditions, thus:

```
wm8 region age sex;rim;200;380;150;270;
+220;200;310;270;
+470;530
```

Here, we have listed the four population totals for region, followed by the four totals for age with the two totals for sex at the end. We have also put the three sets of weights on separate lines. This has been done to make the example easier to read, but you can string them all together on the same line if you wish.

Note that when rim weighting is used, there is a maximum of 16 weighting axes per run.

Rim weighting is also useful when you know the relationship between some axes but not others - for instance, you may know how many people of each sex you have in each age group, but not the relationship between these and the region in which the respondent lives. To weight using age, sex and region as characteristics, create an axis called, say, agesex, which combines the axes age and sex as follows:

```
l agesex
n10Base
n01Male, 18-24;c=c110'1'.and.c112'1'
n01Female, 18-24;c=c110'2'.and.c112'1'
```

Your rim weighting matrix will then contain weights for age and sex combined and region:

```
wm9 agesex region;rim; ...
```

Rim weighting calculates weights using a form of regression analysis. This requires two parameters: a 'limit' which defines how close the weighting procedure must get to the targets you have given in order for the weights to be acceptable, and a number of 'iterations' which defines the number of times the weight calculations may be repeated in order to reach the cell targets.

At the end of each iteration, Quantum compares the root mean square (rms) with the product of the target sample size and the given limit. The iterations continue until the root mean square is within the limit, in which case weighting is considered successful, or until the maximum number of iterations has been reached. If, after the maximum number of iterations, the root mean square is still outside the limit, Quantum issues the message 'rim weighting failure', but continues the run with the weights that have been calculated.

☞ For details of the formula used for the root mean square, see section 2.9, 'The weighting report file' in the Quantum User's Guide Volume 4.

The default limit is 0.005 and the default number of iterations is 12. You may change these parameters by creating a rim weighting parameters file containing a line of the form:

wm=*n* **limit**=*x* **iters**=*y*

where *n* is the number of the weighting matrix concerned, *x* is the new limit (between 0.0001 and 0.05) and *y* is the new number of iterations required (between 5 and 500). For example, you may wish to reduce the limit and increase the number of iterations on a large sample to increase the accuracy of the weights.

☞ For further information about the rim weighting parameters file, see section 1.7, 'The rim weighting parameters file' in the Quantum User's Guide Volume 4.

As with ordinary weighting, rim weighting writes a summary report of the weights it applied in a file called *weightrp*. This shows the weights for each category as they were specified in the Quantum spec, and the input and projected frequencies and percents, and then the weights it calculated. If you wish, you may request a more detailed report that shows the rim weights calculated at every iteration.

This more detailed report has, in addition to the standard pages, one page per iteration showing the root mean square (rms) and limit at that iteration, plus a table showing the current weight, output and projected frequency for each weighting category.

For example:

```
Weighting matrix 1:
After 1 iteration:
rms=607.817042    limit=0.500000
      RIM           OUTPUT        PROJECTED
    WEIGHT        FREQUENCY      FREQUENCY
  ----------     ----------     ----------
    2.200000         10.000         22.000
    1.250000         16.000         20.000
    1.823529         17.000         31.000
    2.250000         12.000         27.000
    0.927721         50.662         47.000
    1.074218         49.338         53.000
```

To request this level of detail, add the option:

report=detailed

to the weight matrix entries in the rim weighting parameters file for which you require the report. (The standard report type is report=normal, but this need never be specified.)

☞ For further information on rim weighting see the Rim Weighting Theoretical Basis Paper entitled 'ON A LEAST SQUARES ADJUSTMENT OF A SAMPLED FREQUENCY TABLE WHEN THE EXPECTED MARGINAL TOTALS ARE KNOWN', by W. Edwards Deming and Frederick F. Stephan, in Volume 11, 1940 of the Annals of Mathematical Statistics.

Maximum and minimum weights with rim weighting

You may use *maxwt=* and *minwt=* with rim weighting as for ordinary jobs. Once Quantum has calculated the weights according to your rim weighting specification, it checks whether there are any that are lower than the minimum value you specified or higher than the maximum. If so, it adjusts the weights as it does for other weighting methods.

Quantum makes 100 attempts (iterations) at setting weights that fall within the given range, after which it stops and reverts to the original weighting factors calculated before the adjustments.

If the adjustments fail, Quantum produces the standard rim weighting report showing the weighting factors calculated by the rim weighting process. If the adjustments succeed, the rim weight, output frequency and output percent columns in the weighting report will contain the following information:

RIM WEIGHT	The original weight factors calculated by the rim weighting process. These are used only if the adjustments fail.
OUTPUT FREQUENCY	The final output frequencies. If the adjustments succeeded then the figures will be based on the adjusted factors, otherwise they will be based on the values reported in the RIM WEIGHT column.
OUTPUT PERCENT	The percentages for each rim element, either adjusted or unadjusted as appropriate.

In addition, the Rim Weighting Efficiency and the Maximum and Minimum Respondent Rim weights will also show adjusted figures if adjustment was successful.

✎ It is not possible to write out the adjusted rim weights as it is the cell weights that Quantum adjusts rather than the rim weights.

1.8 Using weights

Quick Reference

To use weights, type:

wm=*number*

as an option on the *a, sectbeg, flt, tab* or *n* (or equivalent) statement, where *number* is the number of the weighting matrix or axis.

To switch off weighting applied at a higher level, type:

wm=0

Using weights is easy compared to setting them up. All you have to do is tell Quantum which weighting matrix or axis to use when. Weighting is invoked by the option:

wm=*n*

on the *a, sectbeg flt, tab* or *n* line, where *n* is the number of the weighting matrix or axis to be used.

For example, to weight a table using weight matrix/axis 3 we would write:

```
tab ax01 bk01;wm=3
```

wm= on the *a* statement is operative for the whole run, whereas on a *tab* line it refers only to tables created by that statement (remember, *and* statements take their options from the previous *tab* statement). When used on an *n* statement or with elements on a *col, val, fld* or *bit* statement, *wm=* weights that element only, according to the given matrix. If a table contains row elements weighted using one matrix and column elements weighted using a second matrix, each cell will be weighted using the matrix named on the column element for that particular cell. Weighting defined for the row axis is ignored.

To turn weighting off for a particular cell, for example to produce a table containing a weighted and an unweighted base, place the option:

```
wm=0
```

on the element. To produce an unweighted table in an otherwise weighted run, add this option to the *tab* statement for that table.

If a cell is created by combining a weighted row element with an unweighted column element, the cell will be unweighted. If the cell is created by combining an unweighted row element with a weighted column element, the cell will be weighted.

Information about the weights calculated and applied is written to the weighting report file. Some information is also displayed on the screen.

☞ For further information on the name and content of this file, see section 2.9, 'The weighting report file' in the Quantum User's Guide Volume 4.

1.9 Copying weights into the data

Quick Reference

To transfer weights into the data, type:

wttran [*matrix_number*] **c(***start_col***,** *end_col***) :***dec_places*

in the edit.

Often you may wish to transfer the weights created during a run back into the data file itself, in order, for example, to give them to clients. This can be done using the statement:

wttran [*wmm_num*] **c(***m***,***n***) :***dp*

in the edit. This tells Quantum which weighting matrix to use (you may omit this parameter if there is only one matrix), which columns to store the weights in and the number of decimal places required. Remember that the decimal point takes up a column in the data, so you will need to assign at least one column more than there are digits in the largest weight. If some weights have more digits before or after the decimal point than others, do not worry. Quantum always puts the decimal point in the same column.

✎ When you use *wttran*, make sure that you include a *write* or *split* statement in the edit after the *wttran* statement to save the new data in a file. Remember that unless you specifically save your new data, all alterations or additions exist only as long as each record is being processed.

An example of *wttran* might be:

```
wttran 2 c(75,80) :2
```

which copies the respondent's weight from matrix 2 into columns 75 to 80 of the data record. The weight is copied with two decimal places and has the decimal point in c78.

2 Row and table manipulation

Row and table manipulation are two of Quantum's more advanced features. They enable you to create new rows and tables using whole tables or parts of previously created tables.

A previously created table is one whose *tab* statement appears before the *tab* statement for the current table in this run, or one which appears anywhere in a previous run. You cannot manipulate tables which have yet to be created, nor may you manipulate tables of more than two dimensions.

2.1 Row manipulation

Quick Reference

To create an element by manipulating other elements in the axis, type:

m[*element_text*]**; ex=***manip_expression*[**;***options*]

Row manipulation is the process whereby a row is created from other rows — for example, by dividing one row by another or adding two or more rows together. These facilities may also be applied to the columns of a table to create new columns from existing ones. However, if a table contains both row and column manipulation, the row manipulation will be done before the column manipulation. Throughout our explanations of these facilities we will refer to row manipulation only.

Manipulation may be done to an existing row (i.e., one produced by an *n01* etc.) using the keyword *ex=*, or a new row may be generated with an *m* statement:

m[*text*]**;ex=***expression*[**;***options*]

where *text* is the text to be printed in the table and *ex=expression* determines how the row or column is to be created. Options define more specifically how the row is to be printed. All options which are valid on an *n01* are valid with *m*, except *c=*, *inc=* and *wm=*.

☞ For further information about which options are valid on an *n01*, see section 4.8, 'Options on n, col, val, fld and bit statements' in the Quantum User's Guide Volume 2.

Expressions on M statements

Expressions on *m* statements are made up of operands connected by operators. Valid operators are:

+	addition	*min()*	minimum value
–	subtraction	*max()*	maximum value
*	multiplication	*sqrt()*	square root
/	division	*exp()*	exponentiation

The operators +, –, * and / are straightforward, but the others require more detailed explanation.

min(*ex1***,***ex2***, ...)** This returns the lowest value of the expressions within the parentheses. Here, an expression is a number, a reference to another row in the table, or any of *min(), max(), sqrt() or exp()* themselves. For example, if Row1 and Row2 are the names by which those rows are identified, the expression:

```
ex=min(Row1,Row2)
```

will compare the values in rows 1 and 2 for each column separately and print whichever is the smaller in the corresponding column in the manipulated row. Suppose we have the following:

```
Row 1   10  15   9  10
Row 2    6  20   9   1
```

then our new row will read:

```
Row 3    6  15   9   1
```

max(*ex1***,***ex2***, ...)** *max()* returns the highest value of a list of expressions. In all other respects it is the same as *min()* above. If we write:

```
max(Row1,Row6)
```

we will see whatever is the larger value in each column of rows 1 and 6.

sqrt(*expression***)** This function returns the square root of the given expression. All rules listed for *min()* also apply to *sqrt()*. If the value in column 5 of row 3 is 27 and the value in column 5 of row 1 is 16, the expression:

```
sqrt(min(Row3,Row1))
```

will yield the value 4. This is the square root of 16 which is the smaller of the values in the two rows.

exp(*ex1***,***n***)** This expression only has two items in the parentheses: the first is an arithmetic expression and the second is a whole number which is the power to which 'ex1' is raised, that is, $ex1^n$. For example:

```
exp(Row6,4)
```

raises the values in row 6 to the power of 4. If the value in this row is 15, the expression would be 15^4 which is 50625.

Operands may be:

- A positive or negative integer or real number. One might have:

  ```
  ex=10+4-2
  ```

  to create a row in which all cells contain the number 12.

- A vector. That is a list of positive or negative numbers, separated by commas and enclosed in braces, where the number of values matches the number of numeric elements in the column axis.

  If our breakdown axis has five columns and we want to put a new value into each cell we could write:

  ```
  mVector Row;ex={10.0,6.2,-8.3,15.6,-3.5}
  ```

- A numeric element of the current axis which may be referenced in any of four ways described below.

  - Text, or as many characters of that text as make the element unique. All text must be entered exactly as it appears on the element itself, and must be enclosed in single quotes. Let's use the axis *region* as an example; it has six elements:

    ```
    l region
    col 117;Base;Central London;Outer London;
    +England excl. London;Scotland;Wales
    ```

In one particular table it is cross-tabulated with *sex*, as shown here:

	Total	Male	Female
Base	1000	470	530
Central London	166	70	96
Outer London	241	116	125
England excl. London	248	112	136
Scotland	196	92	104
Wales	149	80	69

and we want to create a row showing the total number of people living in England including London. Since this is a simple axis we could use an *n01* with the filter c=c117'1/3', but if we were to use row manipulation instead we would write:

```
mEngland incl. London;ex='Central'+'Outer'+'England'
```

Notice that we have only used the first word of each row text since these are all unique — in fact, just the first letter of each text would be sufficient because they are also unique. Notice also, that the words are entered exactly as they appear in the table and in the original axis specification. If this was not so, the rows would not be recognized. If we look at the table again, we will see the result of our manipulation:

	Total	Male	Female
Base	1000	470	530
Central London	166	70	96
Outer London	241	116	125
England excl. London	248	112	136
England incl. London	*655*	*298*	*357*
Scotland	196	92	104
Wales	149	80	69

Figure 2.1 Creating rows by row manipulation

- Rows to be manipulated may be assigned identifiers using the option *id=* on the *n, col* or *val* statement. When these rows are named on an *m* statement all you have to do is use their element ID. An ID may be up to six letters or numbers long, must start with a letter and must be unique within the axis in which it occurs (r1 in ax01 is not the same as r1 in ax02). In order to be recognized on a manipulation statement, the row ID must be entered in exactly the same way as it appears on the row-creating statement. R1 on an *n01* is not the same as r1 on an *m* statement.

Let's look at the previous example again. This time we write:

```
l region
col 117;Base;Central London;%id=R1;Outer London;%id=R2;
+England excl. London;%id=R3
mEngland incl. London;ex=R1+R2+R3
col 117;Scotland='4';Wales='5'
```

The base row and the last two rows are not included in any manipulation so there is no need to give them IDs. The table produced by this axis is the same as that shown above.

- A third way of referring to a row is by its overall position in the axis. This is calculated by starting with the first element in the axis and counting down until the row in question is reached — all *n* statements, including *n03* and *n09*, all elements on *col* and *val* statements and all intermediate *m* statements count as **one** element each. The only exceptions are *n00* which is ignored and *n25* which creates three unprinted rows.

☞ For information about manipulating *n25* elements, see 'Manipulating the components of an n25' later in this chapter.

When overall row positions are written on the *m* statement, they must be preceded by a hash/pound sign (#). So, if we rewrite the axis region once again we will have:

```
l region
col 117;Base;Central London;Outer London;England excl. London
mEngland incl. London;ex=#2+#3+#4
col 117;Scotland='4';Wales='5'
```

- The fourth method of picking up rows for manipulation is to use their relative position in the axis. This is obtained by counting **backwards** from the *m* statement to the row to be manipulated. All relative positions must be preceded by the 'at' sign (@). @0 and @ both refer to the current line; that is, they refer to the *m* statement itself.

  Therefore, we could create our sum of people living in England including London by writing:

  ```
  mEngland incl. London;ex=@3+@2+@1
  ```

  where @3 is 'Central London' because it is three rows before the *m* statement, @2 is 'Outer London' which is two rows before the *m* statement and @1 is the rest of England and is the row immediately before the manipulation row.

Any of these four options is correct; just use the one which suits you best at the time. You can mix the various types in one statement if you wish.

Manipulating the components of an n25

Although an *n25* statement does not print any rows in the table, it does create three rows which are used as part of statistical calculations such as means and standard deviations. These rows are:

- The sum of x^2.
- The sum of *x*.
- The sum of *n*.

☞ For an explanation of these values in Quantum terms and further information about *n25* statements, see section 5.5, 'The n25 statement' in the Quantum User's Guide Volume 2.

To refer to any of these figures on an *m* statement you must either give the *n25* an identifier or refer to it by its absolute position in the axis. The individual elements of the *n25* can then be called up as:

- #*pos*.1 or *id*.1 for the sum of x^2.
- #*pos*.2 or *id*.2 for the sum of *x*.
- #*pos*.3 or *id*.3 for the sum of *n*.

where *pos* is the absolute position of the *n25* in the axis, and *id* is an identifier assigned to the *n25* with *id*=.

An example of row manipulation

This example shows some of the more usual tasks you might accomplish with table manipulation. The table was created by the statement:

```
tab manax bk01
```

where manax is as follows:

```
l manax
col 109;Base;Single;%id=r1;Married;%id=r2
mSingle / Married;dec=4;ex=r1 / r2
mMarried / Single;ex=#3 / #2
mSingle + Married;ex=@4 + @3
n03
n00;c=c125'1'
n01People Who Bought Bread
n01Number of Loaves Bought;inc=c(250,251)
mLoaves Bought Per Person;ex=#9 / @2
n01Loaves Bought Last Month;inc=c(132,133)
mLoaves Bought Per Person Last Month;ex=@1 / 'People'
```

	Base	Male	Female
Base	200	44	156
Single	44	6	38
Married	122	27	95
Single / Married	.3607	.2222	.4000
Married / Single	3	5	3
Single + Married	166	33	133
People who Bought Bread	190	44	146
Number of Loaves Bought	1182	263	919
Loaves Bought Per Person	6	6	6
Loaves Bought Last Month	2988	720	2268
Loaves Bought Per Person Last Month	16	16	16

Figure 2.2 Row manipulation

Applying spechar and nz to manipulated elements

Quantum does not apply *spechar*, *nz*, *nzrow* and *nzcol* to elements created using manipulation. Also, if either *nzcol* or *nzrow* is in effect and an axis contains all-zero elements, those elements are not suppressed if the axis is tabulated against one containing manipulated elements.

To have Quantum treat manipulated elements the same as other elements with regard to special characters for zero or near-zero values and suppression of all zero elements, place the keyword **manipz** on the *a*, *sectbeg*, *flt* or *tab* statement.

You may revert to the default method of ignoring manipulated elements by placing a **nomanipz** option at the point at which you wish this to happen. You may switch methods many times in a run if you wish.

Here is the same table produced with and without *manipz*. The specification used is:

```
tab q10 sex;nz
1 q10
n10Base
n01First;c=c8'0/1'
n01Second;c=c8'2/3'
n01First + Second;ex=@2+@1
n01Third (all zero);c=c8'4/5'
n01Fourth (all zero);c=c8'6/9'
n01Third + Fourth;ex=@2+@1
l sex
col 15;Base;Male;Female
```

The tables produced without *manipz* are as follows. The Third and Fourth elements created with *n01*s have been suppressed but the Third + Fourth element that is created by manipulating those elements is not.

	Base	Male	Female
Base	89	43	46
First	22	15	7
Second	67	28	39
First + Second	89	43	46
Third + Fourth	0	0	0

The same table produced with *manipz* suppresses not only the all-zero *n01*s but also the all-zero manipulation element:

	Base	Male	Female
Base	89	43	46
First	22	15	7
Second	67	28	39
First + Second	89	43	46

2.2 Manipulation on N-statements

Quick Reference

To manipulate the figures in an existing element, include the option:

ex=*manip_expression*

as part of that element's definition.

All the examples so far have used *m* statements. However, *ex=* may also be used on *n* statements to manipulate the figures in that row prior to printing. For example, the row showing the number of loaves bought per person in the table above could be specified as:

```
n01Loaves Bought Per Person;inc=c(250,251);ex=/@1
```

In most cases you will see no difference in a table between a row created with *m* and the same row created using *n01;ex=*. The difference is an invisible one to do with the efficiency of your code.

An *m* statement performs whatever calculation is specified with its *ex=*. When Quantum reads an *n01* with an *ex=*, it ignores the *ex=* at first and calculates cell counts based on the data and any *inc=* specifications. Once these calculations are finished and the basic cell counts are available, Quantum applies the *ex=* specification.

So, which method should you use? If the values that are used to create the *m* row need to appear in the table as rows in their own right, as in the example on the previous page, then an *m* is more efficient. If an *ex=* expression on an *n01* uses values that need to be calculated by that statement, and those values do not need to appear in the table, then using *n01;ex=* is more efficient. In the example, using this approach would only have been better if we had not wanted to see the row showing the number of loaves bought.

2.3 Creating averages with row manipulation

The averages created by an *n07* are simply the average of values appearing in a row or column; that is, sum of values divided by number of values. To create a row in which the average is the value in one row divided by the value in another row, you will need to write an *m* statement with the appropriate expression.

Suppose a tour operator has conducted a survey of the hotels it uses in various places in an effort to improve the service it offers to holiday-makers. As part of this survey, hotel managers are asked how many rooms and beds are available in their hotel each month and, of those, the number actually occupied each month. The tour operator wants a table summarizing hotel usage for a particular month by showing the average number of rooms and beds occupied during that month. Additionally, all averages are to be shown as percentages rather than as absolute values.

This average can be calculated by dividing the number of rooms available by the number occupied and the number of beds available by the number occupied, using row manipulation. If we wanted to break these figures down according to the regions in which the hotels are situated, we would have two axes as follows:

```
l avers
n10Total Hotels
n01Rooms Available;inc=c(15,18)
n01Rooms Used;inc=c(25,28)
mAverage % Room Occupancy;ex=@1 * 100 / @2
n01Beds Available;inc=c(35,38)
n01Beds Occupied;inc=c(45,48)
mAverage % Bed Occupancy;ex=@1 * 100 / @2
l region
col 10;Base;North;North East;Midlands;East Anglia;
+South West;South East;South;London
```

If our first hotel has 210 rooms available of which 179 were occupied, the average room occupancy is 85%, ignoring all decimal places.

2.4 Manipulating whole tables

Quick Reference

To manipulate the figures in a complete table, type:

ex *manip_expression*

underneath the *tab* statement for that table.

New tables may be generated by manipulating tables created previously in the current run or anywhere in any other run. For instance, in a countrywide survey, the data may have been collected on a regional basis, with each region having a separate directory. We may wish to create some tables which refer to the country as a whole rather than to a particular region. With table manipulation, we could create these tables separately and then add them together to create a new table for the whole country.

Table manipulation of any sort is controlled by the *ex* statement:

ex *expression*

where the expression defines the type of manipulation required.

An *ex* statement by itself means nothing: it must always follow a *tab* statement defining the basic table to be manipulated. The unmanipulated table created by the *tab* statement is not printed as part of the output. For instance:

```
tab ax01 bk01
ex *2.0
```

creates the table ax01 by bk01, stores the cell values and then multiplies them by 2.0 before writing them in the tables file.

The expression comprises operands connected by operators. Valid operators are:

+	addition	*min()*	minimum value
–	subtraction	*max()*	maximum value
*	multiplication	*sqrt()*	square root
/	division	*exp()*	exponentiation

which are exactly the same as for row manipulation, except that the expressions enclosed in the parentheses with *min(), max(), sqrt()* and *exp()* refer to whole tables rather than rows.

☞ For a full description of these operators and other components of manipulation expressions, see section 2.1, 'Row manipulation'.

Operands may be constants or vectors or they may be references to whole tables as discussed below. For example, the statements:

```
tab ax01 bk01
ex *1.45
```

generates a table by multiplying each cell in the table by 1.45, as shown here:

```
Original Table (not printed):          Manipulated Table:

             Base   Col1   Col2                    Base   Col1   Col2
Base           70     30     40        Base       101.5   43.5   58.0
Row1           25     10     15        Row1        36.3   14.5   21.8
Row2           32     12     20        Row2        46.4   17.4   29.0
Row3           13      8      5        Row3        18.8   11.6    7.2
```

Vectors may be used to replace numbers in the table created by the previous *tab* statement or to define constants by which the cells in that table are to be incremented or decremented prior to printing. When supplying raw numbers all you have to do is type in the numbers separated by commas and enclosed in braces. For example:

```
tab ax01 bk01
ex {80.0,32.0,48.0,26.0,11.0,15.0, ...}
```

This creates the table ax01 by bk01 but instead of showing the cell counts read from the data (see example above) it shows the values specified by the *ex* statement. Hence, the table base will be 80.0 instead of 70, the base for Col1 will be 32, and so on. Any cells for which values have not been given will be shown as zero.

On the other hand, when vectors define incremental values any cell for which any incremental value has not been given will retain its original value. For example, we could create a table of ax01 by bk01 with a base of 80 respondents as before, except that instead of entering the exact values for each cell we would enter the values by which the original totals are to be incremented:

```
tab ax01 bk01
ex +{10.0,2.0,8.0,1.0,1.0,0, ...}
```

As you can see, the difference between the original base (70) and the manipulated base (80.0) is 10.0, the original base for Col1 (30) must be incremented by 2.0 to reach the required figure of 32.0, and so on.

In this example we have preceded the braces with a plus sign, but any of the operators –, * and / are equally valid. Note also that long vector lists may be spread over more than one line by ending the first line at a comma and preceding the vector at the start of the second line with a ++ continuation.

Referring to tables in the current run

Quick Reference

To refer to tables in the current run, type one of:

T*quantum_table_number*

T*id_name*

T*#absolute_position*

T*@relative_position*

As with individual rows, a table may be referred to in several ways. The first is to use the ID which is assigned automatically by Quantum: the first table in the run is 1, the second is 2, and so on. These numbers are printed to the right of each table-creating statement in the compilation listing. For example, the first table would be numbered:

```
tab ax01 bk01                    000001
```

and to use it on an *ex* statement we would write:

```
tab ax01 bk01
ex t000001 * 10
```

This produces a table in which each cell is ten times its original value. When automatic IDs are used, they must be preceded by the letter T in upper or lower case, as shown in our example. The leading zeros in the table ID number are optional — we could have written *ex t1 * 10*.

Alternatively, you can make up your own identifiers using the *id=* option on the *tab* statement. This is a code of up to six numbers and/or letters starting with a letter, for example:

```
tab ax01 bk01;id=first
ex Tfirst * 10
```

Note that the automatic identifier is generated for each *tab* statement and each axis on an *and* statement, but not for *add, div, sid* and *und* statements. Also, when an *id=* appears on a *tab*

statement, it interrupts the automatic count so that the next table without a user-specified ID will carry on where the last one left off:

```
tab age sex                 000001
tab demog bk01;id=demog
tab region class            000002
```

✎ You are advised not to use IDs of the form T*n* (for example, T15, T29) as these can easily be confused with Quantum's automatic IDs.

The second method is to refer to the table's absolute position in the run, preceded by the characters T# (the T must be upper case). This is found by starting at the first *tab* statement and counting down until the required table is reached — all *tab, ex, add, sid, und* and *div* statements count as one table, while *and*s cause the count to be incremented by the number of axes they contain. For example, to refer to the third table we would say:

```
ex T#3 / 10
```

Here the cells in the new table will be created by dividing the numbers in the third table by 10. All tables mentioned in this manner must come **before** the current table - you cannot be manipulating table 12 when you are only on table 8.

Finally, tables may be called up according to their relative positions in the run, preceded by the characters T@. As with rows the relative position is calculated by counting backwards from the *ex* statement to the appropriate *tab* statement. T@ and T@0 both mean the current table; that is, the table created by the *tab* immediately before the *ex* statement.

If we had:

```
tab ax01 bk01
tab ax02 bk02
ex exp(T#1,2) / T@0)
```

we would have two tables: the first which was the table 'ax01 by bk01' and the second which is the first table squared and divided by the table 'ax02 by bk02'.

Manipulating tables from other runs

Quick Reference

To manipulate tables in a different run, type:

R*run_id* / *table_manip_expression*

or:

R*run_id*>*table_manip_expression*

In order for tables from previous runs to be manipulated, the numbers in those tables need to be saved somewhere. Quantum does this automatically whenever a run produces tables. Although you need not worry about saving tables, it is nevertheless necessary to understand a little of this mechanism in order to manipulate your tables correctly.

In a run containing no manipulation at all, Quantum saves the cell values of each table in a numbers file. You cannot read this file yourself so think of it as a list of numbers separated by spaces or commas, where the first number belongs in the first cell of the first table, the second number belongs in the second column of the first row of that table, and so on.

Now, when a run contains manipulation statements, all ordinary tables are saved in the numbers file as usual, while both manipulated and unmanipulated figures are saved in the manipulated numbers file. Again, you cannot read this file so just think of it as a list of numbers and spaces.

Tables from anywhere in previous runs may be manipulated by preceding the table specification with the letter R, a run ID of up to six characters and a slash (/). For example, to multiply the second table in a run called JAN by two, we would write:

```
tab a1 b1
ex RJAN/T#2 * 2
```

✎ To use run IDs, you must set up a run definitions file in the same place as your Quantum program. Each line in this file must contain the run ID and the location of the run it represents, separated by a space.

☞ For further information about creating this file, see section 1.4, 'The run definitions file' in the Quantum User's Guide Volume 4.

Sometimes the numbers from previous runs may themselves be the result of some manipulation, but unless you say otherwise, Quantum assumes that you will be using unmanipulated figures and will search for the named table in the ordinary numbers file.

To force it to read manipulated figures from the manipulated numbers file follow the run location name in the definitions file with a space and the word *manip*. If our run definitions file names regA as the location of the numbers file for Region A and regB as the locations of the manipulated numbers file for Region B we would write:

```
tab age sex
ex + RregA/t@ + RregB/t@
```

This might create a table showing age by sex for people interviewed in regions A, B and C (the region we are currently analyzing):

Region A (not printed):

	Base	Male	Female
Base	70	30	40
18-24	25	10	15
25-40	32	12	20
41-60	13	8	5

Region B (not printed):

	Base	Male	Female
Base	85	50	35
18-24	30	21	9
25-40	29	17	12
41-60	26	12	14

Region C (not printed):

	Base	Male	Female
Base	60	28	32
18-24	22	12	10
25-40	27	12	15
41-60	11	4	7

Regions A, B and C:

	Base	Male	Female
Base	215	108	107
18-24	77	43	34
25-40	88	41	47
41-60	50	24	26

✎ When referring to tables in other runs, take great care that you name the right table: the notation T@ meaning the current table should only be used for the other runs if the table being called up is in the same relative position in the run as the table created by the current *tab* statement. If we are on our fifth table, T@ will mean the fifth table in all other runs as well.

Manipulating more than one table

When tables other than the current one are used in an expression, Quantum compares the element texts of those tables with that of the current table. If the texts are **identical**, the elements are manipulated. If an element is present only in the current table, it appears in the table unmanipulated, whereas if it exists only in the previous tables, it is ignored. Elements which are present in both/all tables but have non-identical wording cause the manipulation to fail.

This need not prevent you from manipulating rows with different texts or in different positions in the tables because Quantum will also manipulate rows which have the same ID.

For example, in the following axes:

- The two rows named B will be dealt with together because their row texts are identical.
- Rows A and Z will be dealt with together because they both have the same identifier.
- Row C will be ignored because it only appears in the first table.
- Row X, which is present in the second table only, will appear in its original form.

```
tab ax01 bk01
tab ax02 bk01
ex +T@1
l ax01
col 156;Row A;%id=r3;Row B;Row C
l ax02
col 113;Row X;%id=r1;Row B;Row Z;%id=r3
l bk01
col 127;First;Second;Third
```

This might produce the following tables:

```
ax01 by bk01 (printed):

          First  Second  Third
Row A        10       6     13
Row B         5      17     22
Row C        18      14      3
```

```
ax02 by bk01 (not printed):

          First  Second  Third
Row X        12      20      2
Row B         4      11     19
Row Z         9      15     14
```

```
Final Table (printed):

          First  Second  Third
Row X        12      20      2
Row B         9      28     41
Row Z        19      21     27
```

2.5 Manipulating parts of tables

Quick Reference

To manipulate part of a table, type:

ex *element_ref* = *expression*

after the *tab* statement for that table. Side elements are referred to as *sid* and banner (column) elements are referred to as **b***id*. In both cases, *id* is the row number or identifier.

References to elements in other tables must start with the table reference followed by either / or >.

New tables may also be created on an element by element basis using rows and columns from tables created previously in the same or different runs. Statistical and totalling elements may not be manipulated.

When we say that new tables can be created, what we mean is that you can create a table and replace all the numbers in that table with numbers from other tables. Manipulation is not an alternative to the *tab* statement: manipulation deals only with numbers whereas a *tab* statement takes texts as well and formats them to produce row and column headings.

To create a table using elements from other tables, use the statements:

tab *rows columns*
ex *element=expression*

where the *tab* statement defines the basic table to be modified, *element* names the row or column to be created in the current table and *expression* is an expression defining the manipulation required.

Elements are entered as *sn* for side (row) elements and *bn* for breakdown (column) elements, where *n* is the absolute position of the new element in the table. For example, the first row is s1 and the first column is b1. When calculating an element's position in a table, remember that each *n* statement, other than *n00*, counts as one element.

If the elements to be manipulated have IDs you may use these instead; for example, spr1 for the side element whose ID is pr1.

The expression is made up of operators and operands. Operators and operands allowed are any of those mentioned earlier in this chapter for row and table manipulation, or references to elements in the current table or in any previously created table. Elements in expressions are entered as *sn, sid, bn or bid.* If they are not part of the current table, the element specifications must be preceded by the table reference and a / or > sign.

For example:

```
tab aa bb
ex s1=s1 + T#2/s2
```

creates the table aa by bb and then adds the figures in the second row of the second table in the run to row 1 of aa by bb. Note that the expression could also have been written as `ex s1+T#2>s2`.

This form of *ex* statement must refer either to complete tables or to side elements or to breakdown elements; a combination in one statement is not allowed. To perform manipulations using a variety of elements, write a separate *ex* statement for each type. There is no limit to the number of *ex* statements which may follow a *tab*.

The statements:

```
tab ax01 bk01
ex s3=s1 + s2
ex b1=b1 * 10
```

create the table ax01 by bk01 and then replace the counts in the third row with counts which are the sum of first and second rows. Then, the values in the first column are multiplied by 10. Let's use some numbers to clarify this:

ax01 by bk01 (not printed):

	First	Second	Third
Row A	10	4	15
Row B	8	13	9
Row C	11	7	11

Manipulated Output:

	First	Second	Third
Row A	100	4	15
Row B	80	13	9
Row C	180	17	24

2.6 Creating tables using dummy data

Because Quantum allows you to manipulate tables from previous runs, it is quite possible to create new tables without having a proper data file for the run. For example, suppose we have a monthly survey which asks a panel of respondents the same set of questions each month, and we want to produce a set of tables showing summary figures for the first three months of the year.

We could merge the three data files and rerun our Quantum program against this data, but it would be more efficient to write a short manipulation program to merge corresponding tables from each of the three runs into the summary tables we require. If we decide to use manipulation, the first thing we do is set up a run definitions file naming the runs containing the monthly information. Next, we take a copy of the program used to produce the monthly tables and after each *tab* statement we write an *ex* statement which adds together the figures from this table in each of the three previous months. For instance:

```
tab age progs
ex +Rjan/t@ + Rfeb/t@ + Rmar/t@
```

These statements produce one table of 'age by progs' which is the sum of table 1 for January, February and March. This process is repeated for each table required.

Finally, create a dummy data file to be used for this run. All that's needed is a file containing one record with a serial number and card type in the appropriate columns. If the record contains codes in any other column, you run the risk of it being accepted by the tables, and thus making your three month summary table incorrect.

When you run your job, Quantum will read in the dummy record and create each table according to the *ex* statements in your program.

3 Dealing with hierarchical data

Hierarchical data is data that can be categorized at various levels. In a survey of households we may collect information about the household in general (for example, area of residence, type of residence, social class) followed by data about each individual in that household (for example, age, sex, occupation). So we have a hierarchy with the household data at the top, and the groups of personal data underneath.

Data of this type is frequently entered using trailer cards for the information which occurs more than once; in this example, the answers given by each person in the household. This is not always the case. Very occasionally you will come across surveys in which all data has been entered as one long record. Do not worry. In this section we will discuss ways of dealing with both types of hierarchical data.

3.1 Analysis levels

Analysis levels are a relatively easy way of processing hierarchical data with or without trailer cards. Each analysis level is a field of columns, a card or set of cards containing information for a specific level in the data hierarchy. For example, if card 1 contains information about the household as a whole, and each card 2 stores data for a different person in that household, we have two levels, one for the household and one for people within the household.

You may edit and tabulate data by level by giving each level a name and applying it to the edit and tabulation statements for that level. A maximum of nine levels is allowed.

You may name levels and define the structure of the levels data either using a standard *struct* statement with some additional parameters for levels, or using a levels file. Both methods are described below.

Defining levels using a levels file

Quick Reference

To define the top level, type:

level_name **cards=***card_num1*[**r**][**,** *card_num2*[**r**]**,** ...]

Cards which must be present in every record must have the card number followed by the letter r.

To define subsidiary levels, type:

level_name [**cards=***card_numbers*] < *parent_level*

Levels and where to find the relevant data are defined in the levels file which must be created in the same place as your Quantum program file. Levels must be defined in order of priority, with the highest level first. The top level is specified as follows:

levelname **cards=***n1***,** *n2***,** ...

where *n1* and *n2* are the cards containing data for this level. If any of the cards are mandatory, you must follow the card type with the letter 'r'. Suppose our top level is the whole household, which we call hhold, and the data for this level is stored on cards 1 and 2, both of which are mandatory. We would write:

```
hhold cards=1r,2r
```

The second level is defined in much the same way, except that we also have to show that it is a sub-level of the top level. This is done by following the last card number with a less than sign (<) and the name of the parent level. If our second level refers to the individual people in the household we might write:

```
person cards=3 <hhold
```

From this we can see that data for each person in the household is on card type 3. If there is more than one person in the household, we will have a card 3 for each person.

If the data for a sub-level is on the same card as its parent level there is no need to enter the card specification. Thus, the statement:

```
trip <person
```

tells us that information about journeys made is a sub-level of each person's data but does not have a card of its own: it is all on card type 3.

You must always define the parent level before you define the sub-level. For example, before you define trip as a sub-level of person, you must define the person level.

Any level may contain more than one sub-level of the same importance in the overall hierarchy. Suppose we also have a card 5 for each pet present in the household: this is a sub-level of the top level since the pet belongs to the household as a whole rather than to an individual, so we write:

```
pet cards=5 <hhold
```

Defining the data structure in the levels file

Quick Reference

To define the data structure of records in the data file, type:

ser=*start_pos***,** *end_pos*

crd=*start_pos*[**,** *end_pos*]

max=*highest_card_number*

reclen=*num_cols*

maxsub=*max_cards_per_level*

In addition to defining levels, the levels file also describes the format of the data to be read, thus making the *struct* statement redundant when you are using analysis levels. All descriptions of data must precede the levels specifications.

The serial number field is identified with the statement:

ser=*m***,***n*

where *m* and *n* are column numbers without a preceding c. For example, if the serial number is in columns 1 to 5 of each card we would write:

```
ser=1,5
```

The position of the card type is noted in a similar manner with the statement:

crd=*n* or **crd=***m***,***n*

For example:

```
crd=6      or      crd=79,80
```

If each record contains more than nine cards, you must also enter the number of the highest card type as follows:

max=*n*

Thus, if the highest card type is 15, we would write *max=15*.

If your data file contains records which are longer than 100 columns, you will need to include the statement:

reclen=*length*

in the levels file, where *length* is the record length.

Quantum assumes that each respondent's record will contain a maximum of 4096 sub-records or cards. If any of your records contain more cards than this, you will need to extend the default by entering the statement:

maxsub=*n*

in the levels file, where *n* is the maximum number of sub-records (cards) per record. If a record is found with more than the given number of cards, the datapass terminates and messages are written to out2 and the log file.

The theoretical maximum number of sub-records per record is 2,147,483,647, but the actual limit will depend on the amount of memory space available to the datapass program.

Here is an example of a levels file:

```
ser=1,4
crd=5,6
hhold cards=1r
person cards=2r <hhold
purch cards=3 <person
```

Here we are defining records with a serial number in columns 1 to 4 and a card type in columns 5 and 6. Each record may contain data at up to three levels. The highest level is the household (hhold) which is read from card 1 which is mandatory. The second level is person on a mandatory card 2 which is a sub-level of household. The lowest level is the purchase (purch) which is a sub-level of person. Data for this level is read from an optional card 3.

Defining levels and the data structure using a struct statement

---

Quick Reference

To name levels and define the structure of a levels database, type:

struct; read=2; ser=c(*m,n***); crd=c(***m,n***);**
+lev=*level_name*[<*parent_level*][**,***card_numbers*][**; maxsub=***number*]

As an alternative to the levels file, you may define the structure of a levels data file using a *struct* statement with additional options that name the levels and define their relationships to each other.

On the *struct* statement, specify *read=2*, and use *ser=* and *crd=* to define the serial number and card type columns as you would for an ordinary data file. If records have more than nine cards you will also need to use *max=* to increase the size of the C array so that it has room to store the data for cards 10 and above.

Use **lev=** to name the levels and define their relationships:

lev = *level_name*[<*parent_level*][**,** *card_numbers*]

Where:

- *level_name* is the name of the level you are defining.
- *parent_level* is the name of the level to which the current level belongs: it must be one which has already been declared.
- *card_numbers* are the cards on which the data for *level_name* is held. If the data is on the same card(s) as the parent level you may omit the card reference. You may specify ranges of card numbers either by typing the individual numbers as a comma-separated list, or by using one of the range specifications *start/end* or *start:end*. If some cards must be present in every record, follow the card number with the letter 'r'. If all cards in a range are mandatory and you have used a / or : range specification, you may type 'r' once at the end of the range.

You will need one *lev=* option for each level.

Here are some examples:

```
struct;read=2;ser=c(1,4);crd=c5;
+lev=hhold,1r; lev=person<hhold,2r,3
```

This example declares a data file with two levels. The top level is household which is held on card 1. Every record must have a card 1. The second level is person. This is a sub-level of household and the data is held on a mandatory card 2 and an optional card 3.

```
struct;read=2;ser=c(1,4);crd=c5;
+lev=person,1r; lev=trip<person,2; lev=shop<trip
```

The data file for this example has three levels. Person is the top level and is held on a mandatory card 1. Trip is the second level and, if it is present, is held on a card 2. The third level is shop which, since it has no card specification of its own, is assumed to be held on card 2.

```
struct;read=2;ser=c(1,4);crd=c5;
+lev=doctor,1:3r; lev=patient<doctor,4r,5
```

This example has two levels. Information about the doctor is held on cards 1, 2 and 3, all of which must be present for each record. Information about the doctors' patients is held on cards 4 and 5. Card 4 is mandatory and card 5 is optional.

To define the maximum number of sub-levels a record may have, use the **maxsub=** option:

maxsub=*number*

☞ For further information about sub-levels and *maxsub=*, see 'Defining the data structure in the levels file' earlier in this chapter.

Naming levels in the edit section

Quick Reference

To start the edit in a levels job, type:

ed *level_name*

To start edits for other levels, type:

level *level_name*

To define statements to be executed at the end of a level, type:

endlevel *level_name*

Terminate the edit for each level with:

return

When analysis levels are used, all editing must be done by level. Edits start with the statement:

ed *level_name*

where *level_name* is the name of the level to be edited, and end with a *return*. Subsequent edits for other levels start with:

level *level_name*

and end with *return*. The edit section as a whole is terminated with an *end* statement:

```
ed hhold
/* Edit statements for household data
return
level person
/* Edit statements for person data
return
end
```

All statements between an *ed* or *level* and a *return* are executed after the cards for a given level have been read into the C array. So, if the current level contains two cards of the same type, all statements for that level will be executed twice, but if the cards are of different types, the statements will be executed once only.

Where it is necessary to perform a task after all data at a given level has been read in for the current record, you will need to enter the edit statements preceded by the statement:

endlevel *level_name*

and followed by a *return*. This causes all statements up until the *return* to be executed once when all data for the named level has been read in. In our example this would be when the third card 2 had been read. Note, however, that if the level in question contains data at higher levels the statements following *endlevel* will not be processed until all data at the lower levels has been read. For example:

```
ed hhold
/* ptot counts number of people in household
/* amount accumulates total family income
ptot=0; amount=0
return
level person
ptot=ptot+1
/* Personal income is in C(232,235)
amount=amount+c(232,235)
return
endlevel hhold
/* Calculate average family income
amount=amount / ptot
return
end
```

Here, the statements between *level person* and *return* are repeated for each person. The statements following *endlevel* are not processed until the data for the last person in the household has been read.

Tabulation using levels

When data is processed using analysis levels, *tab* and *l* statements must indicate the level at which the table or axis should be updated - that is, is it to be updated once per higher level or once per sub-level.

Before tables are produced, Quantum creates an intermediate file in which cells are switched on or off depending on whether a respondent satisfies the conditions of a particular axis. If the condition is satisfied, the cell is switched on; if not, it remains off.

Normally, these cells are switched on when the record fulfils the conditions specified on the axis. All cells in the intermediate table are reset to zero before a new record is read. The axis:

```
l sex
col 10;Base;Male;Female
```

has two cells — Male and Female. Whenever a Male respondent is found, the first cell is switched on, and whenever a Female respondent is found, the second cell is switched on. In the example below, a 1 means that the cell is on and a zero means that it is off:

Respondent 1	–	Male	10
Respondent 2	–	Male	10
Respondent 3	–	Female	01
Respondent 4	–	Male	10

If the record contained trailer cards, the cells would be reset between reads. Therefore, with a record containing four trailer cards, the intermediate file would read:

First trailer card	–	Male	10
Second trailer card	–	Male	10
Third trailer card	–	Female	01
Fourth trailer card	–	Male	10

which is the same as the first example. In both cases, tables using these cells would tell us that there are three men and one woman: the table is a count of people. However, this may not always be what you want.

With hierarchical data, you will often want to produce tables in which the base is a count of records rather than a count of the number of times a particular condition was true. For instance, you may want a table in which the base shows the number of households containing women, rather than the number of women in all households. For tables of this sort, you need to update the cells for each trailer card **without** resetting them between reads. This means that for each record, once a cell is switched on, it remains on for the whole of that record, regardless of the number of times the condition is fulfilled.

If we take our household of three men and a woman, the intermediate table would read:

Household 1,	Person 1	–	Male	10
	Person 2	–	Male	10
	Person 3	–	Female	11
	Person 4	–	Male	11
Household 2,	Person 1	–	Female	01

Notice the difference between Persons 3 and 4 in this example and the same people in the previous example. Here, the cells remain switched on until all data has been read, whereas in the previous example cells were reset between reads. Tables produced with these cells would show us that we have one household which contains men and women, but not how many of each there are.

Exactly when the cells in the intermediate table are updated or reset depends upon the type of table required and hence the keyword used on the *tab* or *l* statement. Keywords, described below, are:

anlev=*level_name* **celllev=***level_name* **uplev=***level_name*

Table and axis analysis level

Quick Reference

To define the level at which tables and/or axes are to be created, type:

anlev=*level_name*

on the *a*, *sectbeg*, *flt*, *tab* or *l* statement.

The level at which tables or axes are to be created is defined using the *anlev=* keyword on the *tab* and *l* statements. It means 'update the table or axis when all data for the named level has been read in'.

For example:

```
tab region class;anlev=hhold
ttlBase: Households
l region; anlev=hhold
col 136;Base;North;South;East;West
l class;anlev=hhold
col 126;Base;AB;C1;C2;DE
```

produces a table of region by class where the cells will be incremented once per household.

Why have we used *anlev=hhold* on both *l* statements? All people in a household are of the same class and live in the same region, therefore, each axis need only be updated once per household rather than once per person, which gives us *anlev=hhold* on the axes.

The reason we need *anlev=hhold* on the *tab* statement is because we want a table based on households not people.

Tab statements need not have the same analysis level as the axes which they use. For example, we may require two tables of region by class, one showing the number of households of each class in each region, and the other showing the number of people of each class in each region.

The two tables use the same axes, both of which have an analysis level of household. The first table is a count of households, so it too has an analysis level of household, but the second table is a count of people and thus needs to be updated once for every person in the household. Therefore we give it an analysis level of person.

```
tab region class;anlev=hhold
ttlBase: Households
tab region class;anlev=person
ttlBase: People
l region;anlev=hhold
    .
l class;anlev=hhold
```

However, you cannot use *anlev=* to specify a level on the *tab* statement that is higher than the level on the axes. To do that, you must use the *celllev* option on the *tab* statement as explained next.

Table creation level

Quick Reference

To create a table at a higher level than that of its axes, type:

celllev=*level_name*

on the *tab* statement.

A table may be created at a higher level than the axes by using *celllev=* on the *tab* statement. For example, to produce a table of age by sex which shows, not the number of men between 20 and 30 years of age, but the number of households having men in that age group, we might write:

```
tab age sex;anlev=person;celllev=hhold
l age;anlev=person
val c(213,214);Base;i;Under 18=0-17;18-25;26-35;36-45;
+46-55;56-65;66+
l sex;anlev=person
col 210;Base;Male;Female
```

Both axes are created on a person-by-person basis since each person's data has to be read to obtain his or her age and sex. Because both axes have an analysis level of person, the *tab* statement which cross-tabulates them must also be at that level. Normally this produces a count of people, but as we want a count of households we use *celllev* to specify the household level. This causes Quantum to increment each cell in the table only once per household, regardless of the number of people in each cell.

✎ *celllev=* is only valid on *tab* statements.

The table base is only updated for records that have data at the level defined with *celllev*; that is, the level at which the cell counts are made. For example, if level 1 is household and level 2 is child, tables created with `anlev=child;celllev=hhold` only include households with children because the axes on which the cell counts are based are at child level. If you compare this table's base cell with a table created entirely at household level you should not expect the two table bases to be the same unless every household in the data file has at least one child.

To illustrate this, consider the following basic spec:

```
tab htype region;anlev=hhold
tab age sex;anlev=child;celllev=hhold
l region;anlev=hhold
col 116;Base;North;South;East;West
l htype;anlev=hhold
col 118;Base;Detached house;Terraced house;Bungalow;Flat
l age;anlev=child
val c(213,214);Base;i;Under 5=0-4;5 to 11;12 to 15;16 to 18
l sex;anlev=child
col 210;Base;Male;Female
```

The first table shows the number of households by type of house and region. The table base is the total number of households in the data file. The second table uses data at child level and the table base only includes households with children.

However, sometimes you may you want the base in the second table to include all households regardless of whether or not they have children. To achieve this, you must use *uplev=* and *levbase*, as described in the following section.

uplev and levbase

✎ *uplev*= is an alternative to *celllev* for creating tables at a higher level than the axes in the table. However sometimes *uplev*= can give misleading results. We therefore recommend that you only use it in new Quantum specifications when you need to use *levbase* to increment the base for every record containing data at the higher level, regardless of whether it contains data at the lower level. However *uplev*= without *levbase* is retained in Quantum for backwards compatibility and is therefore documented here in full.

☞ For further information about why *uplev*= can give misleading results, see 'Why celllev is preferable to uplev' later in this chapter.

Quick Reference

To specify a lower level for an axis in a table that is created at a higher level, type:

uplev=*level_name*

on the *l* statement.

To increment the base of an *uplev*'d axis for every record containing data at the higher level, type:

levbase

as an option on the base element.

The way *uplev*= works is different from the way *celllev* works. With *celllev*, you use *anlev*= on the *l* and *tab* statements to specify the update level for the axes and you put *celllev* on the *tab* statement to specify the higher level at which you want to run the table. With *uplev*=, however, you use *anlev*= on *l* and *tab* statements to specify the level at which you want to create the table, and you put *uplev*= on the *l* statement to specify a lower update level for the axis.

For example, suppose you want to know how many households in each region contain people in specific age groups. You need to create the table at household level because it is a count of households, but age is a person level axis. So you put *anlev=hhold* on the *tab* and *l* statements to indicate that you want to create the table at household level and you put *uplev=person* on the age axis to indicate that it is at the person level:

```
tab age region;anlev=hhold
ttlBase=Households
l age;anlev=hhold;uplev=person
val c(223,224);Base;i;Under 18=0-17;18-25;26-35;
+36-45;46-55;56-65;66+
l region;anlev=hhold
col 126;Base;North;South;East;West
```

The intermediate file might look like this:

```
                        Region                          Age
Household 1   – North     1000    Person  1 – 59    0000010
                                  Person  2 – 53    0000110
                                  Person  3 – 26    0010110
Household 2   – East      0010    Person  1 – 36    0001000
```

As you can see, the cells for age are merged using an 'or' comparison to build up an overview of the household before the cells in the final table are incremented.

✎ *uplev=* is only valid on *l* statements.

By default, the base in an axis updated using *uplev=* is only incremented for records which contain data at that level. In the example we have just used, the base is incremented for every record read because every person in the household has an age. Suppose, instead, that we have an axis which counts the number of households owning different makes of car. The base for this axis only reports households that own cars — those that do not own a car are excluded.

If you want the base to be incremented for every record containing data at the higher level, regardless of whether it contains data at the lower level, place the option:

levbase

on the base element.

For example:

```
tab carmake region;anlev=hhold
l carmake;anlev=hhold;uplev=car
n10Base;levbase
col 132;Audi;BMW;Ford;Porsche;Vauxhall;Volkswagen
```

In this example, the table is created at household level and the carmake axis is at car level. Normally the carmake axis includes only households owning at least one car, but by placing *levbase* on the base element we force Quantum to update the base for every household it reads even if they do not contribute to other elements in the table. This creates a base that is all households, rather than just all households owning cars.

Why celllev is preferable to uplev

We recommend that you use *celllev* in preference to *uplev=*, except when you need to use *levbase*, which is not available for *celllev*. This is because *uplev=* sometimes gives misleading results.

For example, suppose we are working on a shopping survey that has two levels, person and shop. The data in the shop level tells us which flavors of yogurt the respondent bought in each shop, and we need to create a table of flavor by store at person level. The cell for banana flavored yogurt bought in Safeway must show the number of people buying banana flavored yogurt in Safeway at least once, **not** the total number of times banana yogurt was bought in Safeway by every person interviewed.

Using *uplev=*, we would specify the table as:

```
tab flav store;anlev=person
l flav;anlev=person;uplev=shop
    .
l store;anlev=person;uplev=shop
```

However, if we look at the table that this creates, it becomes apparent that this does not produce the figures we want. If we take just one person as follows:

Safeway – Banana, Strawberry, Mango
Sainsbury – Banana, Peach, Strawberry

our table will look like this:

```
             Base   Safeway     Sainsbury
Base            1       1            1
Banana          1       1            1
Strawberry      1       1            1
Mango           1       1            1
Peach           1       1            1
      .
```

We can see that we have one person who bought a variety of flavors in both shops, but we cannot see exactly which flavor was bought where. This is because *uplev* takes all answers and 'ors' them together **before** the axes are combined in the table.

However if use *celllev* to specify the table:

```
tab flav store;anlev=shop;celllev=person
l flav;anlev=shop
    .
l store;anlev=shop
```

For the same respondent, our table becomes:

```
              Base   Safeway     Sainsbury
Base             1         1             1
Banana           1         1             1
Strawberry       1         1             1
Mango            1         1             0
Peach            1         0             1
        .
```

We can now see that we have one person who bought banana and strawberry yogurts in both Safeway and Sainsbury, mango yogurt in Safeway and peach yogurt in Sainsbury. By giving the axes and the *tab* statement the same analysis level and updating the table with *celllev*, we force Quantum to retain the relationship between the flavor and store data for each person.

If we merely wanted a table showing the number of times each flavor was bought in each store (that is, a straightforward table of flavor by store) we would just use *anlev=shop* on the *tab* and *l* statements, and we could ignore *celllev* and *uplev* altogether.

Numerics with levels

In levels data, you need to be careful not use numeric data from a lower level at a higher level, because the figures will always be meaningless. However you can safely use numeric data from a higher level at a lower level, although if you use the same numeric field or variable at more than one level, Quantum gives the warning:

```
Same inc= at two different levels
```

For example, suppose we have two levels, household and person, and preweight figures are held in columns 108 to 115 at the household level. We need to apply the preweight to both levels. If we simply use the column spec to define the preweights at both levels, Quantum will give the warning because we are using the same numeric field at two different levels:

```
struct;read=2;ser=c(1,5);crd=c(6,7)
+lev=hhold,1
+lev=person<hhold,2

a;
wm1 ax1;pre=cx(108,115);anlev=hhold;factor;1
wm2 ax2;pre=cx(108,115);anlev=person;factor;1

l ax1;anlev=hhold
n01Total
l ax2;anlev=person
n01Total
```

As long as you have not used a lower level numeric at a higher level, and you are not preparing a study for Quanvert, you can safely ignore the warning. However, if you are preparing a study for Quanvert, you must not use a numeric at more than one level. If necessary you must set up a numeric variable at each level in the edit section. For example:

```
struct;read=2;ser=c(1,5);crd=c(6,7)
+lev=hhold,1
+lev=person<hhold,2
real hwts 1
real pwts 1

ed hhold
hwts=cx(108,115)
level person
pwts=cx(108,115)
return
end

a;
wm1 ax1;pre=hwts;anlev=hhold;factor;1
wm2 ax2;pre=pwts;anlev=person;factor;1

l ax1;anlev=hhold
n01Total
l ax2;anlev=person
n01Total
```

✎ If the preweight figures in this example were held at the person level, this solution would not be appropriate, because you must not use numeric data from a lower level at a higher level.

Statistics with analysis levels

Statistics can be unreliable when you use *celllev* or *uplev* to create a table at a higher level than its axes. For example, suppose you want to know the average amount spent by credit card per respondent. The transaction details are held at the lower, credit card, level but we want the average at the higher, respondent, level. If we simply place an *n12* element in the credit card level axis and use *celllev* to create the table at the respondent level, the *n12* will not give us what we want. Instead we need to accumulate the values for the respondent in the edit section and run the table at the respondent level.

For example:

```
struct;read=2;ser=c(1,4);crd=c(5,6)
+lev=resp,1
+lev=ccard<resp,2

real card_tot 1
real card_a 1
real card_b 1

ed resp
/* initialize at respondent (top) level
clear card_tot, card_a, card_b
level ccard
/* accumulate total value of transactions for each respondent
card_tot = card_tot + cx(212,217)
if ( c211'1' ) card_a = card_a + cx(212,217)
if ( c211'2' ) card_b = card_b + cx(212,217)
return
end

a;op=12

tab card_ave total;anlev=ccard
tab resp_ave total;anlev=resp

l card_ave;anlev=ccard
ttlAverage Transaction Size Per Credit Card
/* increment n25 by value of each transaction
/* base is number of transactions
n25;inc=cx(212,217)
n12Total;dec=2
n25;inc=cx(212,217);c=c211'1'
n12Card A;dec=2
n25;inc=cx(212,217);c=c211'2'
n12Card B;dec=2
```

```
l resp_ave;anlev=resp
ttlAverage Amount Spent Per Respondent
/* increment n25 by value of total amount spent
/* base is number of respondents
n25;inc=card_tot
n12Total;dec=2
n25;inc=card_a
n12Card A;dec=2
n25;inc=card_b
n12Card B;dec=2

l total;anlev=resp
n10Base
```

Suppose we have two respondents with credit card transactions as follows:

Respondent 1,	Card A	1.0
	Card A	2.0
	Card B	3.0
Respondent 2,	Card A	4.0
	Card B	5.0

The tables produced would be:

```
Average Transaction Size Per Credit Card

Base
Total                                3.00
Card A                               2.33
Card B                               4.00

Average Amount Spent Per Respondent

Base
Total                                7.50
Card A                               3.50
Card B                               4.00
```

Special T statistics and analysis levels

All of the special T statistics that are available in Quantum are based on the assumption that the samples being compared are independent of each other. However, in levels data, there is normally a relationship between the lower levels and the higher levels, which means that cases at the lower level are not independent of each other. For example, you would not expect the voting patterns of

the members of a household to be totally independent of each other, nor would you expect the various journeys or shopping trips made by an individual to be unrelated to each other. These relationships mean that the underlying assumptions required for the special T statistics are almost never satisfied when you run the tests on lower level data.

Process with levels

Quick Reference

To use *process* to update tables at a given level only, type:

process *level_name*

process is followed by the name of an analysis level lower than the level currently being processed in the Edit. This passes control to the tabulation section and updates only tables at the named level. The format of the statement is:

process *level_name*

This is useful when information for a specific level is on the same card as its parent level. Suppose the level Trip is a sub-level of Person, as follows:

```
person cards=1
trip <person
```

We might write our loop as:

```
ed person
do 10 t1 = 134,140,2
c(164,165)=c(t1,t1+1)
process trip
10 continue
```

However, process can produce unreliable results with hierarchical data that has more than two levels. So when there is information for more than two levels on the same card, it is preferable to recode the data so that each level is on a separate card.

☞ For a full discussion of *process*, see section 9.10, 'Going temporarily to the tab section' in the Quantum User's Guide Volume 1.

3.2 clear= on the l statement

✎ We recommend that when you write new Quantum specifications, you handle hierarchical data by using analysis levels. When you do that you do not need to use *clear=*. However *clear=* is retained in Quantum for backwards compatibility.

Quick Reference

To reset cells in Quantum's intermediate file when the data contains trailer cards which are not analyzed with levels, type:

clear=*logical expression*

on the *l* statement.

When we talked about tabulating data using levels, we said that cells are normally reset in the intermediate file when a new respondent is read in or between reads when the data contains trailer cards. If you are using analysis levels, the time at which these cells are updated can be altered using *anlev* and *uplev*. If you are not using levels, you can achieve the same effect using the option:

clear=*logical expression*

on the *l* statement. This causes the cells in the intermediate table to be reset **only** when the given logical expression is true. If you want to reset these cells for each new respondent, rather than for each record or read, enter the option as:

```
clear=firstread
```

If we take the axis Sex as we did when we explained analysis levels, you can see that both *clear=* and *anlev* produce the same results. If we write our axis as:

```
l sex;clear=firstread
col 10;Base;Male;Female
```

and take a household of three men and a woman, the intermediate table would read:

Record 1,	Person 1 – Male	10
	Person 2 – Male	10
	Person 3 – Female	11
	Person 4 – Male	11
Record 2,	Person 1 – Female	01

The first cell of Sex is switched on when a household containing a man is found. It remains on until all data for that household has been processed. If a household also contains a woman, the cell for Female is switched on and is not reset until the next record's data is read in. If a household contains both men and women, both cells are switched on.

Once the first card of the next record is read, the reserved variable *firstread* is 1 (true), so both cells will be reset to zero ready for the next household.

When axes which use *clear* are tabulated, the reserved variable *lastread* may be used in a condition on the *tab* statement so that the cells in the final table are only incremented when all cards for a respondent have been read.

A major use of this facility is in the production of Penetration or Profile tables on trailer card data. As the trailer cards are read, the intermediate table is updated to build a profile of the respondent. Cells in the printed table are incremented only when all trailer cards for a respondent have been processed.

In the example below two tables are being produced: the first shows the number of products bought, the second shows the number of products bought per respondent. Card 1 contains demographic data and card 2 (a trailer card) gives details of the items bought.

```
tab items brk
ttlBase = Number of Products Bought by Respondent
tab resps brk;c=lastread
ttlBrands of Product Bought
ttlBase = All Respondents
l brk
col 108;Base;hd=Class;AB;C1;C2;DE
l items
col 210;Base;hd=Brand;A;B;C;D
l resps;clear=firstread.eq.1
col 210;Base;hd=Brand;A;B;C;D
```

As you can see, the elements of items and resps are exactly the same — they are the brands bought. The difference is in the *l* statement which names the axis and defines its conditions. Items is just a straightforward axis whose cells in the intermediate table will be reset to zero between reads. Any tables produced with this axis will be a count of the number of times each brand was bought by respondents in each class. If the cell created by the intersection of Brand B and Class DE contained the number 52, this would mean that Brand B was bought 52 times by class DE respondents. This may mean that 52 respondents bought that brand once each or that 20 respondents bought it, with some buying it more than once.

Resps, on the other hand, has the condition `clear=firstread.eq.1` indicating that cells in the intermediate table should only be reset to zero when a new respondent is reached. This means that these cells will contain respondent profiles — that is, they will tell us whether or not a respondent bought a particular brand at any time; they will not show the number of times he or she bought each one.

The *tab* statement using this axis has the condition `c=lastread.eq.1` meaning that the table itself is not to be updated until all data for a respondent has been read. The cells in this table will tell us how many respondents in each class bought each brand. This time, if the cell created by the intersection of Brand B and Class DE contains the value 52 it will be because 52 class DE respondents bought brand B at least once.

4 Descriptive statistics

Quantum provides facilities for calculation of a set of basic statistics from the figures produced in Quantum tabulations. They include the statistics most commonly used for testing hypotheses about the values of proportions (percentages) and the locations (average values) of variables, and about differences in these between two or more subsets of the data. There are also chi-squared statistics for testing hypotheses about a single distribution or about differences between two or more distributions.

The statistical tests available are:

- One-dimensional, two-dimensional and single classification **chi-squared** tests.
- Four tests of differences between proportions (**Z-tests**).
- Two tests of differences between means (**T-tests**).
- **Friedman**'s test of differences in location between a set of related samples (sometimes known as 'Friedman's two-way analysis of variance').
- **Kolmogorov-Smirnov** test of differences between two samples.
- **McNemar**'s test of the significance of changes.
- **F Test** for testing differences between a set of means (one-way analysis of variance (ANOVA)).
- **Newman Keuls** test of differences between means.

For each statistic, Quantum also calculates and prints an associated significance level so that you can readily see the results of the tests you have performed.

✎ In addition to the statistical tests described in this chapter, Quantum provides a number of other statistical functions.

☞ For further information, see chapter 5, 'Statistical functions and totals' in the Quantum User's Guide Volume 2, and chapter 5, 'Z, T and F tests' and chapter 7, 'Special T statistics' in this volume.

4.1 Using Quantum statistics

The Quantum test statistics are divided into two groups. ***Axis-level*** statistics are specified in the axis, and are calculated for each table in which the axis appears, whether as a row or column axis. ***Table-level*** statistics are specified as an option on the *tab* statement, and are applied only to the table(s) for which they are specified.

In general more than one test may be specified for a given axis or table, and table-level statistics may be specified for tables where the axes also contain statistical elements.

In both cases, statistics are produced as part of a normal Quantum tabulation run. They are requested by means of the keyword **stat=** on the *tab* statement or in the axis.

Axis-level statistics

Quick Reference

To use an axis-level statistic, type:

stat=*statistic_name*[**,***element_text*] [**;***options*]

Axis-level statistics are requested by means of a *stat=* statement as an element in the axis:

stat=*statistic_name* [**,***element_text*] [**;***options*]

where *statistic_name* is the name of the statistic to be computed, *element_text* is an optional element text to be printed in the table against the row or column of statistical values (if no text is given, none is printed), and *options* determine how the statistics will be printed.

The names of statistics which may appear in axes are:

chi1	one-dimensional chi-squared test
friedman	Friedman's test
z1	one-sample Z-test for proportions
t1	one-sample or paired t-test

Options are one or more of *dec=, decp=, fac=* and *id=*. Axis-level statistics are printed by default with two decimal places, but this may be altered by using the *dec=* option. Similarly, the significance levels are printed with three decimal places, and this may be altered using the *decp=* option.

The *fac=* option is only relevant to the **z1** statistic.

The *decp=* option, which normally defines the number of decimal places for percentages, is used on a *stat=* element to define the number of decimal places for the significance level.

☞ For further information on the use of *dec=*, *decp=*, *fac=* and *id=*, see section 4.8, 'Options on n, col, val, fld and bit statements' in the Quantum User's Guide Volume 2.

For example:

```
stat=chi1,1-D chi-sq; dec=3; decp=4
```

calculates a one-dimensional chi-squared statistic and prints it with three decimal places and an element text of '1-D chi-sq'. Significance levels are printed with four decimal places.

Axis-level statistics only use data present in elements which appear before the *stat=* element in the axis. If the statistic is to use all the data in the axis it must come at the end of the axis. Additionally, if the axis contains a base element, the statistic is calculated using elements between the base and the statistics elements only. To include all elements, the base should therefore be the first element in the axis. For instance:

```
l brands
ttlQ5: Brands Bought
col 132;Base;Brand A;Brand B
n03
stat=chi1,1-D chi-sq
col 132;Brand C='3';Brand D='4'
```

In this example the statistics will be calculated for brands A and B only, whereas in the following example:

```
l brand2
ttlQ5: Brands Bought
col 132;Brand A;Brand B
col 133;Base;Brand C='3';Brand D='4'
n03
stat=chi1,1-D chi-sq
```

the statistics will be calculated for brands C and D only.

If appropriate, more than one *stat=* element may be placed in an axis, as long as each one is preceded by a base element, if necessary, and the requisite number of rows. To request statistics for one set of elements in an axis, and further statistics for another non-overlapping set of elements, you will need to divide the axis into ***segments***, each one beginning with a base element and ending with a *stat=* element. To request two or more tests on the same group of rows, enter the *stat=* statements, one to a line, after those rows.

Thus:

```
l allbrd
ttlQ5: Brands Bought
col 132;Base;Brand A;Brand B
n03
stat=chi1,1-D chi-sq A and B
n11Base
col 132;Brand C='3';Brand D='4'
n03
stat=chi1,1-D chi-sq C and D
```

produces two chi-squared statistics, the first for Brands A and B and the second for Brands C and D.

Statistics and the associated significance levels are printed as a row or column (depending on whether the axis is used as a row or column axis in the table), for every table in which the axis appears, at the point at which the *stat=* element is defined in the axis. The row or column text is as specified on the *stat=* statement, although if the axis is to be used as a column axis, the column headings may be defined on *g* statements. The statistic in each column (or row) is calculated from the figures in that column (or row), from the most recent base element, or the beginning of the table, through to the *stat=* element.

Table-level statistics

Quick Reference

To use table-level statistics, type:

stat=*statistic_name*[**,** *statistic_name***,** ...]

on the *tab* statement.

Table-level statistics are requested by the option:

stat=*statname*

on the *tab* statement, where *statname* is the name of the test required, and is one of:

anova	F-test (one-way analysis of variance)
chi2	two-dimensional chi-squared test
chis	single classification chi-squared test
ks	Kolmogorov-Smirnov two sample test
nk	Newman-Keuls test for the comparison of means
t2	two-sample T-test for the comparison of means
z2	two-sample Z-test for proportions

z3 Z-test for sub-sample proportions
z4 Z-test for overlapping samples

If you need more than one statistic on the same table, list their names after the *stat*= keyword, separated by commas. For example:

```
tab usage region;stat=t2,anova
```

produces a table of brand usage by region and runs a 2-sample T test and an F test on it.

The statistic(s) requested are printed at the bottom of the table or on a new page if there is insufficient space on the current page.

The *chi2, ks* and *anova* statistics produce a single statistic and significance level, preceded by text at the left margin naming the statistic. The *chis* statistic prints a + or – sign next to the percentage figure in a cell indicating whether it is significantly larger or smaller than expected.

The *z2, z3, z4, t2* and *nk* statistics all produce a triangular array of statistics and significance levels, titled at the left margin with the name of the statistic. Each row and column of the array is named by the side text defined on the corresponding axis element: the row text is printed in the left margin and the columns are spread automatically across the remainder of the page. If the texts are longer than 15 characters they will be truncated. If there are so many columns in the triangular table that each one would be allocated less than 5 columns, the statistical table is suppressed. The statistic testing the difference between any two axis elements is found at the intersection of the row and column named as those elements. (If this sounds complicated, an examination of one of the examples in the following sections should make it clear.)

In all table-level statistics, the statistics and significance levels are printed to three places of decimals. There are no options for modifying the text or layout of table-level statistics.

General notes

When using Quantum statistics, there are a few points to remember:

- Many of the statistics require that the axis (or axes, with table-level statistics) contains one or more base elements. In the case of some axis-level tests, these are only necessary to separate segments of an axis from one another. Whether or not a base element is required is defined in the notes which follow the description of each statistical test.

  Base elements may be printing or non-printing elements created using *n10* or *n11* statements, or *base* options on *n01, n15, col* or *val* statements.

- Many of the statistics require a certain number of basic count (totalizable) rows. These are rows created by *n01, n15, col* or *val* statements which obtain information directly from the data file. In an ordinary job these rows will generally be counts of people; in levels (hierarchical or trailer card) jobs they will be counts of households, people, trips made, and so on. Other elements, such as text-only elements, are ignored.

- Some tests (generally those based on table-level statistics) require that the elements to be tested are mutually exclusive. This means that respondents may be present in at most one element of the axis. Examples of mutually exclusive axes are sex, age, marital status, product preferred, and so on, where someone present in one category will not be present in any other.

  Complex axes often contain elements which are not wholly mutually exclusive: for example, one containing elements for sex and elements for age, where respondents may be present in both a sex category and an age category. Axes of this type may be used as the basis for statistics although in many cases the values for the overlapping categories should be ignored.

- Some statistical tests become unreliable when performed on tables containing cells with small numbers of respondents in them, for example, less than 10. These are generally the tests resulting in a chi-squared or Z statistic. In these cases it is best, if possible, to combine the row or column element, or both, with the logically nearest element to increase the cell sizes. Otherwise, exclude the row or column from the test altogether by specifying it with the option *ntot*. (The logically nearest element is the one whose meaning is nearest to that of the small element — for example, the '18–24' age range would be combined with the '25–34' age range rather than the '55 and over' age range.)

4.2 Summary table

The following table summarizes the statistical facilities described in this chapter and in chapter 5, 'Z, T and F tests'. The table indicates where each test is specified, what name is used following the *stat*= keyword, what requirements there are for Quantum to be able to perform the test, and whether one statistic or a triangular array of statistics is produced. You will need to refer to the appropriate section of this guide for a full description of the requirements of each test.

This table does not describe in full the statistical requirements of each test. The descriptions in the following sections provide basic information about these requirements.

Test	Name	Type	Base required	Elements required	Special requirements	Statistics produced
1-dim chi-squared	*chi1*	Axis	Yes	2+	–	1
2-dim chi-squared	*chi2*	Table	Both	2+ × 2+	–	1
McNemar's test	*mcnemar*	Axis	Yes	2	–	1
Kolmogorov-Smirnov	*ks*	Table	Both	2 cols	–	1
Friedman's test	*friedman*	Axis	Yes	2+	inc=	1
1-sample Z-test	*z1*	Axis	Yes	1	fac=	1
2-sample Z-test	*z2*	Table	Row	1 row	–	Array
Z-test on subsamples	*z3*	Table	Both	0 rows	–	Array
Z-test overlapping samples	*z4*	Table	Both	2+	axis × itself	Array
1-sample t-test	*t1*	Axis	Yes	any	n25 or fac=	1
2-sample t-test	*t2*	Table	Row	–	n12,n17	Array
1-way ANOVA	*anova*	Table	Row	–	n12,n17	1
Newman-Keuls	*nk*	Table	Row	–	n25 or fac=	Array
Single classification chi-squared	*chis*	Table	Yes	1+	–	1

4.3 Chi-squared tests

One-dimensional chi-squared test

Quick Reference

To request a one-dimensional chi-squared test, type:

stat=chi1 [**,** *element_text*] [**;***options*]

as an element in the axis.

The one-dimensional chi-squared test statistic is an axis-level statistic. You may use it to test whether the counts in an axis or a segment of an axis differ from those which would result from a uniform distribution. A uniform distribution is one where all values have the same relative frequency. Thus if an axis has four elements, and the respondents are uniformly distributed over

that axis, you would expect the number present in each element to be 25% of the base for that axis.

To request a one-dimensional chi-squared test, place a **stat=chi1** element in the axis whose distribution is to be tested at the point at which you want the statistic displayed. The first element in this axis must be a base element: other base elements may be present, and these define the beginning of additional segments in the axis. There must be at least two basic count elements in each segment on which the test is performed, that is, between each *stat=chi1* element in the axis and the most recent base element. For example:

```
l flavor
col =123;Base;hd=Low Fat;Strawberry;Raspberry;Blackcurrant;Pineapple
n03
stat=chi1, 1D chi-squared
n03
n11Base
col =123;hd=Original Flavor;Peach='5';Mango='6'
n03
stat=chi1, 1D chi-squared
```

When checking your table, bear in mind the following:

- If all cell counts in a segment are the same, the chi-squared value will be zero.
- Although the *nz* option suppresses all-zero rows in a table, these rows are still used in the calculation of the chi-squared statistic.
- The elements in the axis or in each segment must be mutually exclusive. This means that a respondent must appear in only one element of the axis or segment.
- Chi-squared tests may give misleading results when expected cell counts are small. In this case, a useful guide is that the total of the counts in the axis, and in each segment tested, should be five times the number of elements in the axis (or segment). That is, the average of the counts in the axis or segment used should be at least five.
- Although a base element must be present as the first element of the axis, or of each segment in the axis, only the basic count elements are actually used in the calculation. Take the following example:

Base	: 60
Row 1	: 25
Row 2	: 19
Chi-squared	
Row 3	: 7
Row 4	: 9

  Here, the statistic will be calculated for Rows 1 and 2 using a base of 44. Quantum will then test whether those two counts are significantly different from 22.

Let's look at an example. You have carried out a survey of purchases of washing powder throughout the country, and now wish to test whether there is a preference for certain powders in different regions. The Quantum program:

```
tab powder region
ttlQ.7 Which brand of washing powder do you usually buy?
ttlBase: All respondents
l region
col 110;Base;North;South;East;West
l powder
col 115;Base;Suds;Washo;Gleam;Sparkle
n03
stat=chi1,1-D chi-sq
n33sig. level
```

produces:

```
Q. 7 Which brand of washing powder do you usually buy?
Base: All respondents

                   Base     North     South    East    West

Base                511       145       194     129     137
Suds                109        35        26      25      23
Washo               113        27        30      26      30
Gleam               149        40        51      31      27
Sparkle             140        38        46      33      33

1-D chi-sq         9.16      3.48     11.52    1.56    1.94
sig. level        0.027     0.324     0.009   0.669   0.585
```

Figure 4.1 One-dimensional chi-squared test

✎ Notice how an *n33* statement has been used to enter text against the row of significance levels.

In this example we are testing whether respondents are equally distributed across the brands. The test shows a result significant at the 2.7% level for the Base column, indicating that there is evidence that overall the number of respondents who chose each of the four brands is not equal. Looking within the individual regions, the only region with a significant result is the South at the 1% significance level. There is no evidence that the respondents in the other regions are not uniformly distributed across the four brands.

Two-dimensional chi-squared test

Quick Reference

To request a two-dimensional chi-squared test, type:

stat=chi2

on the *tab* statement.

The two-dimensional chi-squared test statistic is a table-level statistic which produces an overall chi-squared value for the table. It may be used to test for any *association* between the axes which form the table. For example, you might use it to see whether political opinions vary according to age.

To request a two-dimensional chi-squared test, place a **stat=chi2** option on the *tab* statement. The first element in each axis must be a base element. Other base elements may be present, but are ignored. There must be at least two basic count elements in each axis.

When producing tables with this statistic, remember that:

- If there is no association — that is, the figures in each column (and, equivalently, in each row) are distributed in the same proportions — the chi-squared value will be zero. This would indicate, for example, that political opinions do not vary according to age.
- Elements in which all cells are zero are ignored by this statistic. You may suppress them with the option *nz*.
- The elements in each axis must be mutually exclusive.
- Chi-squared tests may give misleading results when cell counts are small (less than 10) or when there are both row and column bases which are small compared to the overall base.
- The statistic is calculated using the sum of totalizable (basic count) rows, the sum of totalizable columns and the sum of all totalizable cells rather than the row base, column base and table base.

You might use a two-dimensional chi-squared test when you wish to use the results of a survey of political habits to test whether there is an association between voting patterns and region.

The Quantum program:

```
tab region party;stat=chi2
ttlQ3: Which party did you vote for?
ttlBase: All voters
l region
col 110;Base;North;South;East;West
l party
col 126;Base;Labour;Conservative;Liberal/SDP
g       Base   Labour   Conserv-   Liberal/
g                          ative        SDP
p          x        x          x          x
```

produces:

```
Q3: Which party did you vote for?
Base: All voters

                Base      Labour     Conserv-     Liberal/
                                        ative          SDP

Base             605         168          229          208
North            145          43           65           37
South            194          51           73           70
East             129          35           42           52
West             137          39           49           59

CHI SQUARED VALUE = 8.233

SIGNIFICANCE LEVEL = 0.222
```

Figure 4.2 Two-dimensional chi-squared test

The results show a significance level of 22.2%. We can be 77.8% confident that there is a degree of association between region and voting preference.

A single classification chi-squared test

Quick Reference

To request a single classification chi-squared test, type:

stat=chis[**([clevel=***sig_level*] [**, row**] [**, wtform**])]

on the *a, sectbeg, flt* or *tab* statement.

The single classification chi-squared statistic tests whether a subsample proportion differs significantly from the corresponding proportion for the sample as a whole. For example, suppose we ask 40 people which of two brands they prefer, and find that 15 of them prefer the first brand. This leads us to expect that if we look at each sex individually, the number of men or women preferring the first brand would be roughly 15/40 of the total number of men or women interviewed. If the figures in our table are not in this ratio we can use the chi-squared test to check whether the difference between the subsample (for example, women preferring first brand) and the total (for example, all preferring first brand) is significant.

To run the test, place the keyword:

stat=chis[**(***options***)**]

on the *a, flt or tab* statement.

When the test is applied, certain defaults are assumed:

- Results are tested for significance at the 95% level.
- The rows of the table are taken as the responses (for example, brand preferred), and the columns as the subsamples (for example, sex=female). The + or − sign is printed to the right of the column percentage.
- The test uses unweighted data only, even if the table itself is weighted.

Options on the command line which change these are as follows. If more than one option is required, the keywords must be separated with commas.

Option	Explanation
clevel=*n*	Test for significance at the *n%* level. *n* may be 90, 95 or 99.
row	The columns of the table are the responses and the rows are the subsamples to be compared. + or − will be printed to the right of the row percentages.
wtform	use the alternative formula which takes account of weighting. Note that the unweighted formula may be used with weighted tables when you wish to ignore the weights when calculating significance.

The test is applied to all cells in the table unless:

- the row or column includes the option *nontot*, or
- there is no previous base element in either direction (that is, a missing row base, a missing column base, or both row and column bases are missing), or
- the row or column does not have the appropriate *op*= option on it. The appropriate options are op=2 if the columns are the subsamples or op=0 if the rows are the subsamples.

In addition, if the weighted formula is requested, the following condition also applies. The weighted formula uses both weighted and unweighted data, so when looking at the subsample elements in a weighted table, Quantum expects each of those elements to be preceded by a version of itself which is suppressed, unweighted and nontotalizable:

```
n15;c=condition;wm=0;nontot
n01Subsample 1;c=condition
```

The test will not be applied to elements where this suppressed element is not found. You should note that Quantum does not check that the condition on the suppressed element matches that on the corresponding subsample element.

The test is applied identically to single-coded and multicoded responses, and, although it compares absolute figures, prints the results next to the appropriate percentage figures.

Whether or not a value is significant depends on the value of chi-squared at the given confidence level, the value of chi-squared for the subsample being tested, and the size of the subsample in relation to the total. Critical values used for testing significance are:

- 2.71 at the 90% level
- 3.84 at the 95% level
- 6.63 at the 99% level

If the value of chi-squared returned by the test is greater than the value of chi-squared at the given level, and the subsample proportion is greater than the total proportion, the sample is deemed to be significantly greater than expected, and a + sign is printed next to the subsample proportion. If the value of chi-squared returned by the test is less than the value of chi-squared at the given level, and the subsample proportion is less than the total proportion, the sample is deemed to be significantly less than expected, and a – sign is printed next to the subsample proportion.

In all other cases the difference is deemed insignificant and nothing is printed.

Here is an example of a Quantum script and the table it produces:

```
tab sex hswk;stat=chis(clevel=99,row);op=01;dsp;flush
ttlQ7: Do you think that household chores are evenly
ttl    shared in your household?
foot
ttl
ttlRows are subsamples to be compared
l sex
col 110;Base;Male;Female
l hswk
col 156;Base;Yes;No;DK
```

```
                                        Absolutes/row percents
Q7: Do you think that household chores are evenly
    shared in your household?

               Base           Yes           No           DK

Base            200            61          113           26
                             30.5%         56.5%        13.0%

Male             80            39           30           11
                             48.8%+        37.5%-       13.8%
Female          120            22           83           15
                             18.3%-        69.2%+       12.5%

Rows are subsamples to be compared
```

Figure 4.3 Single classification chi-squared test

The cell for men answering yes is flagged with a + sign. This means that it is significantly greater than would be expected according to the overall proportion of people who answered yes. In statistical terms this means that:

- the value of chi-squared for that cell is greater than 6.63, and that
- 39/80 is greater than 61/200 at the 99% confidence level.

Where no + or – sign is shown, the subsample proportions are not significantly different from the proportion for the sample as a whole.

4.4 Non-parametric tests on frequencies

Kolmogorov-Smirnov test

Quick Reference

To request a Kolmogorov-Smirnov test, type:

stat=ks

on the *tab* statement.

The Kolmogorov-Smirnov statistic is a table-level statistic. It may be used to compare the cumulative frequency distributions of two samples to test whether they are from the same population. For example, you might wish to compare frequency of shopping in Safeway with frequency of shopping in Sainsbury to test whether one frequency increases more rapidly than the other.

To request a Kolmogorov-Smirnov test, include the option **stat=ks** on the *tab* statement. The table must have three columns only: the first must be the base column, and the other two columns divide the sample into the two groups to be compared. For example, in the shopping survey the table would require a base column and a column for each supermarket.

The first row of the table must be the base row, while the other rows represent some ordered classification of the respondents — numbers, numeric ranges, or measurements on some ordered scale — listed in increasing order of magnitude.

Notes for this test are:

- Both the row and column axes must contain only elements which are mutually exclusive.
- When the rows comprise numeric ranges, remember that the test is based only on the figures in the table, and therefore the more information there is in the table, the more powerful the test will be. In other words, the more categories the better — you can lose information by collapsing data too much into a few large categories. The counts in the cells of the table can be small, even zero.
- This test uses the sum of totalizable rows rather than the figures in the base row in its calculation.

As an example, let's expand on the shopping survey we mentioned just now. Suppose you wish to compare frequency of shopping between a sample of people who shop at Sainsbury and a sample who shop at Safeway, and you wish to know, not whether the average number of visits differ, but whether the distributions themselves differ. A Kolmogorov-Smirnov test is appropriate:

```
tab freq shop;stat=ks
ttlMonthly frequency of shopping at ....
ttlBase: All respondents
l freq
val 157;Base;1–3 times;4–6 times;7–9 times;10 or more times
l shop
col 167;Base;Sainsbury;Safeway
```

produces:

```
Monthly frequency of shopping at ....
Base: All respondents
                        Base        Sainsbury        Safeway
 Base                    605              304            301
 Once                     54               24             29
 Twice                    82               46             36
 3 Times                 129               65             64
 4 Times                 194               93            101
 5-7 Times                91               51             40
 8 or more times          55               24             31

KOLMGOROV - SMIRNOV VALUE = 0.120
SIGNIFICANCE LEVEL = 0.942
```

Figure 4.4 Kolmogorov-Smirnov test

The results of this test show a significance level that is close to 1. This indicates that there is little evidence to suggest a difference between the frequency distributions for the two supermarkets.

McNemar's test

Quick Reference

To request a McNemar test, type:

stat=mcnemar [**,** *element_text*] [**;***options*]

as an element in the axis.

McNemar's test is used to test for differences in a variable with just two possible values (for example, *yes/no*). It is most commonly used to test whether differences between 'before and after' measurements on the same sample indicate a real change or are simply due to chance.

To run a McNemar test, you will need a **stat=mcnemar** element in the axis. This must be preceded by exactly two basic count elements representing the changes. For instance, one might count those respondents answering *yes* before and *no* after, and the other those answering *no* before and *yes* after.

The first element in the axis must be a base element. If several McNemar tests are required in the same axis, each must follow a base element (use *n11* if you don't want to see these extra bases) and a pair of elements representing the changes.

When looking at your table, you should remember that:

- The McNemar test is not concerned with the number of respondents whose opinions do not change.
- If the two counts are equal, the statistic will have a small but non-zero value.
- In the same way as for the one-dimensional chi-squared test, the sum of the two counts should be at least 10 to avoid giving misleading results.

Here is an example. To examine whether trying out a washing powder affects respondents' willingness to buy it, you might write this Quantum program:

```
tab change ban1
ttlQ9 Likelihood of Buying Washo
ttlBase: All Trying Sample
l change
n10Base
n01Yes then No;c=c34'12'.and.c48'45'
n01No then Yes;c=c34'45'.and.c48'12'
n03
stat=mcnemar,McNemar Value
l ban1
col 12;Base;Male;Female
col 15;AB;C1;C2;DE
g                Sex                    Social Class
g      Base     Male    Female     AB      C1      C2      DE
```

This produces:

```
Q9: Likelihood of buying Washo
Base: All Trying Sample

                        Sex                   Social Class
                 Base   Male   Female     AB     C1      C2     DE

Base              400    184      216     88     96     112    104
Yes then No        96     56       40     48      8       8     32
No then Yes        99     32       56     16     40       8     24

McNemar Value    0.27   6.01     2.34  15.02  20.02    0.06   0.88
                0.600  0.014    0.126  0.000  0.000   0.803  0.350
```

Figure 4.5 McNemar test

In this example we have obtained highly significant results for respondents in social classes AB and C1. If we look at the result for respondents in social class AB, we see that 48 changed their mind negatively after trying Washo and decided that they would not buy it. Only 16 respondents in the same social group made the opposite decision. The significant result shows that for respondents in social class AB, trying Washo adversely affects their likelihood of purchasing it.

4.5 Friedman's two-way analysis of variance

Quick Reference

To request Friedman's two-way analysis of variance, type:

stat=friedman [**,** *element_text*] [**;***options*]

as an element in the axis.

Friedman's test is performed in Quantum using an axis-level statistic. It is used to test whether the location (average value) of a variable differs between a set of matched samples. Usually, the samples are a set of test scores obtained under different conditions, or given to different products, by the same set of respondents. The data can therefore be matched by comparing each respondent's score for one product or test with the same respondent's score for the other products or tests.

The test is performed by ranking each set of scores — that is, giving the value 1 to the lowest score given by each respondent, 2 to the next lowest, and so on. (Sometimes, the data is already in this form; for example, respondents may themselves have been asked to rank their preferences for a set of products.)

Friedman's tests are produced by a **stat=friedman** element in the axis. Each such element must be preceded by at least two basic count elements identifying the products, tests etc. to be compared. Each element must contain an *inc=* to calculate the sum of the ranks given to the item specified in that element (as shown in the example below).

If your data columns contain scales which are not ranked (such as scores), you must use statements in the Quantum edit to set the ranks — numbers from 1 upwards — into variables. These can then be used to define the *inc*s on the *n* statements used by the test. For example:

```
data rnk 4s
ed
if (c131'9') set rnk1'1'
if (c131'7') set rnk1'2'
```

The first element in the axis must be a base element: other base elements may be present, in which case they define the beginning of additional segments in the axis.

Notes for this test are:

- If there is no overall tendency for one product (or test or whatever) to score or be ranked more highly than another, the value of Friedman's statistic will be zero. On the other hand, the greater the disagreement between the ranks due to the different respondents, the greater this value will be.

- It makes no difference whether ranks are assigned by giving a rank of 1 to the lowest score or preference and so on upwards, or to the highest score or preference and so on downwards. Though the sums of ranks will, of course, be different, the value of Friedman's statistic in each case will be exactly the same.

- Friedman's test is extremely sensitive to any errors in assigning ranks to the elements in the axis or segment. Each respondent must have assigned a score or rank to each item in the axis or segment.

  If the ranks are read directly from columns of data, you must ensure that the columns contain one rank for each item and that the ranks the respondent has given are valid. For example, when ranking four products on a scale of 1 to 4, the respondent must have ranked each product within the range 1 to 4.

  If the data columns contain scores, your Quantum edit must convert these correctly into a valid set of ranks. Normally these will be exactly one of each of the numbers from 1 to the number of elements; thus if there are four products which have been ranked, there would be 4 elements in the axis or segment of the axis, and, for each respondent, each element would contain one of the numbers 1 through 4.

  If some products have not been ranked or invalid ranks are present in any of the data columns, Friedman's statistic will be incorrect.

- A respondent may assign the same rank to more than one product.

- In order for the significance level associated with this statistic to be correct, there should be a minimum of 10 respondents who have assigned scores or ranks to all the items in the axis or segment.

- Elements whose cells are all zero are included in the calculation of this statistic.

In the example below, respondents have expressed their preference among four washing powders by giving the one they use most a value of 1, the next a value of 2, then 3 and 4. We may write a section in the Quantum edit to check that these values result in a valid product ranking, and then construct an axis which sums the ranks given to each product, and performs Friedman's test on the results.

The resulting Quantum program is:

```
ed
r sp '1/4' o c(29,32)
c81 = xor(c29,c30,c31,c32)
if (c81 = '1/4') go to 5
write c(29,32) $product ranking incorrect$
5 continue
end
tab prdrank age
ttlProduct Preference
ttlBase: All Respondents
l prdrank
n10Base
n01Washo;c=c29'1/4';inc=c29
n01Suds;c=c30'1/4';inc=c30
n01Gleam;c=c31'1/4';inc=c31
n01Sparkle;c=c32'1/4';inc=c32
n03
stat=friedman,Friedman value;dec=2
n33Sig. level
l age
col 10;Base;18-24;25-34;35-44;45-54;55+
```

and it produces:

Product Preference
Base: All Respondents

	Base	18-24	25-34	35-44	45-54	55+
Base	650	96	194	91	126	98
Washo	2072	329	672	311	432	328
Suds	1649	261	506	250	342	290
Gleam	862	137	284	129	180	132
Sparkle	1467	233	478	220	306	230
Friedman value	750.64	118.88	234.62	113.65	155.85	134.13
Sig. level	0.000	0.000	0.000	0.000	0.000	0.000

Figure 4.6 Friedman's test

All the results in this table are highly significant. We can have more than 99.9% confidence that there are significant differences in the sample as a whole as well as in all of the individual age groups. This means that there is strong evidence that there are differences between the ranks that the respondents have given each of the brands.

4.6 Formulae

The formulae for the statistical tests in this chapter are shown below. The following conventions have been used in these formulae:

- In the formulae for axis-level test statistics, the formula is applied separately to the counts in each column or row, according to whether the axis containing the *stat=* option is the row or column axis:

k	Represents the number of basic count elements in the axis or segment.
n_i	Represents the (weighted) count in the *i*th cell of a row or column representing that axis.
N	Represents the (weighted) base of that row or column.
U	Represents the unweighted base of that row or column.

- In formulae for table-level test statistics:

r	Represents the number of basic count rows from which the statistic is calculated.
c	Represents the number of basic count columns from which the statistic is calculated.
n_{ij}	Represents the (weighted) count in row *i*, column *j*.
N, N_i, N_j	Represent the (weighted) bases of the table overall, column *i* and row *j* respectively.

- A dot suffix indicates summation over the replaced index; so, for example, the formula for a column total is:

$$n_{\cdot j} = \sum_{i=1}^{r} n_{ij}$$

The one-dimensional chi-squared test

If there are *k* elements in the axis, then:

$$X^2 = \sum_{i=1}^{k} \frac{(n_i - \bar{n})^2}{\bar{n}}$$

is tested against the χ^2 distribution with (k - 1) degrees of freedom.

where:

$$\bar{n} = \frac{n.}{k}$$

is the expected number in each cell.

The two-dimensional chi-squared test

$$X^2 = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{(n_{ij} - e_{ij})^2}{e_{ij}}$$

is tested against the χ^2 distribution with ($r - 1$) ($c - 1$) degrees of freedom.

where:

$$e_{ij} = \frac{n_i.n._j}{n..}$$

is the expected number in each cell.

Single classification chi-squared test

Where:

O	is the observed value of the subsample.
e	is the expected value of the subsample.
a	is the number of respondents in the cell being tested (that is, the subsample).
b	is the (weighted) number of respondents in the element.
n	is the number of respondents giving a particular answer (that is, the sample).
N	is the (weighted) total number of respondents in the table.

$$\chi^2 = \sum \frac{(O_i - e_i)^2}{e_i}$$

$$\chi^2 = \frac{\left(a - \frac{b*n}{N}\right)^2}{\frac{b}{N} * n} + \frac{\left((n-a) - \frac{(N-b)}{N}*n\right)^2}{\frac{N-b}{N} * n}$$

Kolmogorov-Smirnov test

$$X_{KS}^2 = 4D^2 \frac{N_1 N_2}{N_1 + N_2}$$

is tested against the χ^2 distribution with 2 degrees of freedom.

where:

$$D = \max_{i=1}^{r} \left[\frac{\sum_{k=1}^{i} n_{k2}}{N_2} - \frac{\sum_{k=1}^{i} n_{k1}}{N_1} \right]$$

is the maximum difference found between the two cumulative distributions.

McNemar's test

$$X_{MN}^2 = \frac{(|n_1 - n_2| - 1)^2}{n_1 + n_2}$$

is tested against the χ^2 distribution with 1 degree of freedom.

Friedman's test

$$X_r^2 = \frac{12}{Nk(k+1)} \sum_{i=1}^{k} R_i^2 - 3N(k+1)$$

where R_i is the sum-of-ranks in cell i of the axis, is tested against the χ^2 distribution with k - 1 degrees of freedom.

5 Z, T and F tests

5.1 Z - tests

Quantum provides four types of Z-test for comparing proportions with specified values and with other proportions.

One-sample Z-test on proportions

Quick Reference

To request a one-sample Z-test, type:

stat=z1[**,** *element_text*] **;fac=***factor* [**;***options*]

as an element in the axis. *factor* is a value between 1 and 100 and is the proportion against which values are to be compared.

This is an axis-level statistic which is used to test whether the percentage of respondents in a particular element differs from a given value. For example, if you wanted to see whether more than 40% of yogurt buyers buy low-fat brands only, you would compare each sample percentage with the number 40.

To request a one-sample Z-test, place a **stat=z1;fac=***n* element in the axis. The *fac*= option on the *stat* statement specifies the value, expressed as a percentage, with which the percentages in the preceding element are to be compared. *n* may be a whole number or a decimal number with up to six decimal places. Each *stat*= element must be preceded by a base element and a single element of basic counts.

The Z-tests may give misleading results when the bases from which proportions are calculated are small. In this case, the base element should contain at least 10 respondents for the test to be valid.

The Z-value will be zero if the difference is zero, otherwise it will be negative if the calculated percentage is smaller than the specified value, and positive if it is greater.

The example which follows defines an axis for a survey investigating purchases of dairy products. We are checking whether 50% of respondents buy low-fat brands of yogurt, and then testing this hypothesis among different age-related sub-samples.

The Quantum specification is:

```
tab brand age
ttlQ9: Type of Yogurt Purchased
ttlBase: All Yogurt Buyers
l tried
col 121;Base;Buys Low-Fat
stat=z1,Z Value for 50%;fac=50
l age
col 108;Base;18–24;25–34;35–44;45–54;55+
```

The table it produces is:

```
Q9: Type of Yogurt Purchased
Base: All Yogurt buyers
                    Base    18-24    25-34    35-44    45-54      55+
Base                 605       96      194       91      126       98
Buys Low-Fat         334       50      108       51       73       52
Z value for 50%     2.56     0.41     1.58     1.15     1.78     0.61
                   0.010    0.683    0.114    0.249    0.075    0.544
```

Figure 5.1 One-sample Z-test on proportions

If we look at the results in the Base column of this example, we find a significant result at the 99% confidence level. This would suggest that we should reject the null hypothesis that 50% of yogurt buyers buy low fat yogurt. However, none of the age groups differ significantly from the 50% figure at the 95% confidence level.

Two-sample Z-test on proportions

Quick Reference

To request a two-sample Z-test, type:

stat=z2

on the *tab* statement.

The two-sample Z-test is a table-level statistic. It is used to test differences between column percentages in a single row of a table. For example, we may wish to test whether younger women are as likely as older women to have full-time jobs — that is, to compare the proportions of women with full-time jobs across groups of women in different age-groups.

This test is produced by the option **stat=z2** on the *tab* statement. The table must consist of a base row and one row of basic counts only. The test calculates a Z-value comparing each column percentage with each of the other column percentages, and produces a triangular table showing all the Z-values and their associated significance levels. The triangular table is labeled with the text 'Z TEST – TYPE 2'.

Points to remember are:

- The row of basic counts defines an attribute which respondents in that row have, for example, the attribute of having a full-time job.

- The percentages (proportions) which are compared are always calculated for the test by dividing the count in each cell of the row to be tested by the corresponding cell in the base row. It is not necessary for the column percentages to be printed using the option *op=2* (though you might find it confusing to use a two-sample Z-test and print the row or total percents instead).

- The columns of the table should define the different groups of people in such a way that each group is mutually exclusive — for example, age groups or sex. If the column axis defines more than one set of mutually exclusive elements the test will still be printed, but the comparisons between elements which are not mutually exclusive will be meaningless and should be ignored. For example, if the column axis contains both sex and age breakdowns, the comparison between, say, 'Female' and 'Age 18–25' must be ignored since some respondents may be women and aged 18–25.

- The Z-tests may give misleading results when the bases from which proportions are calculated are small. In this case, tests involving a column whose base is less than 10 should be treated as approximate only. Such columns should preferably be combined with the nearest logical equivalent.

- The calculation for Z subtracts the first proportion from the second, rather than the more usual method of subtracting the second proportion from the first.

The Quantum program below compares the proportions of women in full-time employment between different age-groups:

```
tab ftjob age;stat=z2
ttlJob Status
ttlBase: All Women
l ftjob
col 145;Base;Full-Time
l age
col 108;Base;18–24;25–34;35–44;45–54;55+
```

produces:

```
Job Status
Base: All Women
                   Base   18-24   25-34   35-44   45-54   55+
Base                605      96     194      91     126    98
Full-Time           297      29     107      66      75    20
Z TEST - TYPE 2
                       18-24       25-34       35-44     45-54
          25-34        2.388
                       0.017

          35-44        3.800       2.273
                       0.000       0.023

          45-54        2.687       0.586      -1.615
                       0.007       0.558       0.106

             55+      -0.776      -2.877      -4.100    -3.145
                       0.438       0.004       0.000     0.002
```

Figure 5.2 Two-sample Z test on proportions

This example shows us that there is no significant difference between the 18-24 and 55+ age groups in the proportion of respondents in full time employment. However, these two age groups differ significantly (a 5% or higher significance level) from each of the other age groups. There is an additional difference between the proportion in full time employment between the 25-34 and 35-44 age groups.

Z-Test on sub-sample proportions

Quick Reference

To request a Z-test on subsample proportions, type:

stat=z3

on the *tab* statement.

The Z-test on sub-sample proportions is a table-level statistic. It is used to test differences between all row percentages in a single row of a table. For example, we may wish to test whether we have the same proportion of respondents in each age group.

This test requires a **stat=z3** option on the *tab* statement. The table must consist of a base row only, and the first element of the column axis must be a base element. The test calculates a Z-value comparing each row percentage with each of the other row percentages, and produces a triangular table showing all the Z-values and their associated significance levels. The table is labeled with the text 'Z TEST – TYPE 3'.

When using this test you should bear in mind that:

- The percentages (proportions) which are compared are always calculated by dividing the count in the base column into the count in each other element of the row. It is not necessary for the row percentages to be printed using the option *op=0*.

- The columns of the table should define groups of respondents in such a way that the groups are mutually exclusive — for example, age groups or sex. If the column axis defines more than one set of mutually exclusive elements the test will still be printed, but the comparisons between elements which are not mutually exclusive will be meaningless and should be ignored. For example, if the column axis contains both sex and age breakdowns, the comparison between, say, 'Female' and 'Age 18–25' must be ignored since some respondents may be women and aged 18–25.

- The Z-tests may give misleading results when the base from which proportions are calculated is small. In this test the base should be at least 20.

- The calculation for Z subtracts the first proportion from the second, rather than the more usual method of subtracting the second proportion from the first.

As an example, the Quantum program below compares the proportions of respondents in different age-groups:

```
tab justbase age2;stat=z3
ttlAge group distribution
ttlBase: All respondents
l justbase
n10Base
l age2
col 120;Base;18–24;25–34;35–44;45+
```

The table produced is:

```
Age group distribution
Base: All respondents
                     Base     18-24    25-34    35-44     45+
Base                  400        96      104      104      96
Z TEST - TYPE 3
                              18-24        25-34         35-44
              25-34           0.556
                              0.571

              35-44           0.556        0.000
                              0.571        1.000

                45+           0.000       -0.556        -0.566
                              1.000        0.571         0.571
```

Figure 5.3 Z-test on sub-sample proportions

The results in this example show that there is no evidence to suggest that there is a difference in the proportion of respondents who fall into each of the four age groups.

Z-Test on overlapping samples

Quick Reference

To request a Z-test on overlapping samples, type:

stat=z4

on the *tab* statement.

The Z-test on proportions in overlapping samples is a table-level statistic. It is used to test differences between row percentages in a base row for a column axis with overlapping categories. For example, in a wine-tasting survey we may wish to test whether the proportion of respondents trying sweet red wine is the same as the proportion trying dry white wine.

This test requires a **stat=z4** option on the *tab* statement. The table must consist of an axis tabbed against itself, and the axis must allow multicoding. The first element of the axis must be a base element, and there must be at least two basic count elements in the axis. The test calculates a Z-value comparing each row percentage with each of the other row percentages in the base row, and produces a triangular table showing all the Z-values and their associated significance levels. This table is labeled with the text 'Z TEST – TYPE 4'.

Things you should remember about this test are:

- The percentages (proportions) which are compared are always calculated by dividing the count in each element of the base row by the overall base. It is not necessary for the row percentages to be printed using the option *op=0*.
- Although the test is only comparing proportions in the base row, it is necessary to have the axis tabbed against itself because Quantum needs to know the extent of the overlap between the different elements of the axis.
- The Z-tests may give misleading results when the base from which proportions are calculated is small. In this test the base should be at least 20.
- The calculation for Z subtracts the first proportion from the second, rather than the more usual method of subtracting the second proportion from the first.

As an example of a type-4 Z-test, suppose you have asked a multiple-response question about brand usage, and you wish to see whether different proportions of respondents have tried the products. Because the groups of respondents who have tried different products may not be mutually exclusive, we cannot use the type-3 Z-test.

The Quantum program is:

```
tab brand brand;stat=z4
ttlQ7: Which of these brands have you ever tried?
ttlBase: All Respondents
l brand
col 123;Base;Washo;Suds;Gleam;Sparkle
```

The table produced is:

```
Q7: Which of these brands have you ever tried?
Base: All Respondents
                 Base     Washo     Suds     Gleam     Sparkle
Base              427       334       92        66          78
Washo             334       334       50        36          40
Suds               92        50       92        18           9
Gleam              66        36       18        66          12
Sparkle            78        40        9        12          78
Z TEST - TYPE 4
                           Washo         Suds           Gleam
            Suds         -17.610
                           0.000

           Gleam         -21.201       -2.369
                           0.000        0.018

         Sparkle         -19.160       -1.137          1.097
                           0.000        0.255          0.273
```

Figure 5.4 Z-test on overlapping samples

The results of this example show us that:

- The proportion of respondents who have tried Washo is highly significantly different from the proportions who have tried any of the other three brands.

- There is a difference between the proportions who have tried Gleam and Suds, with a significance level of 1.8%. So we can be 98.2% confident that there is a difference between these two proportions.

- We can have only low levels of confidence in the difference between the proportions for those who tried Sparkle compared to Suds and Gleam.

5.2 T-tests and F-tests

In this section we describe a set of tests which may be used to investigate whether means differ significantly from each other or from specified values. The statistics used are the T and F statistics, two of the so-called 'classical' test statistics.

One-sample and paired T-test

Quick Reference

To request a one-sample or paired T-test, type:

stat=t1[**,** *element_text*] [**;***options*]

as an element in the axis.

The one-sample T statistic is an axis-level statistic. It may be used to test whether the mean of a numeric variable or factor (*fac=*) is significantly different from zero or some other specified value. It may also be used to test for differences between means measured on matched samples (the paired T-test) — for example, between the means of two variables both obtained from the same sample of respondents (see the Notes below).

For example, you may wish to test whether respondents spent the same length of time per day, on average, watching broadcast television before and after purchasing a video recorder. To request a one-sample or paired T-test you should include **stat=t1** element in the axis at the point at which you want the statistic displayed. To define the variable or factor to be tested and perform the necessary statistical summations, you will need either a *fac=* option on each basic count element in the axis, or an *n25* element in the axis with the *inc=* option.

Notes for these tests are:

- The value of T will be zero if there is no difference in the data, otherwise the sign of T will reflect the sign (direction) of the difference.

- It is not necessary to use *n12* (mean), *n17* (standard deviation) or *n19* (standard error) elements in the axis as these are automatically calculated by the *stat=t1* element. However, you will probably wish to print at least the mean using *n12* so that you can see the values which are being tested by the T statistics.

- In a weighted run, the compiler inserts an unweighted *n15* with the option *nontot* in the axis so that the T-test can be calculated using unweighted figures.

- The simplest use of the one-sample T-test is when testing whether the mean of a variable already coded in columns of the data is zero. In this case you need only specify the required columns on the *inc=* option of the *n25* statement. For example:

```
n25;inc=c(120,122);c=c119'1'
stat=t1,One-Sample T-test
```

- There may be occasions when you want to use a one-sample T-test on values which are not the same as those in the data. You may create these values using *fac=* on *n01* or *col* elements.

☞ For more details about *fac=*, see chapter 5, 'Statistical functions and totals' in the Quantum User's Guide Volume 2.

- If you wish to test whether a mean may be different from some non-zero value, you should subtract that value from each data value. In other words, to test whether the mean number of visits to a supermarket is equal to 2, you actually test whether the mean of (number of visits to supermarket – 2) is equal to 0. For example:

```
n25;inc=c(120,122)-2;c=c119'1'
stat=t1,One-Sample T-test
```

- If you wish to make a paired test between two data values, you should test whether the difference between them is zero. For example, to make a test of the difference between the data values in columns 120–122 and in columns 123–125 you would write:

```
n25;inc=c(123,125)-c(120,122);c=c119'1'
stat=t1,Paired T-test
```

If the calculation of the values to be used by the T-test is more complicated than this, you may need to write an edit to calculate the values. A simple example which has the same effect as that shown above is:

```
/*Named variable to store mean difference
int mdiff 1s
ed
mdiff = c(123,125) – c(120,122)
end
         .
         .
n25;inc=mdiff;c=c119'1'
stat=t1,Paired T-test
```

- If the axis being tested contains *fac=* and *inc=*, Quantum scans backwards through the axis from the *stat=t1* element and uses whichever of the two it finds first; that is, whichever of *fac=* or *inc=* occurs closest to, but still before, the statistical element.

Here is an example of a one-sample T-test. Respondents have rated a particular brand of washing powder on a scale of 1 (Excellent) to 5 (Very poor) and you wish to test whether the rating was, on average, satisfactory. The Quantum program:

```
tab rating age
ttlQ4Rating for Washo Soap Powder
ttlBase: All Respondents
l rating
col 45;Base;Excellent;%fac=2–1;Very Good;Satisfactory;Poor;Very poor
n03
n12Mean Rating;dec=3
n19Std. Error;dec=3
stat=t1,T-Values
l age
col 9;Base;18–30;31–44;45–54;55+
```

produces:

```
Q4 Rating for Washo Soap Powder
Base: All Respondents
                   Base     18-30     31-44     45-54       55+
Base                340        65        93        76        70
Excellent            95        21        20        28        16
Very Good            21         4         5         7         5
Satisfactory         70        15        21        15        19
Poor                 69        14        21        17        17
Very poor            49        11        21         9        13
Mean Rating       0.145     0.154     0.129     0.368    -0.086
Std. Error        0.085     0.186     0.156     0.168     0.169
T-Values           1.71      0.83      0.83      2.19     -0.51
                  0.087     0.409     0.408     0.029     0.611
```

Figure 5.5 One-sample T-test on means

The results of this example show us that, at the 90% confidence level, we have some evidence that the overall mean rating differs from zero (the exact significance level is 8.7%). The most highly significant result is among the 45-54 age group, in which the mean score differs from zero at the significance level of 2.9%.

An example of a paired T-test follows. For a comparison of the differences between means of two ratings, both by the same respondents, we use a paired T-test:

```
tab ratdif age
ttlComparison of Ratings for Suds
ttlBase: All Respondents
l rating
col 46;Base;hd=Rating Having Seen Advertising;
+Excellent;Very Good;Satisfactory;Poor;Very poor
n03
col 56;Base;hd=Rating Having Tried Product;
+Excellent;Very Good;Satisfactory;Poor;Very poor
n03
n25;inc=c46–c56
n12Mean Difference;dec=3
n19Std. Error;dec=3
stat=t1,T-Values;dec=3
l age
col 9;Base;18–30;31–44;45–54;55+
```

produces:

```
Comparison of Ratings for Suds
Base: All Respondents
                    Base     18-30     31-44     45-54       55+
Base                 340        65        93        76        70
Rating Having Seen Advertising
Excellent             95        21        20        28        16
Very Good             21         4         5         7         5
Satisfactory          70        15        21        15        19
Poor                  69        14        21        17        17
Very poor             49        11        21         9        13
Rating Having Tried Product
Excellent             85        20        24        24        17
Very Good             55        11        15        12        17
Satisfactory          65        10        23        19        13
Poor                  68        17        19        16        16
Very poor             31         7        12         5         7
Mean Difference    0.168     0.154     0.086     0.079    -0.386
Std. Error         0.102     0.221     0.181     0.208     0.218
T-Values           1.641     0.697     0.474     0.380     1.773
                   0.101     0.486     0.635     0.704     0.076
```

Figure 5.6 Paired T-test on means

In this example, the highest significant result is in the 55+ age group. We can have confidence at the 92.4% level that for this age group the mean product ratings differ after trying the product.

Two-sample T-test

Quick Reference

To request a two-sample T-test, type:

stat=t2

on the *tab* statement.

The two-sample T statistic is a table-level statistic. It may be used to test whether the means of a numeric variable or factor are the same in two separate samples or sub-samples, or to make a number of such comparisons, pair-wise, between more than two samples. For example, you might wish to compare the length of time per day, on average, spent watching broadcast television by owners of video recorders, with the same figure for non-owners.

This test is produced by a **stat=t2** option on the *tab* statement. The column axis defines the groups to be compared. These must be mutually exclusive. The row axis must include a base element, a mean *(n12)* and a standard deviation *(n17)* — these require *fac=* options on the axis elements or an *n25* element with the *inc=* option.

☞ For more information about these elements, see chapter 5, 'Statistical functions and totals' in the Quantum User's Guide Volume 2.

Notes for this test are:

- This statistic calculates T values using rows of means and standard deviations. Each mean in the *n12* row is compared against every other mean value in that row. A triangular matrix of T values and significance levels is produced with values for each pair of means. It is labeled with the text 'T TEST – TYPE 2'.

- The column axis must define groups of respondents which are mutually exclusive — for example, age groups or sex. If there is more than one set of mutually exclusive elements in the axis the test will still be printed, but the comparisons between elements which are not mutually exclusive will be meaningless and should be ignored. For example, if the column axis contains both sex and age breakdowns, the comparison between, say, 'Female' and 'Age 18–25' should be ignored since some respondents may be women and aged 18–25.

- The value of T will be zero if there is no difference in the data, otherwise the sign of T will reflect the sign (direction) of the difference.

- The calculation for T subtracts the first mean from the second rather than usual method of subtracting the second mean from the first.

- Elements whose cells are all zero are excluded from this test. You may suppress them with the *nz* option if you wish.

- This test uses the sum of totalizable rows and the input to the means and standard deviation in its calculations.

- If the axis being tested contains *fac=* and *inc=*, Quantum scans backwards through the axis from the *stat=t1* element and uses whichever of the two it finds first; that is, whichever of *fac=* or *inc=* occurs closest to, but still before, the statistical element.

As an example, take the Quantum program:

```
tab hours vcr;stat=t2
ttlQ15 Hours per week spent watching TV
ttlBase: All Respondents
l hours
col 156;Base;Under 5 hours;%fac=1+1;5–6 hours;
+7–10 hours;11–15 hours;16+ hours
n03
n12Mean;dec=3 n17Std. Deviation;dec=3
l vcr
col 155;Base;Has no video;Owns a video
g    Base      Does not own a      Owns a video
g              video recorder         recorder
p                         x                x
```

This produces:

```
Q15 Hours per week spent watching TV
Base: All Respondents
                   Base      Does not own a        Owns a video
                             video recorder            recorder
Base                305                181                124
Under 5 hours        45                 24                 21
5-6 hours            93                 50                 43
7-10 hours           62                 40                 22
11-15 hours          51                 31                 20
16+ hours            54                 36                 18

Mean              2.921              3.028              2.766
Std. Deviation    1.330              1.335              1.314
T TEST - TYPE 2
                           Owns a video
Has no video               -1.691
                            0.091
```

Figure 5.7 Two-sample T-test on means

This example shows a significant result at the 9.1% significance level.

Notice that, in this example, the column headings at the top of the main table are different from those in the statistical table. Those at the top of the main table are defined by the *g* statements in the axis, whereas those in the statistical table are taken from the *col* statement. The reason for this is that the full element text, as shown on the *g* statements is too long to fit into the 15 characters allocated to the statistical columns.

☞ For general information on the size and layout of statistical output, see 'Axis-level statistics' in chapter 4, 'Descriptive statistics'.

5.3 F values and T values

Quick Reference

To calculate and print an F value and a triangle of T values for a group of columns, type:

nft

as an element in the axis.

The **nft** element creates an F value and a triangle of T values for groups of columns. A group of columns starts at a base or *n23* statement and continues until another base or *n23* is read, or until the end of the axis, whichever is sooner. Columns that are non-totalizable (such as, *n04, n05, n12)* and columns that are not printed are ignored. Quantum also ignores any groups of columns that contain fewer than two elements that are included in the calculation.

The *nft* element is meaningful only in row axes and is therefore ignored in column or higher dimension axes. It is specified simply as *nft* with no row text or options.

With ordinary formatted output, the F value for a group is printed under the middle of that group. With PostScript output, it is printed under the right-most column in the group. The probability, expressed as a percentage, is printed underneath the F value.

The triangle of T values is lined up with the values in the columns to which they refer. The probability for each T value, expressed as a percentage, is printed underneath the corresponding T value. Asterisks are printed down the leading diagonal of T values.

Here's a simple spec and the table it produces. The spec is:

```
tab rating age
l rating
col 45;Base;Excellent;%fac=2-1;Very Good;Satisfactory;Poor;Very poor
nft
l age
col 9;Base;18-30;31-44;45-54;55+
g     Base     18-30    31-44    45-54      55+
g     ----     -----    -----    -----     ----
p        x         x        x        x        x
```

and the table it produces is:

	Base	18-30	31-44	45-54	55+
Base	340	65	93	76	70
Excellent	95	21	20	28	16
Very Good	21	4	5	7	5
Satisfactory	70	15	21	15	19
Poor	69	14	21	17	17
Very poor	49	11	21	9	13
F stat			2.40		
F prob			6.55		
T stat		*	-1.47	0.85	-0.95
T prob			14.03	39.30	34.06
T stat			*	2.50	0.52
T prob				1.26	60.51
T stat				*	-1.92
T prob					5.68
T stat					*
T prob					

Figure 5.8 F and T values produced with nft

5.4 F-test – one-way analysis of variance

Quick Reference

To request a one-way analysis of variance, type:

stat=anova

on the *tab* statement.

An F-test or analysis of variance uses a table-level statistic to investigate whether a set of means, calculated from independent samples, differ significantly from one another. It is used in exactly the same way as the two-sample T-test, but instead of independently making pair-wise comparisons between the means, it makes a single overall comparison of them all.

Being a table-level statistic, this test requires a **stat=anova** option on the *tab* statement. The column axis defines the groups to be compared, which must be mutually exclusive. The row axis must include a base element, a mean (*n12*) and a standard deviation (*n17*) — these require *fac=* options on the axis elements or an *n25* element with the *inc=* option.

☞ For information on these elements, see chapter 5, 'Statistical functions and totals' in the Quantum User's Guide Volume 2.

When inspecting the results, bear in mind the following:

- The value of F will be near to one if there is no significant difference to be found between the means, while high values indicate different means.
- The F-test is invalid if the column axis defines groups which are not mutually exclusive.
- Elements whose cells are all zeros are included in this test.
- The calculation uses the sum of totalizable rows and the input to the mean and standard deviation rather than the base and the mean and standard deviation values themselves.
- If the axis being tested contains *fac=* and *inc=*, Quantum scans backwards through the axis from the *stat=t1* element and uses whichever of the two it finds first; that is, whichever of *fac=* or *inc=* occurs closest to, but still before, the statistical element.

As an example, we may use the F-test to examine more carefully the results of the previous example, used to illustrate the two-sample T-test. The Quantum spec. is the same as that used in the previous example, except that the *tab* statement becomes:

```
tab hours vcr;stat=anova,t2
```

The table which this produces is:

```
Q15 Rating for Brand Bought Most Recently
Base: All Respondents
                    Base      Does not own a        Owns a video
                              video recorder            recorder
Base                 305                181                 124
Under 5 hours         45                 24                  21
5-6 hours             93                 50                  43
7-10 hours            62                 40                  22
11-15 hours           51                 31                  20
16+ hours             54                 36                  18

Mean               2.921              3.028               2.766
Std. Deviation     1.330              1.335               1.314
ANALYSIS OF VARIANCE  F VALUE = 3.365
SIGNIFICANCE LEVEL = 0.068
T TEST - TYPE 2
                            Owns a video
Does not own a video        -1.691
                             0.091
```

Figure 5.9 F-test or analysis of variance

Notice how the significance level of the F-statistic (6.8%) allows us to be more confident about the real significance of the differences between the means (9.1% significance level).

5.5 Newman-Keuls test

Quick Reference

To request a Newman-Keuls test, type:

stat=nk*sig_level*

on the *tab* statement. *sig_level* may be 90, 95 or 99.

The standard Newman-Keuls test (as described in Winer, *Statistical Principles in Experimental Design*) is a table-level statistic that can be used as an alternative to T-tests when you want to compare the differences between the means of two or more samples of the same size.

The test is produced by the option **stat=nk***nn* option on the *tab* statement, where *nn* is 90, 95 or 99 depending on the level at which results are required. The column axis defines the groups to be compared. The row axis must include a base element, a mean (*n12*) and a standard deviation (*n17*) — these require *fac=* options on the axis elements or an *n25* element with the *inc=* option.

☞ For details about these elements, see chapter 5, 'Statistical functions and totals' in the Quantum User's Guide Volume 2.

Notes for this test are:

- This statistic calculates Q-values at the 90%, 95% or 99% level, as defined on the *tab* statement. A triangular matrix of Q-values is produced with values for each pair of means. It is labeled with the text 'NEWMAN-KEULS STATISTICS' followed by the level at which the values have been calculated.

- This statistic uses the sum of totalizable rows and the input to the mean and standard deviation rather than the base and the mean and standard deviation themselves.

- If the axis being tested contains *fac=* and *inc=*, Quantum scans backwards through the axis from the *stat=t1* element and uses whichever of the two it finds first; that is, whichever of *fac=* or *inc=* occurs closest to, but still before, the statistical element.

- Where a Q value is significant at the chosen level, an asterisk is printed underneath the value.

- The formula adjusts for the fact that in practice sample sizes are seldom identical by using the harmonic mean of the sample sizes. This approach is described by Snedecor and Cochran in *Statistical Methods* and by Miller in *Simultaneous Statistical Inference*. However, it should be noted that this test is inappropriate when sample sizes differ markedly.

The following table uses the same row and column axes as those used for the Two-Sample T-test (see Figure 5.7). It was created by the statement:

```
tab hours vcr;stat=nk95
```

```
Q15 Hours per week spent watching TV
Base: All Respondents
                    Base      Does not own a        Owns a video
                              video recorder            recorder
Base                 305                 181               124
Under 5 hours         45                  24                21
5-6 hours             93                  50                43
7-10 hours            62                  40                22
11-15 hours           51                  31                20
16+ hours             54                  36                18

Mean               2.921               3.028             2.766
Std. Deviation     1.330               1.335             1.314
NEWMAN-KEULS STATISTICS (95%)
                             Owns a video
Has no video                 2.392
```

Figure 5.10 Newman-Keuls test

The results show that, at the 95% level, there is no evidence of a difference between the mean scores.

5.6 Formulae

The formulae for the statistical tests in this chapter are shown below. The following conventions have been used in these formulae:

- In the formulae for axis-level test statistics, the formula is applied separately to the counts in each column or row, according to whether the axis containing the *stat=* option is the row or column axis:

k	Represents the number of basic count elements in the axis or segment.
n_i	Represents the (weighted) count in the *i*th cell of a row or column representing that axis.
N	Represents the (weighted) base of that row or column.
U	Represents the unweighted base of that row or column.

- In the formulae for table-level test statistics:

r	Represents the number of basic count rows from which the statistic is calculated.
c	Represents the number of basic count columns from which the statistic is calculated.
n_{ij}	Represents the (weighted) count in row *i*, column *j*.
N, N_i, N_j	Represent the (weighted) bases of the table overall, column *i* and row *j* respectively.

- A dot suffix indicates summation over the replaced index; so, for example, the formula for a column total is:

$$n_{\cdot j} = \sum_{i=1}^{r} n_{ij}$$

- The sum of factors, mean, standard deviation, standard error and sample variance of a row or column are calculated in exactly the same way as by the *n13*, *n12*, *n17*, *n19* and *n20* statements.

The sum of factors is given by: $\dot{x} = \sum n_i x_i$

The mean is given by: $\bar{x} = \frac{\sum n_i x_i}{N}$

The standard deviation is given by:

$$s = \left(\frac{\sum n_i x_i^2 - \frac{\left(\sum n_i x_i\right)^2}{N}}{N-1} \right)^{\frac{1}{2}}$$

The standard error of the mean is given by:

$$\mathrm{se}(\bar{x}) = \frac{s}{\sqrt{N}}$$

The sample variance of the mean is given by:

$$\mathrm{sv}(\bar{x}) = (\mathrm{se}(\bar{x}))^2$$

In all cases, x_i represents the factor or increment associated with the *i*th cell.

One-sample Z-test on proportions

$$z = \frac{p - \mathrm{p}_0}{\left(\frac{1}{N}\mathrm{p}_0(1-\mathrm{p}_0)\right)^{\frac{1}{2}}}$$

where: $p = \frac{n}{N}$

and p_0 is the value specified in the *fac*= option, converted to a proportion.

Two-sample Z-test on proportions

For each pair of columns:

$$z = \frac{p_2 - p_1}{\left(\left(\frac{1}{n_1} + \frac{1}{n_2}\right)\bar{p}(1-\bar{p})\right)^{\frac{1}{2}}}$$

where:

$$p_j = \frac{n_j}{N_j}$$

and:

$$\bar{p} = \frac{n_1 + n_2}{N_1 + N_2}$$

Z-test on sub-sample proportions

For each pair of columns:

$$z = \frac{p_2 - p_1}{\left(\frac{1}{N}(p_1(1-p_1) + p_2(1-p_2) + 2p_1p_2)\right)^{\frac{1}{2}}}$$

where:

$$p_j = \frac{N_j}{N}$$

Z-test on overlapping samples

For each pair of columns:

$$z = \frac{p_2 - p_1}{\left(\frac{1}{N}(p_1(1-p_1) + p_2(1-p_2) + 2p_1p_2 - 2p_{12})\right)^{\frac{1}{2}}}$$

where:

$$p_j = \frac{N_j}{N}$$

and:

$$p_{ij} = \frac{n_{ij}}{N}$$

One-sample and paired T-test

$$t = \frac{\bar{x}}{\mathrm{se}(\bar{x})}$$

is tested against Student's *t*-distribution with *N* - 1 degrees of freedom.

The two-sample T-test

For each pair of columns:

$$t = \frac{\bar{x}_2 - \bar{x}_1}{\left(\frac{((N_1-1)s_1^2 + (N_2-1)s_2^2)}{N_1+N_2-2}\left(\frac{1}{N_1}+\frac{1}{N_2}\right)\right)^{\frac{1}{2}}}$$

is tested against Student's *t*-distribution with $N_1 + N_2$ - 2 degrees of freedom.

F and T values from an nft statement

The formula for the F value of a group is as follows.

Let: *ncol* be the number of columns in the group.

coln be the number of cases in column *n.*

colnx be the sum over all cases in column *n* of the *fac*= or *inc*= values.

colnxx be the sum over all cases in column *n* of the squared *fac*= or *inc*= values.

Then:

$$F = \frac{\left(\sum_{i=1}^{ncol} \frac{(colnx)^2}{coln} - \frac{\left(\sum_{i=1}^{ncol} colnx \right)^2}{\sum_{i=1}^{ncol} coln} \right)}{\left(\sum_{i=1}^{ncol} colnxx - \frac{(colnx)^2}{coln} \right) \Bigg/ \left(\sum_{1=1}^{ncol} coln - ncol \right)}$$

The formula for the T value for a pair of columns is as follows.

Let: *colPnxx*, *colPnx* and *colPn* be the same as *colnxx*, *colnx* and *coln*, defined above, for values P=1 and P=2.

Then:

$$T = \frac{\left(\frac{col2nx}{col2n} \right) - \left(\frac{col1nx}{col1n} \right)}{\sqrt{(ts1 + ts2)}}$$

where *tsP* is calculated for P=1 and P=2 as:

$$tsP = \frac{colPnxx - \frac{colPnx^2}{colPn}}{colPn * (colPn - 1.0)}$$

F-test / one-way analysis of variance

The between-sample estimate of variance is given by:

$$MS_B = \frac{\sum_{j=1}^{c}\left(\frac{\left(\sum_{i=1}^{r} n_{ij}x_{ij}\right)^2}{N_j}\right) - \frac{\left(\sum_{j=1}^{c}\sum_{i=1}^{r} n_{ij}x_{ij}\right)^2}{N}}{c-1}$$

$$= \frac{\sum N_j \bar{x}_j^2 - \frac{\left(\sum (N_j \bar{x}_j)\right)^2}{N}}{c-1}$$

And the within-sample estimate of variance is given by:

$$MS_W = \frac{\sum_{j=1}^{c}\sum_{i=1}^{r} n_{ij}x_{ij}^2 - \sum_{j=1}^{c}\left(\frac{\left(\sum_{i=1}^{r} n_{ij}x_{ij}\right)^2}{N_j}\right)}{N-c}$$

$$= \frac{\sum SS_j - \sum N_j \bar{x}_j^2}{N-c}$$

$$= \frac{\sum ((N_j - 1)s_j^2)}{N-c}$$

where:

$$SS_j = \sum_i n_{ij} x_{ij}^2$$

is the sum of squares in column *j*.

Then the statistic:

$$F = \frac{MS_B}{MS_W}$$

is tested against Fisher's F distribution with c - 1 and N - c degrees of freedom.

Newman-Keuls test

The formula for two columns, *i* and *j*, is:

$$q_{ij} = \frac{(M_i - M_j)}{\sqrt{MS_{error} / \tilde{n}}}$$

where:

M_i	Represents the mean value in column *i*.
M_j	Represents the mean value in column *j*.
$\tilde{n}$	Represents the harmonic mean of the group and is calculated as: $\tilde{n} = \frac{k}{\sum_{c=1}^{k} \frac{1}{n_c}}$

✎ The columns are sorted so that M_i is always greater than or equal to M_j.

$$MS_{error} = \frac{\sum_{c=1}^{k} \left((x_c^2) - \frac{(x_c)^2}{n_c} \right)}{\sum_{c=1}^{k} (n_c - 1)}$$

where:

k	Represents the total number of columns in the test with a maximum of 20.
n_c	Represents the number of observations in column *c*.
x_c	Represents the sum of values in column *c*.
x_c^2	Represents the sum of the squared values in column *c*.
df	Represents the degrees of freedom, calculated as: $df = \sum_{c=1}^{k} (n_c - 1)$

References

- Miller, R. G. *Simultaneous Statistical Inference*. 2nd Edition. New York: Springer-Verlag. ISBN 0-387-90548-0

- Snedecor, G. W. and Cochran, W. G. *Statistical Methods*. 7th Edition, Ames, Iowa: The Iowa State University Press. ISBN-8138-1560-6

- Winer, B. J. *Statistical Principles in Experimental Design*. 3rd Edition. New York: McGraw-Hill. ISBN 0007070923

6 Other tabulation facilities

This chapter describes miscellaneous features of the tabulation section. These are the inclusion of C programming code or edit statements in the tabulation specifications and the sorting (ranking) of tables.

6.1 C code in the tabulation section

Quick Reference

To include C code in the tabulation section, type:

#c
C code
#endc

Statements written in the C programming language may be included in the tabulation section. Quantum will pass them directly to the load section of the run (see Appendices on running Quantum) at the point at which they occur **without** error checking.

All C code must be enclosed in the statements **#c** and **#endc**, thus:

```
#c
/* C code here
#endc
```

6.2 Editing in the tabulation section

Quick Reference

To include edit statements in the tabulation section, type:

#ed
edit statements
#end

Edit statements may by embedded in the tabulation section be enclosing them in **#ed** and **#end** statements. This can be useful when you need to do a recode in an axis but do not want to write a full edit. For instance:

```
l ax01
#ed
if (c104'2') c181=or(c151,c152,c153,c155,c155)
#end
n01First Row;c=c181'1'
```

performs exactly the same function as:

```
ed
if (c104'2') c181=or(c151,c152,c153,c154,c155)
end
  .
l ax01
n01First Row;c=c181'1'
```

✎ When you use an *#ed — #end* statement in an axis to assign a value to a column or variable that is referenced in other axes in the run, the results for that column or variable can be unpredictable in the other axes. You therefore need to take care when using this construction.

6.3 Sorting tables

Sometimes we wish rows to be arranged according to the size of the counts or values in the cells, the largest in the first row, the second largest in the second row, and so on. The keywords that control the production of sorted or ranked tables are shown in the following table.

Keyword	Explanation
sort	Row-wise sort; that is, largest row first, smallest row last.
rsort	Row-wise sort. This is the same as *sort*.
csort	Column-wise sort; that is, largest column first, smallest column last.
pcsort	Sort on percentages in the direction defined by *sort* or *csort*.
nosort	Do not sort this table.
sortcol	Selects the column on which to sort when sorting is row-wise. The default is to sort on the figures in the base column.

You may place any of these keywords on the *tab* statement of the table which is to be sorted. Alternatively you can put the keywords on the *a*, *flt* or *sectbeg* statement, in which case all tables at that level will be sorted (tables which are not to be sorted would then need the keyword *nosort* on the *tab* statement).

Sorting rows

Quick Reference

To sort the rows of a table, type:

sort

on the *a*, *sectbeg*, *flt*, *tab* or *l* statement.

Sorting is usually done on the figures in the base column. To sort on a different column, type:

sortcol

on that element.

To define an unsorted element, axis or table in an otherwise sorted axis, table or run, type:

nosort

on that element, *l* or *tab* statement.

The statement:

```
tab prefer sex;sort
```

produces a table of prefer by sex in which the product preferred by most people forms the first row, and the product preferred by fewest people is the last row.

For example:

	Base	Male	Female
Suds	59	10	49
Bubbles	49	11	38
New Foam	35	7	28
Sparkle	30	6	24
Glow	18	8	10
Extra Glow	9	2	7

As you can see, sorting is done using the figures in the base column of the table. By chance, this means that the column for women is also sorted in descending order.

If you want to sort on, say, the second column of the table, just put the option *sortcol* on the appropriate element in the axis:

```
l sex
col 10;Base;Male;%sortcol;Female
```

If you want an axis to be sorted every time it is used as a row axis, you may place *sort* on the *l* statement. Alternatively, if the axis is always to be unsorted in an otherwise sorted run, place the keyword *nosort* on the *l* statement. Both these methods are quicker and easier than remembering to place *sort/nosort* on every *tab* statement which uses the axis.

Sorting columns

Quick Reference

To sort the columns of a table, type:

csort

on the *a*, *sectbeg*, *flt*, *tab* or *l* statement.

To sort the columns of a table, place the keyword **csort** on the *tab* statement. For example, the statements:

```
tab region party;csort
ttlQ3: Which party did you vote for?
ttlBase: All voters
l region
col 110;Base;North;South;East;West
l party
col 126;Base;Labour;Conservative;Liberal/SDP
g        Base      Labour    Conserv-    Liberal/
g                                ative         SDP
p           x           x           x           x
```

would, depending on the data, produce a table such as:

```
Q3: Which party did you vote for?
Base: All votes
               Base      Conserv-      Liberal/      Labour
                            ative           SDP

Base            605           229           208          168
North           145            65            37           43
South           194            73            70           51
East            129            42            52           35
West            137            49            59           39
```

Sorting percentages

Quick Reference

To sort on percentages rather than absolutes, type:

pcsort

with either *sort* or *csort*.

To sort on percentages rather than absolutes, use **pcsort**. This is not a keyword that you can use by itself: you must use it with *sort* or *csort* since these define whether sorting is by rows or columns. Without one of these keywords, Quantum will not know which direction to sort in and will therefore ignore *pcsort*.

The direction of sorting also determines what type of percentages will be sorted. If you are sorting vertically in row order with *sort*, Quantum will sort column (vertical) percentages; if you are sorting horizontally in column order with *csort*, Quantum will sort row (horizontal) percentages. You cannot sort row percentages in row order or column percentages in column order. In terms of keyword combinations, this means Quantum will sort percentages for the following combinations only:

```
pcsort; sort; op=2
pcsort; csort; op=0
```

Sorting at different levels

Tables may be sorted at different levels: rows may be grouped together and sorted internally within the group before the group as a whole is sorted with the other elements of the axis. This is extremely useful when you are sorting tables containing nets.

All rows in a net may be sorted amongst themselves, completely separately from the rows of any other net. Then the nets may be sorted according to the number of people in each net. The resultant table will show the largest net first and within that, the most frequently occurring response.

There are two ways of sorting nets. The simplest is to place the keyword *netsort* on the *a* or *l* statement. This sorts nets and their component elements automatically, and indents each net and standard element text by a fixed amount according to the level at which the text occurs. The second method is to define the elements to be sorted as a group using the keywords *subsort* and *endsort* to mark the start and end of each sort group. You may find this method useful if you want to indent element texts by different amounts for each net level or sort group.

The sections which follow deal with each method separately, but use the same sample tables as a means of illustrating the differences and similarities between the two methods.

Sorting with netsort

Quick Reference

To create a sorted table of nets, type:

netsort[=*spaces_per_level*]

where *space_per_level* is the number of additional spaces to ident element texts at each level.

The quickest way to define an axis which will create a sorted table of nets is to place the keyword *netsort* on the *a* or *l* statement (*netsort* is not valid on *sectbeg, flt* or *tab* statements) and the keyword *sort* on the *a/sectbeg/flt/tab* statement. When these two keywords are used in the same table, a *net* statement determines not only the level at which the net is to be created (i.e. whether it is a top level net, a subnet, a sub-subnet, and so on), but also the level at which the net and the elements it contains are to be sorted in relation to the other elements in the table. This means that nets at level one will be sorted and, within them, nets at level two will be sorted, and so on. Individual elements within a net will be sorted too.

In section 3.6, 'Netting' in the Quantum User's Guide Volume 1, we said that *netsort* determines the number of spaces by which each net and element text is indented. This is still the case when *netsort* is used in sorted tables. Texts at each level below level one will be indented by two spaces per level, thus nets at level two are indented by two spaces (1×2 spaces); nets at level three are indented by four spaces (2×2 spaces). The elements comprising a net are indented by an additional two spaces. You may request a different indent by typing netsort=*n*, where *n* is the number of spaces by which to indent, instead of *netsort* by itself.

To turn off indenting for a single table in a run where indenting is the default, add the option *nonetsort* or *netsort=0* to the *l* statement of the table's row axis.

Sometimes the axis will contain rows which are not to be sorted at all. These elements require the option *nosort*. If the element is part of a group, it will retain its original position in the group even if the group later occupies a different position in the sorted output. If the element is not part of any group, it will retain its original place regardless of any other elements.

Let's take a simple example to start with. We have an axis dealing with peoples' opinions of a new chocolate bar they have tried. Responses are netted under the headings Taste and Texture, and there is also a row to gather respondents giving no answer at all. Taste and texture are to be sorted so that the one containing the most respondents appears first. The No Answer row is to remain as the last row of the table.

Within taste and texture the various comments are to be sorted so that the one mentioned by most people is printed first. Each net contains a Don't Know row which must always be the last line of the net. Element texts are to be indented by one space per net/sort level. To satisfy this specification we write:

```
tab taste brand;sort
l taste;netsort=1
n10Base                                  (350)
net1Taste Observations                   (310)
n01Too Sweet;c=c220'1'                    (80)
n01Just Right;c=c220'2'                  (105)
n01Not Sweet Enough;c=c220'3'             (95)
n01Don't Know;c=c220'4';nosort            (30)
net1Texture Observations                 (340)
n01Too Coarse;c=c221'1'                   (60)
n01Just Right;c=c221'2'                  (125)
n01Too Fine;c=c221'3'                     (85)
n01Don't Know;c=c221'4';nosort            (70)
netend1
n01No Answer;c=-;nosort                   (10)
```

The numbers in the parentheses are not part of the row specifications: they are the number of respondents giving each response.

First, let's see how each group is sorted internally. The sort is conducted on the first column (created by the first condition in the axis). We will assume that this is the base, so the figures in parentheses are totals.

With Taste, all elements down to the *net* statement for Texture are assumed to be part of the same net and sort group. Thus, we would have:

```
Taste Observations         310
 Just Right                105
 Not Sweet Enough           95
 Too Sweet                  80
 Don't Know                 30
```

The same method of sorting applies to the Texture group, except that this time the net and sort group is terminated by the *netend1* statement.

Then, the two groups are sorted in relation to each other. Since the net for Texture (340) is larger than the net for Taste (310), Texture Observations is placed first, without regard to the other values within either group. Thus, our final table is as follows:

Base	350	(This row is not part of any sort)
Texture Observations	340	
Just Right	125	
Too Fine	85	
Too Coarse	60	
Don't Know	70	(Always last within group because of *nosort*)
Taste Observations	310	
Just Right	105	
Not Sweet Enough	95	
Too Sweet	80	
Don't Know	30	
No Answer	10	(Last because of *nosort* on *no1* statement)

Notice that the three elements which were specified with the *nosort* keyword have all retained their original places in relation to the other elements in their groups. Notice, also, that the texts at level one are not indented, whereas those at level two (the elements which make up the two nets) are indented by one space, as requested with *netsort*.

Net and sort groups may be made up of any number of rows and may themselves be subdivided into smaller groups. This is called nesting. Suppose our taste and texture nets refer to the chocolate topping on a cake, and our axis also has comments about the body of the cake itself. This gives us two main groups for sorting — the topping and the cake — and within each we have two sublevels, namely the taste and the texture.

This would be specified as follows:

```
l taste;netsort
n10Base                                    (142)
net1Chocolate Topping (Net)                 (27)
n01Not Sweet Enough;c=c121'2'               (19)
net2Texture Observations (Sub-net)          (41)
n01Too Course;c=c122'1'                     (20)
n01Too Fine;c=c122'2'                       (21)
net1Cake (Net)                              (50)
net2Taste Observations (Sub-net)            (48)
n01Too Sweet;c=c123'1'                      (22)
n01Not Sweet Enough;c=c123'2'               (26)
net2Texture Observations (Sub-net)          (44)
n01Too Course;c=c124'1'                     (20)
n01Too Fine;c=c124'2'                       (24)
```

and would produce the following table:

	Total
Base	142
Cake (Net)	50
Taste Observations (Sub-net)	48
Not Sweet Enough	26
Too Sweet	22
Texture Observations (Sub-net)	44
Too Fine	24
Too Course	20
Chocolate Topping (Net)	47
Taste Observations (Sub-net)	46
Too Sweet	27
Not Sweet enough	19
Texture Observations (Sub-net)	41
Too Fine	21
Too Course	20

Figure 6.1 Sorted table of nets using *netsort*

If an *n33* statement is read immediately after a *net* statement, it is assumed to be in the same sort and indent level as the *net*. Therefore, if the text of the first net2 element was entered as:

```
net2Taste Observations on
n33the Chocolate Topping (Sub-net)
```

both lines would be indented by two spaces in the table.

If the axis contains an *ntt* to create a text-only net element, the elements in the *ntt* group will be sorted although the group as a whole, including the *ntt* element, will retain its original position in the axis. To illustrate this, let's add a group of miscellaneous comments to the end of the previous axis:

```
l taste;netsort
n10Base
net1Chocolate Topping (Net)
net2Taste Observations (Sub-net)
n01Too Sweet;c=c121'1'
n01Not Sweet enough;c=c121'2'
net2Texture Observations (Sub-net)
n01Too Course;c=c122'1'
n01Too Fine;c=c122'2'
net1Cake (Net)
```

```
net2Taste Observations (Sub-net)
n01Too Sweet;c=c123'1'
n01Not Sweet enough;c=c123'2'
net2Texture Observations (Sub-net)
n01Too Course;c=c124'1'
n01Too Fine;c=c124'2'
netend1
n01No Mentions;c=c(121,124)$ $;nosort
ntt1Miscellaneous Mentions
n01Other topping observations;c=c121'3/&'.or.c122'3/&'
n01Other cake observations;c=c123'3/&'.or.c124'3/&'
```

The table which this axis produces is:

	Total
Base	142
Cake (Net)	50
Taste Observations (Sub-net)	48
Not Sweet Enough	26
Too Sweet	22
Texture Observations (Sub-net)	44
Too Fine	24
Too Course	20
Chocolate Topping (Net)	47
Taste Observations (Sub-net)	46
Too Sweet	27
Not Sweet enough	19
Texture Observations (Sub-net)	41
Too Fine	21
Too Course	20
No Mentions	80
Miscellaneous Mentions	
Other cake observations	35
Other topping observations	29

Figure 6.2 Sorted table of nets with a text-only net

Sorting with subsort and endsort

Quick Reference

To sort the table in sections, define the start of each section with:

subsort

on the first element in the section. Mark the end of the section with:

endsort[=*num_sections*]

where *num_sections* is the number of sections that this element terminates.

The second way of specifying sorted nets is to use the keywords **subsort** and **endsort** to mark the start and end of each sort group in the axis. Although this involves you in more work, it can be useful when you want to use different amounts of indentation for different sort levels — for instance, to indent all second-level elements by 1 space and all third-level elements by 2 spaces.

If we rewrite the first netsort example it will become:

```
tab taste brand;sort
l taste
n10Base
net1Taste Observations
n01 Too Sweet;c=c220'1';subsort
n01 Just Right;c=c220'2'
n01 Not Sweet Enough;c=c220'3'
n01 Don't Know;c=c220'4';nosort;endsort
net1Texture Observations
n01 Too Coarse;c=c221'1';subsort
n01 Just Right;c=c221'2'
n01 Too Fine;c=c221'3'
n01 Don't Know;c=c221'4';nosort;endsort
netend1
n01No Answer;c=-;nosort
```

The differences between this example and the previous version are as follows:

- *netsort* has been removed from the *l* statement.
- All indentation has been done manually by preceding each element text with a space.
- The start of each subgroup has been identified by *subsort*.
- The end of each subgroup has been identified by *endsort*.

Notice that there is no need to mark the top-level sort groups (that is, the net and No Answer elements) since these will be sorted automatically by the keyword *sort* on the *tab* statement.

Groups defined with *subsort* and *endsort* may be nested up to a depth of seven levels of sorting — that is, you may type in up to seven *subsort*s before typing an *endsort* to terminate one of the groups. If one row terminates more than one group, *endsort* must be entered as *endsort=n* where *n* is the number of groups terminated. If we rewrite the specification for Figure 31.1, it becomes:

```
l taste
n10Base
net1Chocolate Topping (Net)
net2 Taste Observations (Sub-net);subsort
n01  Too Sweet;c=c121'1';subsort
n01  Not Sweet Enough;c=c121'2';endsort
net2 Texture Observations (Sub-net)
n01  Too Coarse;c=c122'1';subsort
n01  Too Fine;c=c122'2';endsort=2
net1Cake (Net)
net2 Taste Observations (Sub-net);subsort
n01  Too Sweet;c=c123'1';subsort
n01  Not Sweet Enough;c=c123'2';endsort
net2 Texture Observations (Sub-net)
n01  Too Coarse;c=c124'1';subsort
n01  Too Fine;c=c124'2';endsort=2
```

The top level of sorting is between the rows 'Chocolate Topping' and 'Cake'. Within the topping net we have two sublevels, each of which is delimited by the keywords *subsort* and *endsort*. The row entitled 'Too Fine' in the Texture subnet terminates the texture subsort as well as the sort between taste and texture observations in general, so we use *endsort=2* to indicate that we are terminating two levels of sorting.

Text-only nets with *ntt* work with *subsort* and *endsort* exactly the same as with *netsort*, except that the elements within the net will require *subsort* and *endsort* keywords if they are to be sorted within the net.

If we indent second-level texts by one space and third-level texts by two spaces, the specification for Figure 6.2 becomes:

```
tab taste brand;sort
l taste
n10Base
net1Chocolate Topping (Net)
net2 Taste Observations (Sub-net);subsort
n01  Too Sweet;c=c121'1';subsort
n01  Not Sweet enough;c=c121'2';endsort
net2 Texture Observations (Sub-net)
n01  Too Course;c=c122'1';subsort
n01  Too Fine;c=c122'2';endsort=2
net1Cake (Net)
net2 Taste Observations (Sub-net);subsort
n01  Too Sweet;c=c123'1';subsort
n01  Not Sweet enough;c=c123'2';endsort
net2 Texture Observations (Sub-net)
n01  Too Course;c=c124'1';subsort
n01  Too Fine;c=c124'2';endsort=2
netend1
n01No Mentions;c=c(121,124)$ $;nosort
ntt1Miscellaneous Mentions
n01 Other topping observations;c=c121'3/&'.or.c122'3/&';subsort
n01 Other cake observations;c=c123'3/&'.or.c124'3/&';endsort
```

and the table produced is:

	Total
Base	142
Cake (Net)	50
Taste Observations (Sub-net)	48
Not Sweet Enough	26
Too Sweet	22
Texture Observations (Sub-net)	44
Too Fine	24
Too Course	20
Chocolate Topping (Net)	47
Taste Observations (Sub-net)	46
Too Sweet	27
Not Sweet enough	19
Texture Observations (Sub-net)	41
Too Fine	21
Too Course	20
No Mentions	80
Miscellaneous Mentions	
Other cake observations	35
Other topping observations	29

Figure 6.3 Sorted table of nets created with *subsort* and *endsort*

Text-only elements in sorted tables

Text-only elements created with *n03* statements automatically attach themselves to the next numeric element in the axis and are sorted with that element. If there is no subsequent numeric element, or the sort level changes (for example, with a new net statement or with subsort) the *n03* remains unsorted. To force an *n03* to be unsorted, add the option *nosort* at the end of the statement.

n33 statements which define continuation text attach themselves to the element whose text they continue and remain with that element if it is sorted.

Subheadings created with *n23* statements are always unsorted unless they specifically carry the option *sort*.

The two examples below show how to use an *n03* to separate unsorted No Answer and Don't Know responses from the rest of the table. The table itself is shown at the end of this chapter, but the overall layout that we want to achieve is this:

Efficacy Net
  Comments
  Freshening Sub-Net
    Comments

Fragrance Net
  Comments
  Fragrance Intensity Sub-Net
    Comments

Miscellaneous
  Comments
DK/NA

The highest level is the two nets and the group of miscellaneous statements. The second level is the comments within these groups, and the third level is the comments within the two sub-nets.

Before we write our axis, there are some other points to bear in mind. First, the group of miscellaneous comments and the DK/NA row are to remain in that order at the bottom of the axis, even though the miscellaneous comments themselves are to be sorted. Second, all subnets are to remain at the end of the main net, even though their components are to be sorted. Third, since there will only be a few rows in the efficacy net, they are not to be sorted at all.

Here is our axis:

```
tab dislike ban1;sort
l dislike;netsort
ttlTable 5: What is there about this product
ttl         that you think you would dislike?
n10Total Respondents
net1EFFICACY (Net)
n33==============
n01Doesn't Work;c=c223'12';nosort
n01Other Efficacy Comments;c=c223'&';nosort
net2 Freshening (Sub-Net)
n33----------------------
n01 Doesn't Freshen Room;c=c223'7';nosort
n01 Other Freshening Comments;c=c223'&';nosort
net1FRAGRANCE (Net)
n33===============
n01Dislike Fragrance;c=c224'125'
n01Smells Artificial;c=c224'3'
n01Fragrance Name not Descriptive Enough;c=c224'4'
```

```
n01Other Fragrance Comments;c=c224'&';nosort
net2 Fragrance Intensity (Sub-Net)
n33----------------------------
n01Strong Overpowering Smell;c=c227'1'
n01Weak Fragrance/ Not Strong Enough;c=c227'2'
n01Other Fragrance Intensity Comments;c=c227'34&';nosort;endnet1
ntt1MISCELLANEOUS
n33============
n01Doesn't Last Long;c=c225'1'
n01Too Expensive;c=c225'7'
n01More Expensive Than Other Products;c=c225'8'
n01Difficult / Inconvenient to Use;c=c228'1/4'
n01Don't Buy This Type of Product;c=c226'12'
n01Might be Harmful / React Chemically;c=c226'34'
n01Allergic to This Type of Product;c=c226'678'
n01Prefer Other Types of Products;c=c226'90-'
n01Other Miscellaneous Comments;c=c226'&'.or.c229'1';nosort
n03
n01Nothing Disliked;c=c232'-';nosort
n01Don't Know / No Answer;c=c232'0&';nosort
l ban1
col 116;base=Total;London;Leeds;Cardiff;Glasgow
```

Because we want a sorted table, we start by putting the keyword *sort* on the *tab* statement. This sorts all rows which are not part of a sublevel, namely the two net rows, the miscellaneous row, and the two rows at the end of the axis. The nets are counts of people and are therefore created by *net* statements, but Miscellaneous is a heading only and is therefore created by an *ntt* at the appropriate level. The rows entitled 'Nothing Disliked' and 'Don't Know/No Answer' are to remain in their original places so they take the option *nosort*.

When the table is sorted, the two top level nets are sorted according to the number of respondents they contain, and within that the subgroups are sorted. The group of miscellaneous comments is sorted but retains its original place in the axis (after the two nets).

☞ The table which these axes produce is shown in Figure 6.4 at the end of this chapter.

The other way of writing this axis is to use *subsort* and *endsort* in place of *netsort*:

```
l dislike
ttlTable 5: What is there about this product
ttl         that you think you would dislike?
n10Total Respondents
net1EFFICACY (Net)
n33==============
n01 Doesn't Work;c=c223'12';subsort;nosort
n01 Other Efficacy Comments;c=c223'&';nosort;endsort
net2 Freshening (Sub-Net)
n33--------------------
n01  Doesn't Freshen Room;c=c223'7';subsort;nosort
n01  Other Freshening Comments;c=c223'&';endsort;nosort
net1FRAGRANCE (Net)
n33===============
n01 Dislike Fragrance;c=c224'125';subsort
n01 Smells Artificial;c=c224'3'
n01 Fragrance Name not Descriptive Enough;c=c224'4'
n01 Other Fragrance Comments;c=c224'&';nosort;endsort
net2 Fragrance Intensity (Sub-Net)
n33----------------------------
n01  Strong Overpowering Smell;c=c227'1';subsort
n01  Weak Fragrance/ Not Strong Enough;c=c227'2'
n01  Other Fragrance Intensity Comments;c=c227'34&';nosort;endsort;endnet1
ntt1MISCELLANEOUS
n33==============
n01  Doesn't Last Long;c=c225'1';subsort
n01  Too Expensive;c=c225'7'
n01  More Expensive Than Other Products;c=c225'8'
n01  Difficult / Inconvenient to Use;c=c228'1/4'
n01  Don't Buy This Type of Product;c=c226'12'
n01  Might be Harmful / React Chemically;c=c226'34'
n01  Allergic to This Type of Product;c=c226'678'
n01  Prefer Other Types of Products;c=c226'90–'
n01  Other Miscellaneous Comments;c=c226'&'.or.c229'1';nosort;endsort
n03
n01Nothing Disliked;c=c232'–';nosort
n01Don't Know/ No Answer;c=c232'0&';nosortcR
```

Here, the first row in each net has the keyword *subsort*, indicating that it and all subsequent rows form a subgroup of the net and are to be printed in rank order beneath it. The group of miscellaneous comments also form a subgroup. Subgroups are terminated by the keyword *endsort*.

Notice that even though the subnets Freshening and Fragrance Intensity are part of the nets, they are dealt with as a separate group within the net and they have their own *subsort/endsort* group. This is because we want to keep all comments to do with freshening and fragrance intensity under their respective net rows. If we left them as part of the overall Efficacy or Fragrance net, the individual comments would be sorted with the other comments in those nets according to their size, rather than being kept together as a subgroup.

Totals, statistics and manipulated elements in sorted tables

Subtotals, totals, statistical elements and elements created using *m* statements are not sorted unless you place the *sort* keyword on the element itself.

Sorting takes place after the cell counts for these elements have been calculated, not before, and subtotals and totals are not recalculated after elements have been sorted. If you are careless in the way you write your spec you could create tables that look wrong simply because the order of elements in the axis has changed after the cells' values were calculated.

Sorted tables of means

You can create a sorted table of means using just *n25* and *n12* statements. For example:

```
tab q1 banner;sort
l q1
n25;inc=c120;c=c120'1/7'
n12Mean;dec=2;sort
n25;inc=c121;c=c121'1/7'
n12Mean;dec=2;sort
n25;inc=c122;c=c122'1/7'
n12Mean;dec=2;sort
n25;inc=c123;c=c123'1/7'
n12Mean;dec=2;sort
```

An alternative is to use the *means* option on the *tab* statement. This example also uses *op=3* to print the rank numbers under each cell:

```
tab q1 banner;means;sort;op=123
l q1
n10Base
n01First mean;c=c56'1/5';inc=c56
n01Second mean;c=c57'1/5';inc=c57
n01Third mean;c=c58'1/5';inc=c58
n01Fourth mean;c=c59'1/5';inc=c59
```

Some people like to create sorted summary tables of means, standard deviations and standard errors, where the row axis consists of blocks of these three elements for a number of different columns. The table is sorted on the means, but each mean needs to be followed by its own standard deviation. To solve this problem, create an include file with the basic specification and then include it as many times as necessary with the appropriate substitutions defined on the **include* element. The specification in the include file might be as follows:

```
n12;inc=ca00;c=ca00'1/7'
n03&txt;&unl;sort
n12 Mean;dec=2;nodsp;sort
n17 Std. dev;dec=2;nodsp;subsort
n19 Std error;dec=2;nodsp;endsort
```

Absolutes/col percents

Table 5: What is there about this product that you think you would dislike?

	Total	London	Leeds	Cardiff	Glasgow
Total Respondents	687	119	182	190	196
FRAGRANCE (Net)	107	8	42	26	31
==========	15.6%	6.7%	23.1%	13.7%	15.8%
Dislike Fragrance	26	1	13	3	31
	1.2%	0.8%	7.1%	1.6%	4.6%
Smells Artificial	8	-	4	2	2
	1.2%	-	2.2%	1.1%	0.5%
Fragrance Name not Descriptive Enough	5	-	2	2	1
	0.7%	-	1.1%	1.1%	0.5%
Other Fragrance Comments	10	2	3	3	2
	1.5%	1.7%	1.6%	1.6%	1.0%
Fragrance Intensity	60	5	21	16	18
----------	8.7%	4.2%	11.5%	8.4%	9.2%
(Sub-Net)					

Strong Overpowering Smell	44	3	19	13	9
	6.4%	2.5%	10.4%	6.8%	4.6%
Weak Fragrance/ Not Strong Enough	6	2	-	1	3
	0.9%	1.7%	-	0.5%	1.5%
Other Fragrance Intensity Comments	11	-	10	3	7
	1.6%	-	5.5%	1.1%	3.6%
EFFICACY (Net)	19	-	10	2	7
==========	2.8%	-	5.5%	1.1%	3.6%
Doesn’t Work	14	-	6	2	6
	0.9%	-	3.3%	1.1%	3.1%
Other Efficacy Comments	1	-	1	-	-
	0.7%	-	0.5%	-	-
Freshening (Sub-Net)	5	-	4	-	1
----------	0.7%	-	2.2%	-	0.5%
Doesn’t Freshen Room	4	-	3	-	1
	0.6%	-	1.6%	-	0.5%
Other Freshening Comments	1	-	1	-	-
	0.1%	-	0.5%	-	-

Figure 6.4 Sorted table of nets

Absolutes/col percents

	Total	London	Leeds	Cardiff	Glasgow
MISCELLANEOUS					
Doesn’t Last Long	101	16	35	29	21
	14.7%	13.4%	19.2%	15.3%	10.7%
Don’t Buy This Type of Product	79	-	26	5	48
	11.5%	-	14.3%	2.6%	24.5%
Difficult / Inconvenient to Use	67	13	16	20	18
	9.8%	10.9%	8.8%	10.5%	9.2%
Too Expensive	62	8	23	11	20
	9.0%	6.7%	7.1%	5.8%	10.2%
Allergic to This Type of Product	37	-	13	5	19
	5.4%	-	7.1%	2.6%	9.7%
More Expensive Than Other Products	13	-	4	7	2
	1.9%	-	2.2%	3.7%	1.0%
Prefer Other Types of Product	12	1	3	2	6
	1.7%	0.8%	1.6%	1.1%	3.1%
Might be Harmful / React Chemically	9	1	3	1	4
	1.3%	0.8%	1.6%	0.5%	2.0%
Other Miscellaneous Comments	13	2	4	2	5
	1.9%	1.7%	2.2%	1.1%	2.6%
Nothing Disliked	252	75	29	98	50
	36.7%	63.0%	15.9%	51.6%	25.5%
Don’t Know/ No Answer	23	2	9	7	5
	3.3%	1.7%	4.9%	3.7%	26.3%

Figure 6.4 (continued) A sorted table of nets

7 Special T statistics

Quantum provides a variety of special types of T-test for comparing pairs or groups of rows or columns. They are:

- T-test on column proportions.
- T-test on column means.
- Significant Net Difference test.
- Paired Preference test.
- Newman-Keuls test on differences between means.
- Least Significant Difference test for means.

All tests are two-tailed. This means that the tests indicate whether there are no significant differences between the figures and whether they are not equal.

7.1 Which elements are tested?

All tests, except the Paired Preference test, compare columns of data; the Paired Preference test compares rows.

Quantum normally includes all basic elements in a test. Basic elements are elements created by *n01, n15, n10* or *n11* statements or their counterparts on *col, val, bit* or *fld* statements, and elements created by *m* statements which manipulate basic elements. All other types of element are ignored.

If you do not want to test all basic elements in the axis you may either select the ones you do want to test or reject those you do not. To exclude elements from a test, place the keyword **notstat** on the statements that create those elements. For example:

```
l age
n10Base
n01Under 30;c=c112’1’;id=A
n0130 to 50;c=c112’2’;id=B
n01Over 50;c=c112’3’;id=C
n01Not answered;c=c112’ ’;notstat
```

✎ Although the base element is not flagged with *notstat* Quantum always excludes it from all tests. If you want to test the base element you must flag it with **tstat**.

If you want to exclude more elements than you want to include, an alternative is to place *notstat* on the *l* statement to set exclusion as the default for the axis, and then to flag the elements you want to test with **tstat**. Here is the previous example in reverse:

```
l age;notstat
n10Base
n01Under 30;c=c112'1';id=A;tstat
n0130 to 50;c=c112'2';id=B;tstat
n01Over 50;c=c112'3';id=C;tstat
n01Not answered;c=c112' '
```

tstat and *notstat* are also valid on *a, flt, sectbeg* and *tab* statements to request or suppress T-tests for all tables at the given level. If you use *notstat* on a *tab* statement, for instance, Quantum will ignore *tstat* statements under the *tab* statement as well as *tstat* options in the column axis of that table. This can be useful when you want to produce several tables using the same column axis but only want the T-tests in certain of those tables. You define the axis with the necessary statements and options for the T-tests, and then request or suppress the tests using *tstat* or *notstat* on the *tab* statement.

7.2 Setting up axes for T statistics

For all tests except the paired preference test, each column element to be tested must have an ID. For the paired preference test each row element to be tested requires an ID. IDs are single upper case letters assigned with *id=*. You are therefore restricted to testing 26 columns (rows for the Paired Preference Test), because there are 26 letters in the alphabet. Quantum prints the identifiers with the element texts and uses them in the rows or columns to mark elements which differ significantly from the other elements in the test. In addition, when the elements form the columns of a table, Quantum generates an extra line of column headings showing the element IDs for each column.

✎ If you are only using one confidence level, you can use lower case IDs as well as upper case IDs. This increases the maximum number of columns (rows for the Paired Preference Test) you can test to 52.

All row axes in tables to be tested must have a base element. Although the base itself is not normally tested, the figures from the base are used to determine whether there are sufficient respondents in the element for the test to produce valid results. The base figures are also used in the calculations of some of the statistics.

7.3 T statistics on weighted tables

Quick Reference

If you are requesting T statistics on weighted tables place the keyword:

nsw

on the *a* statement.

If the table is weighted, you must weight the base element using the same weighting matrix as the *tab* statement. Either specify the weighting matrix for the whole axis using *wm=* on the *tab* statement or make sure that the base element and the *tab* statement have identical *wm=* options.

The T statistics need to know the sum of the squared weights for the axis. Each respondent's weight is squared and then added into the total for the axis. To create this figure, place the keyword **nsw** on the *a* statement. Quantum will then insert a squared weighting statement after each base element in every axis and before every *n12* as it compiles your spec. If the spec contains weighting matrices and T statistics but no *nsw* option on the *a* statement, Quantum issues a warning message at the end of the compilation stage.

If you want to see the squared weight element in your table, you may type the *nsw* statement by hand after the base element:

```
n10Base
nswSum of squared weights
```

The effective base

Quick Reference

To print an effective base which will not affect the count in any c=– elements, type:

n31*element text*[**;***options*]

To print an effective base which will affect the count in any c=– elements, type:

*element text***;effbase**

on an *n01*, *n10, n11* or *n15* statement.

Quantum creates T statistics on weighted tables using a special base figure called the ***effective base***. The purpose of the effective base is to reduce the likelihood of the statistics producing significant results simply because the weighting has made adjustments to the data.

When surveys are conducted it is impossible to interview everyone, so what usually happens is that a sample of respondents is interviewed and then the results are weighted so that they match the total population or the proportions in the total population. As an illustration, consider the case where the data is weighted by sex. The sample consists of 30% women and 70% men whereas the total population is made up of 52% women and 48% men.

☞ For further information on weighting, see chapter 1, 'Weighting'.

In this case, the weighting will inflate the answers given by women and deflate the answers given by men in order to match the population proportions. Any answers given by women will count as greater than 1 in the tables and any answer given by men will count as less than 1. To be precise, women's answers count as 52 ÷ 30, which is 1.733, and men's answers count as 48 ÷ 70, which is 0.686.

The effective base takes these adjustments into account. It is calculated by dividing the squared sum of weighting factors for an axis by the sum of the squared weighting factors; that is:

EB = (sum of weight factors)^2 / sum of squared weight factors

If the data for a particular column has both unweighted and weighted bases of 40, and comes from 12 women and 28 men, the effective base is 32.509. The calculation that produces this value is:

$(12\times1.733 + 28\times0.686)^2 / (12\times(1.733)^2 + 28\times(0.686)^2)$

$= 1600 / 49.2162 = 32.509$

The effective base is a good criterion for judging how good your weighting is. If the weighting is inflating the answers from a particular group by a large factor, the effective base tends to be much smaller than the unweighted and weighted bases. The closer the effective base is to the unweighted base, the better the weighting is.

You can print the effective base in your tables in two ways; one will affect the results of any special *c*=– elements and the other will not. *c*=– conditions can be used to produce a count of respondents who have not been included in any element since the last base in the axis. When making this calculation, Quantum ignores all respondents in the last base element, but includes all respondents in an effective base element created using the *effbase* keyword. This could result in no one appearing in the *c*=– element, thus defeating its purpose.

To print the effective base without affecting any *c*=– results, write:

n31*Element text*[**;***options*]

For example:

```
n31The Effective Base;dec=2
```

Otherwise, use the **effbase** keyword on an *n10, n11, n01* or *n15* statement as follows:

```
l ax01
n10Base
n10Effective Base;effbase
```

☞ For more details on when you should use *n31* and when you should use *effbase*, see section 5.8, 'Printing the effective base' in the Quantum User's Guide Volume 2.

In order for Quantum to report the effective base correctly, make sure that your axis is specified as follows:

- There is a base element before the effective base element.
- Any condition applied to the effective base element is the same as that applied to the most recent base element.
- The weighting matrix applied to the effective base element is the same as that applied to the most recent base element. If you just want to check what the effective base is you can use the *debug* or *tstatdebug* options to produce a file of intermediate values used in the calculation of the statistics.

☞ For more information about the *debug* or *tstatdebug* options, see 'Checking how Quantum calculated your statistics' later in this chapter.

Effective base elements in Quanvert for Windows databases

Elements created by *n31* or *effbase* will only appear in Quanvert for Windows databases if you are running a version of Quanvert for Windows later than v1.2r6.

7.4 Special T statistics and hierarchical data

All of the special T statistics are based on the assumption that the samples being compared are independent of each other. However, in levels data, there is normally a relationship between the lower levels and the higher levels, which means that cases at the lower level are not independent of each other. For example, you would not expect the voting patterns of the members of a household to be totally independent of each other, nor would you expect the various journeys or shopping trips made by an individual to be unrelated to each other. These relationships mean that the underlying assumptions required for the special T statistics are almost never satisfied when you run the tests on lower level data.

7.5 The base for T statistics

Quick Reference

The default value for a small base, when the base will be flagged as small, is 100 and the default value for a very small base, when no statistics will be calculated, is 30. To define your own small base figure, place the keyword:

smallbase=*number*

on the *a, sectbeg, flt* or *tab* statement or on the *tstat* statement for the test. To define your own very small base figure, place the keyword:

minbase=*number*

on the *a, sectbeg, flt* or *tab* statement or on the *tstat* statement for the test.

The base element plays an important part in T statistics, not least because its size determines whether or not the tests are run at all.

The previous section explained how Quantum uses the effective base rather than the weighted base in the calculation of T statistics in weighted tables. In unweighted tables the unweighted base is the same as the effective base.

Where an axis contains more than one base element, the T statistic is calculated on the most recent base. In weighted runs, a new *nsw* element will be created for each base with the same conditions as the main base.

All statistics are more reliable if they are based on large samples. If the base for a T statistic in an unweighted table, or the effective base in a weighted table, is less than 100, Quantum treats it as a small base and flags the base figure for the column in the printed table with a single asterisk.

If the unweighted base/effective base for a column (row for the Paired Preference test) is less than 30, Quantum sees it as a very small base and flags it with two asterisks and does not carry out the test. Quantum issues the message 'base too small (<30)' to warn you that a test has been suppressed.

You can specify your own values to act as small and very small bases instead of these defaults. To define your own small base, place the option:

smallbase=*number*

on the *a, sectbeg, flt* or *tab* statement or on the *tstat* statement that requests the test.

✎ The maximum value for both a small base and a very small base is 255, and the minimum value for both is zero.

To define your own value for a very small base, place the option:

minbase=*number*

on the *a, sectbeg, flt* or *tab* statement or the *tstat* statement of the test itself.

✎ It is possible to request multiple statistical tests on a single table and to define different small and very small bases on each test. However, when you do this Quantum does not use the small and very small bases defined for each test. Instead, Quantum searches the tests in a specific sequence and uses for all the tests on the table the small and very small base figures that are in force on the first test it finds. If minbase and smallbase were not defined on the test selected by Quantum, the default values are used. The sequence for the search is paired preference test, significant net difference test, T-test on column proportions, T-test on column means and Newman-Keuls test on means.

7.6 Titles for tables with T statistics

Quantum prints a footnote for each test that it runs on a table. The footnote reports the name of the test, the IDs of the elements tested, and the risk level at which the T statistic was tested for significance. If the table contains small or very small bases the footnote reminds you that * marks a small base and ** marks a very small base.

Here is simple table on which a T-test on proportions (prop) was run:

```
                    Base    North    South    East    West
                              (A)      (B)     (C)     (D)

Base                 911      125      312     248     210
Brand A              256       16     128AD    88AD     24
Brand B              192       38B       24     56B     74BC
Brand C              247       47BC      72      56      72BC
Brand D              216       24        88CD    48      40
-------------------------------------------------
Proportions: All Columns Tested (5% risk level)
```

The footnote shows that all combinations of column pairs were tested, and that the results were checked for significance at the 5% risk level — that is, the 95% confidence level.

Suppressing footnotes

Quick Reference

To suppress the footnotes that Quantum generates automatically place the keyword:

notauto

on the *a, sectbeg, flt* or *tab* statement.

You can suppress these automatic footnotes and, if you wish, replace them with titles of your choice. To suppress the footnotes, place the keyword:

notauto

on the *a, sectbeg, flt* or *tab* statement.

Defining your own titles

Quick Reference

To define your own titles for tables with T statistics use *tt* statements with any of the following keywords where you require information about the T statistic:

<<**minbase**>>	**<<smallbase>>**	**<<conf>>**
<<risk>>	**<<test>>**	**<<cols>>**

If you suppress Quantum's automatic footnotes you may wish to replace them with titles of your own. Set up your titles using *tt* statements, as you would for any other titles. The position of the *tt* statements in the spec determines when and where the titles will be printed. For example, a title under the *tab* statement will be printed on that table only whereas one in the axis will be printed in all tables of which that axis is a part.

When Quantum generates its own automatic footnotes it can pick up the variable information such as the name of the test or the names of the columns tested from the instructions it holds internally about how the tables are to be created. To allow you the same flexibility in titles that you define yourself, Quantum provides a number of special keywords that you may use on *tt* statements at *tab* level and above. When Quantum prints the title it replaces any special keywords with the appropriate type of information specific to the current table.

All keywords are enclosed in pairs of double angle brackets. The following table provides details.

Keyword	Explanation
<<**minbase**>>	The minimum base/minimum effective base.
<<**smallbase**>>	The small base setting.
<<**conf**>>	The confidence level for the current table.
<<**risk**>>	The risk level for the current table (100 minus the confidence level).
<<**test**>>	The name of the test run on this table.
<<**cols**>>	The combinations of columns tested.

Here are some examples of titles using these keywords:

```
ttl<<test>>. Columns tested: <<cols>>
ttlMinimum effective base = <<minbase>>; Small base = <<smallbase>>
```

These types of titles work with one T statistic per table only. You cannot request two different tests on the same table and define different titles with different replaceable texts for each test.

If you specify titles with these special texts for tables without T statistics, Quantum treats the replaceable texts keywords as ordinary text and prints them. If you do need to define global titles for tables with T statistics you can flag them with the word **tstat** and Quantum will print those titles only on tables with T statistics. For example:

```
ttl<<test>>. Columns tested: <<cols>>; tstat
```

☞ For further information about the *tstat* option on *tt* statements, see 'Titles for T statistics tables only' in chapter 8, 'Table texts' in the Quantum User's Guide Volume 2.

7.7 Requesting a test

Quick Reference

To request a special T-test, type:

tstat *name*; **elms**=*elm_ids* [**;clevel**=*sig1*[,*sig2*]] [**;minbase**=*num*] [**;smallbase**=*num*] [**;debug**] [**;pvals**]

underneath the *tab* or *l* statement for the table or axis in which the test is required.

You request T statistics using a **tstat** statement. This goes underneath the *tab* statement for the table on which the statistics are required, or underneath the *l* statement if the test is required whenever that axis is used. The full syntax of a *tstat* statement is:

tstat *name*; **elms**=*elm_ids* [**;clevel**=*sig1*[,*sig2*]] [**;minbase**=*num*] [**;smallbase**=*num*] [**;debug**] [**;pvals**]

where *name* is the name of the test you want to run and *elm_ids* are the ids of the elements you want to compare. Both these parameters are required; the others are all optional. The sections below explain each parameter in turn.

☞ For information about *minbase*= and *smallbase*=, see section 7.5, 'The base for T statistics'.

You may run more than one test on a table by listing the appropriate *tstat* statements under a single *tab* or *l* statement. The exceptions to this are:

- A combination of *prop* (T-test on column proportions) and *ppt* (paired preference test). This combination is not allowed because one tests rows and the other tests columns.
- A combination of *prop* and *mean* which you request with the option *propmean*.

✎ *tstat* is disallowed under *a*, *sectbeg* and *flt* statements, after *add*, *div*, *sid* and *und*, and before *and* statements.

Choosing your test

To define the test you want to run, type the name of the test immediately after the *tstat* keyword. Valid names are shown in the following table.

Name	Test
prop	T-test on column proportions.
mean	T-test on column means.
propmean	T-test on column means and proportions.
nkl	Newman-Keuls test on means.
ntd	Significant net difference test.
ppt	Paired preference test.

The automatic footnote tells you which test(s) were applied to each table, or you can define your own titles using the <<*test*>> keyword to insert the test name.

Which elements to compare?

Although you give elements identifiers and flag them with *tstat* or *notstat*, these do not in themselves tell Quantum which elements to use with a specific test. The identifier is a marker that you can use to refer to the element and *tstat* and *notstat* flag elements as eligible or ineligible for inclusion in tests.

To choose the elements for a particular test, place the option:

elms=*element_ids*

on the *tstat* statement, separated by a semicolon from the test's name. For example:

```
tstat prop;elms=ABC
```

tells Quantum to run a T-test on column proportions on all possible pairs of columns from A to C; that is, on AB, AC and BC. Any other elements in the axis are ignored even if they have identifiers and are flagged with the *tstat* option.

You can control more precisely which combinations of elements are compared by listing the pairs or sets individually, separated by commas.

The option:

```
elms=ABC,DE
```

tells Quantum to test all possible pairs within ABC plus the pair DE. Combinations of elements from the two groups, such as AD or CE are ignored.

Different tests expect to test different numbers of elements. The notes given with each test tell you the exact requirements.

The automatic footnote lists the combinations of elements tested, or you can print your own title using the <<*cols*>> keyword to list the columns tested.

If the test finds that a comparison of two elements is significantly different, it prints the element identifier of the larger value next to the cell count of the smaller value. You will see how this looks in the sample tables shown later in this chapter.

Confidence and risk levels

Confidence level and risk level are two ways of looking at the same thing. The confidence level tells you how certain you can be that any significant differences between the columns tested are due to some external factor rather than being due to chance. The risk level is the opposite of the confidence level and tells you how likely it is that any differences are simply due to chance rather than being significant for some other reason.

The sum of the confidence level and the risk level is 100, so a confidence level of 95% implies a risk level of 5%, and vice versa.

Quantum can test the significance of statistical values at a number of confidence levels. Acceptable values in Quantum for all tests except the Newman-Keuls test are 99, 95, 90, 85, 80, 75 and 68. Acceptable values for the Newman-Keuls test are 99, 95 and 90 only.

If you do not specify a confidence level, Quantum uses the default of 95% confidence.

To specify the confidence level you want for a particular test, add the option:

clevel=*level_1*[**,***level_2*]

to the *tstat* statement. If you want to set a global confidence level for all tests, place this option on the *a* statement instead.

The option requires you to specify one confidence level, but allows an optional second level. If you specify a second level it must be lower than the first level and must be separated from it by a comma.

✎ If you define two confidence levels you must specify the element IDs with *elms*= all in the same case. A mixture of upper and lower case is not allowed.

For the proportions, means and Newman-Keuls tests, Quantum checks first for significance at the higher level and prints an uppercase letter if the value is significant at that level. If the test fails, Quantum tests for significance at the lower level and prints a lowercase letter if the value is significant at that level. Otherwise, no letter is printed.

The same rules apply to the paired preference test, but significance at a given level is shown by the letter S (higher level) or s (lower level) as appropriate.

The significant net difference test uses the higher level only and silently ignores any lower level that is set.

The automatic footnote reports the risk level at which significance was tested. You can specify your own titles that show the risk level or the confidence level using the options <<*risk*>> and/or <<*conf*>> on the *tt* statement that creates the title.

Checking how Quantum calculated your statistics

Sometimes you may be surprised by the results of your tests and you will want to check how Quantum arrived at a particular value. By placing the keyword **debug** on the *tstat* statement you can have Quantum write out the intermediate figures it used to calculate the statistics. This information is written to a file called tstat.dmp. Here is part of the file that was created for the table shown earlier in this chapter (some long lines have been split and printed on two lines):

```
Props for row (#1) "Brand A"
COL         SUM(W)       SUM(WX)     SUM(WX2)      SUM(W2)        EFFBA
 A       125.000000    16.000000   16.000000  125.000000  125.000000
 B       312.000000   128.000000  128.000000  312.000000  312.000000
COLS     SUM(W) SUM(WX1) SUM(WX12) SUM(WX2) SUM(WX22) SUM(WX1X2) SUM(W2)
A B   0.000000 0.000000  0.000000 0.000000  0.000000  0.000000 0.000000
TEST     P1        P2        CORR     S.E.     D.O.F. RISK QCRIT       QVAL
A B 0.128000 0.410256 0.000000 0.049813 436.0000 0.05 2.770 -8.013446
      TVAL       PVAL
 5.666362  0.000000
```

The values in this report are as follows. The first section refers to all respondents in the row being tested:

SUM(W) The sum of weights for column A, shown as w_k in the formulae.

SUM(WX) The sum of (weights times factors) for column A, shown as

$$\sum_{i=1}^{n_k} w_{ki} X_{ki}$$

in the formulae. Factors are 1.0 for all tests except the test on column means.

SUM(WX2) The (sum of (weights times factors squared)), shown as

$$\sum_{i=1}^{n_k} w_{ki} \bullet X_{ki}^2$$

in the formulae. Factors are 1.0 for all tests except the test on column means.

SUM(W2) The sum of squared weights. Shown as

$$\sum_{i=1}^{n_k} w_{ki}^2$$

in the formulae.

EFFBA The effective base. Shown as e_k in the formulae.

The second section refers to respondents who have overlapping data in the row being tested:

SUM(W) The sum of weights for overlapping data in column A, shown as w_o in the formulae.

SUM(WX1) The sum of (weights times factors) for overlapping data in column A. Factors are 1.0 for all tests except the test on column means.

SUM(WX12) The (sum of (weights times factors)) squared for overlapping data in column A. Factors are 1.0 for all tests except the test on column means.

SUM(WX2) The sum of (weights times factors) for overlapping data in column B. Factors are 1.0 for all tests except the test on column means.

SUM(WX22) The (sum of (weights times factors)) squared for overlapping data in column B. Factors are 1.0 for all tests except the test on column means.

SUM(WX1X2) SUM(WX1) times SUM(WX2).

SUM(W2) The sum of squared weights.

The third section refers to the test itself:

P1 For a proportions test, the proportion in column A. For a means test, the value is labeled MEAN1 and shows the mean for column A.

P2 For a proportions test, the proportion in column B. For a means test, the value is labeled MEAN2 and shows the mean for column B.

CORR The correlation co-efficient.

S.E The standard error.

D.O.F. The degrees of freedom.

RISK The risk level requested for this test (100–*clevel*).

QCRIT The critical value of the Student *t* distribution times $\sqrt{2}$.

QVAL The value calculated by the statistic.

TVAL The T-value calculated as QVAL divided by $\sqrt{2}$.

PVAL The P-value.

If you want to see this type of information for all special T statistics in a run, place the option *tstatdebug* on the *a* statement and omit *debug* from the individual *tstat* statements.

Printing probability values

Probability (P) values tell you how likely it is that a value returned by a statistic could have happened by chance in a 2-tailed T-test. The general rule is that the smaller the probability the greater the significance of the statistic. A probability value of less than 0.05 means that there is less than a 5% chance that the result is due to chance. This probability corresponds to the 5% risk and 95% confidence levels. If the statistic has a probability of less than 0.05 then the results are significant at the 95% confidence level.

To see P-values include the option **pvals** on the *tstat* statement.

Quantum reports probabilities as a decimal value with three decimal places, where 1.000 corresponds to 100%. A probability of 0.862 indicates that the value of the statistic is 86.2% likely to occur by chance in a 2-tailed T-test. The number of decimal places is not affected by either *dec=* or *decp=*.

The way Quantum reports P-values varies according to the type of test you are running; for example, with a test on column proportions Quantum writes the P-values out to a separate file whereas with a significant net difference test the P-values are printed on the table itself.

7.8 Overlapping data

Quick Reference

If the T-test is to be performed on overlapping data, type:

overlap

on the *a, sectbeg, flt* or *tab* statement.

To suppress the footnote that is automatically generated whenever the overlap formulae are used place the keyword:

nooverlapfoot

on the *a, sectbeg, flt* or *tab* statement.

The T-tests on column proportions and column means, the significant net difference test and the Newman-Keuls test will work on mutually exclusive or overlapping data.

When the data is overlapping, the formulae which calculate the statistics must be modified to take this into account. To do this, place the keyword **overlap** on the *tab* statement of the table to be tested, or on the *a, flt* or *sectbeg* statement above it:

```
tab region reasons;overlap
tstat prop;elms=AB,CD
```

To clarify the difference between mutually exclusive data and overlapping data consider a question that asks respondents which of two products they tried. If you create an element that is 'Tried one product only' it will be single coded so *overlap* is not needed. The element 'Tried both A and B' will always be multicoded so you must use *overlap*. Elements such as 'Tried A or B, or A and B' could be single coded if no one tried more than one product, or multicoded if some respondents tried both products. You should always use *overlap* with elements of this type. You should also use *overlap* if you are testing combinations of overlapping and non-overlapping data.

If you do not use *overlap* when the data is overlapping you can obtain incorrect results. Specifying *overlap* when the data is mutually exclusive does not affect the validity of the statistic as this part of the calculation will simply produce a value of 0.0.

When you request tests on overlapping data, Quantum displays the message 'Overlap formulae used' as part of the standard footnote, just above the small base/very small base messages. You may suppress this message by placing the keyword **nooverlapfoot** in the *a*, *sectbeg*, *flt* or *tab* statement. To switch the message back on after switching it off use **overlapfoot**.

For a more on the theory of overlapping samples, see Kish, *Survey Sampling*.

7.9 The T-test on column proportions

Quick Reference

To request a T-test on column proportions, type:

tstat prop [**;***options*] [**; propcorr**]

after the *tab* or *l* statement. Use *propcorr* to apply a continuity correction to the numerator of the proportion's T-value.

This test looks at each row of the table independently and compares pairs of columns to test whether the proportion of respondents in one column is significantly different from the proportion in the other. For each pair in which the difference between the columns is significant, the ID of the smaller column is printed beside the figures in the larger column. For example, if you compare columns A and B, and proportion A is found to be significantly smaller than proportion B, the letter A will be printed beside the figures in column B.

If two confidence levels have been defined, the ID will be shown in uppercase if the test was significant at the higher level, or in lowercase if it was significant at the lower level.

The T-test is a two-tailed test. You can check which side of the curve the T statistic is on by running the test with one of the options *tstatdebug* or *debug* and looking in the tstat.dmp file. Negative values are on the left of the curve and positive values are on the right.

You may run this test by itself or with a T-test on column means.

With a T-test on column proportions, either on its own or with a T-test on column means, a continuity correction can reduce the difference between the two proportions compared. It is applied to the numerator of the proportion's T-value. If the difference between the two proportions is positive, Quantum subtracts the correction value from the difference. If the difference is negative, Quantum adds the correction value to the difference.

✎ When you use *propcorr* with a *propmean* test, the correction is applied to the proportions part of the test only. It is ignored for the means test.

To request a T-test on column proportions:

1. Insert a **tstat prop** statement after the *tab* or *l* statement of the table or axis to be tested.

   To run this test with a similar test for column means, use **tstat propmean** instead.

   Any number and combination of column pairs may be specified as noted earlier in 'Which elements to compare?'.

2. To request the optional continuity correction, add the keyword **propcorr** to the *tstat prop* or *tstat propmean* statement.

Example of a T-test on column proportions

This example tests the differences between the proportions of people trying various types of wine in London and Manchester. In the comparison between columns B and D (women in London and women in Manchester) a significant difference has been found for those trying brand A: the letter D beside the figure for women in column B indicates that the proportion of women in London is significantly larger than the proportion of women in Manchester. A similarly significant difference exists between columns C and D for brand A. The other comparisons in that row (columns A and B, and A and C) do not produce significant differences at the 90% level.

The program written for this test was:

```
tab wine ban01;op=2;nopc
tstat prop;elms=AB,CD,AC,BD;clevel=90
l ban01
n00;c=c231'1'
n23London;hdlev=1;unl1
col 132;Male;%id=A;Female;%id=B
n00;c=c231'2'
n23Manchester;hdlev=1;unl1
col 132;Male;%id=C;Female;%id=D
l wine
n10Base
col 111;Brand A;Brand B;Brand C;Brand D;Brand E;Brand F;
+Brand G
n01Non;c=-7
```

The table produced is:

	London		Manchester	
	Male (A)	Female (B)	Male (C)	Female (D)
Base	90	86	90	73
Brand A	8	9D	9D	2
Brand B	8	11	14A	11
Brand C	18B	9	11	10
Brand D	8	6	7	10
Brand E	21	23	24	22
Brand F	6	9	7	6
Brand G	21	19D	20D	12
None	10	14	10	27CB

Proportions: Columns Tested (10% risk) AB / CD / AC / BD

Figure 7.1 T-test on column proportions

In this example, the test was carried out using a 90% confidence level. This means that significant differences at this level are shown in the table. To see the actual level of significance, we need to look at the P-values.

P-values for a T-test on column proportions

Quantum generates a P-value for each pair of columns tested in each row. These show the actual level of significance. So that all these values may be viewed in a legible form, Quantum writes them out to a separate log file, tstat.dmp. This file is laid out so that there is one display column per pair of columns tested, and one row per row tested. Headings indicate which display column refers to each pair, and the side text for each row (truncated if necessary) is printed at the side of each row.

For example:

	A/B	A/C	A/D	B/C	B/D	C/D
Effect bas	1.000	1.000	1.000	1.000	1.000	1.000
18-24	0.964	0.862	0.629	0.811	0.562	0.754
25-34	0.400	0.356	0.263	0.840	0.664	0.830
35-44	0.212	0.185	0.547	0.005	0.054	0.482
45-54	0.452	0.455	0.551	0.106	0.155	0.889
55-64	0.038	0.850	0.340	0.023	0.310	0.255

Figure 7.2 P-values for a T-test on column proportions

The P-values in this example show that we can have confidence at the 96.2% level that there is a difference between the proportions of respondents aged 55-64 in the A and B columns.

Lines in the T statistics log file may be a maximum of 160 characters long, and a minimum of five characters is required for each P-value. If your table has many columns and you have requested T-tests for many pairs of columns, you may find that Quantum has insufficient space to write all the information it needs in one line. You'll know when this happens because you will see error messages of the form:

```
Page n, table m: pvals
Error: cannot print number cols in width 160
```

This does not affect the validity of the T statistics in the table; it merely points to a problem in writing to the log file.

7.10 The T-test on column means

Quick Reference

To request a T-test on column means, type:

tstat mean [**;***options*]

after the *tab* or *l* statement.

This test is similar to the T-test on column proportions, except that instead of comparing proportions it compares column means which have been created with *n12* statements (this test does not work on tables of means). Where the mean in one column is significantly different from the other mean in the pair, the ID of the smaller mean is printed next to the figures in the larger column.

If two confidence levels have been defined, the ID will be shown in uppercase if the test was significant at the higher level, or in lowercase if it was significant at the lower level.

To request a T-test on column means:

1. Insert a **tstat mean** statement after the *tab* or *l* statement of the table or axis to be tested.

   To run this test with a similar test for column proportions, use the option **tstat propmean** instead.

2. Make sure that the row axis contains an *n12* statement for the mean.

The Quantum programs required to run a T-test on column means, and the tables produced, are as shown above for the test on column proportions, except that the row axis must contain an *n12*. The test will place letters next to those means which are significantly different from those with which they were compared.

The 'P-values for a T-test on column proportions' section above is also applicable to the T-test on column means.

☞ For an alternative method of testing means, see section 7.14, 'Testing means using the least significant difference test'.

7.11 The Newman-Keuls test

Quick Reference

To request a Newman-Keuls test, type:

tstat nkl [**;***options*]

after the *tab* statement or after the *l* statement.

This test compares the differences between the means of two or more samples. For each pair of means in which the difference is significant at the chosen level, the ID of the smaller column(s) is printed next to the figures in the larger column, as for the T-test on column proportions in section 7.9, 'The T-test on column proportions'.

If two confidence levels have been defined, the ID will be shown in uppercase if the test was significant at the higher level, or in lowercase if it was significant at the lower level.

To request a Newman-Keuls test:

- Insert a **tstat nkl** statement after the *tab* statement of the table to be tested or after the *l* statement of the axis to be tested.

  The test is applied to all rows for which the *tstat* flag is set. If the row is an *n12*, then the calculation uses the *means* formulae; if not, the *propns* formulae are used.

The 'P-values for a T-test on column proportions' section above is also applicable to the Newman-Keuls test.

7.12 The significant net difference test

Quick Reference

To request a significant net difference test, type:

tstat ntd [**;***options*]

after the *tab* or *l* statement, and:

stat ntd, *element_ids*

in the axis at the point at which the results should be printed.

This test deals with each row independently and compares the proportions in four columns at a time to test whether the difference between the values in the first pair of columns is significantly different from the difference between the values in the second pair of columns. For example, when comparing columns A, B, C and D, the difference between A and B will be tested against the difference between C and D to see whether the difference between the two is significant.

To request a significant net difference test:

1. Insert the statement **tstat ntd** under the *tab* or *l* statement which creates the table or axis to be tested.

   Columns to be tested must be defined in groups of exactly four. For example:

   ```
   stat ntd;elms=ABCD,EFGH
   ```

   If the number of sets of letters does not match the number of *stat ntd* statements in the axis, the excess of either type is ignored and a warning message to this effect is issued. Therefore, if there are three groups of columns defined with *tstat* but only two *stat ntd* elements in the axis, only two statistics will be calculated.

2. For each group of columns to be tested, place a **stat ntd** statement in the column axis to determine where the results for those columns should be printed.

☞ The *stat ntd* statement has the same format as the statements discussed in chapter 4, 'Descriptive statistics' and chapter 5, 'Z, T and F tests'.

---

3. Optionally, add *decp*= with a value greater than 0 to the row elements of the table being tested. (*decp*= at any higher level has no effect on the way Quantum prints this statistic.)

The number of decimal places shown for the T statistic is controlled by *decp*=, for which the default is zero. If the value of the T statistic is less than one you could find that the T statistic is replaced by the *spechar* characters. Setting the number of decimal places to one or more prevents this happening.

Example of a significant net difference test

This example tests whether the difference between working and non-working women who have tried non-alcoholic wine in London is significantly different from the difference between the same groups of women in Manchester. The column labeled ABCD shows the value of the T statistic for each row.

The Quantum program is:

```
tab wine women;op=2;nopc;c=c132'2';decp=0
tstat ntd;elms=ABCD
l women
n11TOTAL
n00;c=c231'1'
n23London;hdlev=1;unl1
n01Works;c=c149'12';id=A
n01Does!Not!Work;c=c149n'12';id=B
n00;c=c231'2'
n23Manchester;hdlev=1;unl1
n01Works;c=c149'12';id=C
n01Does!Not!Work;c=c149n'12';id=D
n23;hdlev=1
stat ntd,ABCD
l wine
n10TOTAL
n00;c=c111'1/7'
col 111;Past 7 days;1-2 weeks;2-4 weeks;1-3 months;3-6 months;
+over 6 months
n00
n01Never tried non-alcoholic wine;c=–
```

The table it produces is:

		London		Manchester		
	TOTAL	Works (A)	Does Not Work (B)	Works (C)	Does Not Work (D)	ABCD
TOTAL	1800	572	440	382	384	
Past 7 days	23	35	20	23	11	
1-2 weeks	15	19	15	12	11	
2-4 weeks	13	15	10	17	11	
1-3 months	12	8	15	23	6	25
3-6 months	6	8	5	6	6	
Over 6 months	6	4	5	6	11	
Never tried non-alcoholic wine	24	12	30	14	43	-11

NTD: Columns Tested (5% risk level) - A/B/C/D

Figure 7.3 Significant net difference test

This example shows that there are significant differences in the 1-3 months and Never tried rows. However we cannot tell the degree of significance from these results. For that we need to look at the P-values.

P-values for the significant net difference test

In significant net difference tests, the P-values are printed in the ntd column in place of the value returned by the T statistic. Here is the same table as shown above but with P-values instead of the value of the T statistic shown in the ABCD column.

		London		Manchester		
	TOTAL	Works (A)	Does Not Work (B)	Works (C)	Does Not Work (D)	ABCD
TOTAL	1800	572	440	382	384	
Past 7 days	23	35	20	23	11	0.432
1-2 weeks	15	19	15	12	11	0.208
2-4 weeks	13	15	10	17	11	0.894
1-3 months	12	8	15	23	6	0.000
3-6 months	6	8	5	6	6	0.241
Over 6 months	6	4	5	6	11	0.059
Never tried non-alcoholic wine	24	12	30	14	43	0.008

NTD: Columns Tested (5% risk level) - A/B/C/D

Figure 7.4 Significant net difference test with P-values

The P-value is the probability that the difference is significant. In this table, any value less than or equal to 0.05 indicates a difference that is significant at the 95% confidence level or higher.

The P-values in this example show highly significant differences in the 1-3 months and Never tried rows. The differences between columns A and B are significantly different from the differences between columns C and D. No other rows have significant differences at the 95% confidence level.

7.13 The paired preference test

Quick Reference

To request a paired preference test, type:

tstat ppt [**;***options*] [**;ppnse**]

after the *tab* or *l* statement, and:

stat ppt, *element_text*

in the axis at the point at which the letters should be printed. *ppnse* tells Quantum to print NS or E depending on whether or not the value of the statistic is significantly different from $\sqrt{2}$.

This test deals with each column independently and compares pairs of rows to see whether the figures in each pair differ significantly from one another. If the results of the test are significant at the selected level, the letter S is placed in that column. Thus, if the proportion of women preferring Brand A is larger than those preferring Brand B, and the difference between the two proportions is significant, the letter S will be printed in the column for women.

If two confidence levels have been defined, significance at the higher level is shown by an uppercase S and significance at the lower level is shown by a lowercase s.

This test is generally used in product tests where respondents test two or more products: the rows are then the products tested.

✎ If a proportion is zero or one, meaning that no-one preferred, for example, brand A, the correlation coefficient is set to –1.0, the test is carried out and the letter S is printed in the appropriate column.

The presence of overlapping data is irrelevant with this test, and since overlap calculations involve more processing time and a larger nums file, you are advised not to use *overlap* with this test.

To request a paired preference test:

1. Insert a **tstat ppt** statement underneath the *tab* or *l* statement which creates the table or axis to be tested.

   Row IDs must be entered in sets of exactly two, for example:

   ```
   tab brands sex
   tstat ppt;elms=AB,CD,AC,AD,BC,BD
   ```

2. Insert a **stat ppt** statement in the row axis to create a row in which the significance letters should be printed:

   ```
   stat ppt,Paired Preference Test
   ```

✎ You need to insert one *stat ppt* statement into the axis for each pair of rows you are comparing; for example you would need to insert six *stat ppt* statements for the *tstat ppt* shown in step 1 above.

---

3. Optionally, place the option **ppnse** on the *tstat* statement to have the letters NS and E printed in the columns where the difference is not significant at the given level. NS indicates that the value of the statistic is significantly different from $\sqrt{2}$; E indicates that the value of the statistic does not differ significantly from $\sqrt{2}$.

Example of a paired preference test

This example tests whether the number of respondents preferring brand A is significantly different from the number preferring brand B.

```
tab opref age;notauto
tstat ppt;elms=AB;ppnse
foot
ttlRows tested A B with nse
l age
n0113-29;c=c205'12'
n0113-39;c=c205'123'
n0113-55;c=c205'1/5'
g       Target                    Total
g       market  Respondents  respondents
g    age 13-29    age 13-39    age 13-55
u2
p            x            x            x
l opref
n10Base
n03Overall preference
n01Prefer Brand A (A);c=c209'1';id=A
n01Prefer Brand B (B);c=c209'2';id=B
n01No preference;c=c209'3'
stat ppt,Paired Preference
n03
n01Total;c=c209'123';notstat
```

The table produced is:

	Target market age 13-29	Respondents age 13-39	Total Respondents age 13-55
Base	206	275	303
Overall preference			
Prefer Brand A (A)	104	147	166
Prefer Brand B (B)	101	126	135
No Preference	1	2	2
Paired Preference	E	NS	NS
Total	206	275	303

Rows tested A B with nse

Figure 7.5 Example of a paired preference test

In this example, the paired preference row shows that in the target market there are no significant differences at the 5% risk level between the number of respondents preferring Brand A and Brand B. However, there are significant differences at the 5% risk level in the other categories.

P-values for paired preference tests

In paired preference tests with the P-value option, the P-value is printed instead of the letter indicators S, NS, E or blank, except that the S is printed next to the value if appropriate. For example, a paired preference test without P-values might produce the following output:

	Base	Single (A)	Married (B)	Divorced (C)	Other (D)
Base	200	45*	49*	59*	47*
Effective base	100	100	100	100	100
Prefers A	27	27	29	22	30
Prefers B	26	27	18	31	26
No preference	4	7	4	3	0
Total	56	60	51	56	55
1st ppt		E	S	E	E
Prefers C	18	13	20	22	15
Prefers D	23	22	29	17	23
No preference	4	4	0	5	6
Total	45	40	49	44	45
2nd ppt		S	E	E	E

Paired Preference: Columns Tested (32% risk level) - A/B - C/D
*small base

Figure 7.6 Paired preference test without P-values

When run with the *pvals* option on the *tstat* statement, the output would be:

```
                Base   Single   Married  Divorced    Other
                          (A)      (B)       (C)       (D)

Base             200      45*      49*       59*       47*
Effective base   100     100      100       100       100
Prefers A         27      27       29        22        30
Prefers B         26      27       18        31        26
No preference      4       7        4         3         0
Total             56      60       51        56        55

1st ppt                 1.000    0.304S    0.370     0.698

Prefers C         18      13       20        22        15
Prefers D         23      22       29        17        23
No preference      4       4        0         5         6
Total             45      40       49        44        45

2nd ppt                 0.324S   0.420     0.533     0.352
-------------------------------------------------------------
Paired Preference: Columns Tested (32% risk level) - A/B - C/D
*small base
```

Figure 7.7 Paired preference test with P-values

Although this test was carried out using the 32% risk level (68% confidence level), using the P-values enables the actual level of significance to be displayed. For example, there is a difference between the numbers preferring Brand A to Brand B at the 30.4% risk level among married respondents. Note that the value of 1.000 displayed in the Single column for the first paired preference statistic indicated that there are no differences between the numbers preferring the two brands.

7.14 Testing means using the least significant difference test

Quick Reference

To request a least significant difference test, type:

tstat *statistic_name* [**;***options*] **;lsd**

where *statistic_name* is *mean* or *propmean*.

There is an alternative to testing means using the mean and propmean tests. This involves the use of a least significant difference (LSD) in which the variance is computed across all the columns defined with one *elms*= keyword at once rather than pairwise.

The calculation used depends on whether the sample is independent (non overlapping) or overlapping. In both cases, the LSD is compared against the difference between each pair of means. When the difference is greater than the LSD it is significant and the column is marked with a letter in that same way as for other tests. The value of the LSD is printed directly under the mean to which it relates and, for each value, Quantum reports the group of elements (non-overlapping groups) or pair of columns (overlapping groups) tested.

To request a least significant difference test:

- Append the keyword **lsd** to the *tstat* statement for your test on column means or column means and proportions.

7.15 Formulae

The formulae for the statistical tests described in this chapter are shown below.

When you ask for special T statistics, Quantum compares the T statistic that is calculated from your data with a formula that calculates the critical values of a T distribution. If the number calculated from the data is greater than the number derived from the formula, this is significant and you should expect to see a T statistic letter on your table. (The number is treated as significant if greater, regardless of whether it is positive or negative.) If you have asked Quantum to print the intermediate figures used in the calculation of the statistics, you will see that the last two figures shown per test are the significance value from the formula, and the T statistic which is derived from the data.

General notation

X_{ki} is the value of the *i*th case in column *k*.

w_{ki} is the weight for the *i*th case in column *k*.

n_k is the number of cases in column *k*.

w_k is the sum of the weights of the cases in column *k*; that is

$$\sum_{i=1}^{n_k} w_{ki}$$

r_k is the value of the cell count in the row being tested in column *k*.

p_k is the proportion of w_k in the cell; that is

$$r_k / w_k$$

π_k is the population proportion from which sample *k* is drawn.

μ_k is the mean of the population from which sample *k* is drawn.

σ_k^2 is the variance of the population from which sample *k* is drawn.

$\bar{X}_k$ is the mean of sample *k*.

S_k^2 is the variance of sample *k*.

e_k is the effective base in column *k*. It is calculated as

$$e_k = \frac{\left(\sum_{i=1}^{n_k} w_{ki} \right)^2}{\sum_{i=1}^{n_k} w_{ki}^2}$$

n_o is the number of cases in overlap; that is the number of cases in both columns being tested.

e_o is the effective base of the cases in overlap. It is calculated as

$$e_o = \frac{\left(\sum_{i=1}^{n_o} w_{oi}\right)^2}{\sum_{i=1}^{n_o} w_{oi}^2}$$

w_o is the sum of the weights of the overlapping cases.

X_{oi} is the value of the *i*th overlapping case.

w_{oi} is the weight of the *i*th overlapping case.

T-test on column means

This test compares the values of the means in two columns of a table. For each of the two columns (k=1, 2) we are testing the hypothesis that the population means are the same; that is $\mu_1-\mu_2=0$.

The sample means are calculated as

$$\bar{X}_k = \frac{\sum_{i=1}^{n_k} w_{ki} X_{ki}}{\sum_{i=1}^{n_k} w_{ki}}$$

The sample variance is calculated as

$$S_k^2 = \frac{1}{w_k - 1} \left(\sum_{i=1}^{n_k} w_{ki} \bullet X_{ki}^2 - \frac{\left(\sum_{i=1}^{n_k} w_{ki} X_{ki}\right)^2}{\sum_{i=1}^{n_k} w_{ki}} \right)$$

It is assumed that each sample is drawn from the same population, so $\sigma_1^2 = \sigma_2^2$.

We can therefore represent the population variance from which each sample is drawn as σ^2.

As we do not know the value of σ^2 we use S^2, a pooled estimate of σ^2, where

$$S^2 = \frac{v_1 + v_2}{\left(w_1 - \frac{w_1^2}{w_1}\right) + \left(w_2 - \frac{w_2^2}{w_2}\right)}$$

$$v_k = (w_k - 1) \bullet S_k^2$$

In the case of unweighted data, this reduces to

$$S^2 = \frac{(n_1 - 1)S_1^2 + (n_2 - 1)S_2^2}{n_1 + n_2 - 2}$$

In the case of no overlap and if $n_1, n_2 > 30$, the variable

$T = \dfrac{\bar{X}_1 - \bar{X}_2}{S\sqrt{\dfrac{1}{e_1} + \dfrac{1}{e_2}}}$ is distributed *t* with $e_1 + e_2 - 1$ degrees of freedom.

In the case of overlap, the T statistic must be adjusted and so

$T = \dfrac{\bar{X}_1 - \bar{X}_2}{S\sqrt{\dfrac{1}{e_1} + \dfrac{1}{e_2} - \dfrac{2r_o e_o}{e_1 e_2}}}$ and is distributed *t* with $e_1 + e_2 - e_o - 1$ degrees of freedom.

where r_o is the correlation coefficient, defined as

$$r_o = \frac{\sum_{i=1}^{n_o} x_{oi}^2 - \sum_{i=1}^{n_o} \frac{x_{1i}x_{2i}}{w_o}}{\sqrt{\left[\sum_{i=1}^{n_1} x_{1i}^2 - \frac{\left(\sum_{i=1}^{n_1} x_{1i}\right)^2}{w_o}\right]\left[\sum_{i=1}^{n_2} x_{2i}^2 - \frac{\left(\sum_{i=1}^{n_2} x_{2i}\right)^2}{w_o}\right]}}$$

where

$$x_{ki} = X_{ki} \bullet w_{ki}$$

$$x_{ki}^2 = X_{ki}^2 \bullet w_{ki}$$

This formula reduces to 1 for all cases except for overlapping grid tables.

For a more on the theory of overlapping samples, see Kish, *Survey Sampling*.

T-test on column proportions

This test compares the values of the proportions in two columns of a table. For each of the two columns (k=1, 2) we are testing the hypothesis that the population proportions are the same; that is $\pi_1 - \pi_2 = 0$, where the sample proportions are p_1 and p_2, defined as

$$p_1 = \frac{r_1}{w_1} \quad p_2 = \frac{r_2}{w_2}$$

It is assumed that the samples are drawn from a common population so we estimate the population proportion variance, $\hat{p}$, using the formula

$$\hat{p} = \frac{r_1 + r_2}{w_1 + w_2}$$

The variable *T* is calculated as

$$T = \frac{p_1 - p_2 \pm cc}{\sqrt{\frac{\hat{p}(1-\hat{p})}{1 - \frac{1}{e_1 + e_2}}\left(\frac{1}{e_1} + \frac{1}{e_2}\right)}}$$ and is distributed *t* with $e_1 + e_2 - 1$ degrees of freedom.

where

cc is the continuity correction, defined as

$$cc = \frac{1}{2}\left(\frac{1}{e_1} + \frac{1}{e_2}\right)$$

In the case of overlap, the T statistic must be adjusted for the covariance term and becomes

$$T = \frac{p_1 - p_2 \pm cc}{\sqrt{\frac{\hat{p}(1-\hat{p})}{1 - \frac{1}{e_1 + e_2}}\left(\frac{1}{e_1} + \frac{1}{e_2} - \frac{2e_o r_o}{e_1 e_2}\right)}}$$

and is distributed *t* with $e_1 + e_2 - e_o - 1$ degrees of freedom.

where r_o is the correlation coefficient, defined as

$$r_o = \frac{p_o - p_1 p_2 / w_o}{\sqrt{[p_1 - p_1^2 / w_o][p_2 - p_2^2 / w_o]}}$$

For more on the theory of overlapping samples, see Kish, *Survey Sampling*.

The significant net difference test

For any row, and any set of four columns (k=1,2,3, and 4) let

- The sum of weights (w_k),
- The sum of squared weights (w_k^2),
- The effective base (e_k) and
- The proportions (p_k)

be as previously described.

Let p_{kj} represent the column proportion in the overlap between columns k and j, and e_{kj} represent the effective base in the overlap.

The estimated population variance S^2 is calculated as

$$S^2 = \sum_k \frac{p_k(1-p_k)}{e_k} - \sum_{k \neq j} \frac{2(p_{kj}-p_{kj}^2)(e_{kj}-1)}{(e_k-1)(e_j-1)}$$

Then

$$T = \frac{(p_2-p_1)-(p_4-p_3)}{\sqrt{S^2}}$$

and is distributed t with $e_1 + e_2 + e_3 + e_4 - e_o - 4$ degrees of freedom.

For a more on the theory of overlapping samples, see Kish, *Survey Sampling*.

The paired preference test

For any column and any pair of paired preference rows, let:

- w_0 be the weighted base for the column
- $w^2{}_0$ be the sum of squared weights for that column
- $e_o = \dfrac{{w_0}^2}{w^2{}_0}$ be the effective base for that column

For rows 1 and 2 in this column,

- P_j is the proportion in the *j*th row
- c_j is the absolute value (as created by *op=1*) in row *j*

Let the correlation co-efficient between the two rows be:

$$r = \frac{-p_1 p_2}{\sqrt{p_1 p_2 (1-p_1)(1-p_2)}}$$

Let:

$$S^2 = \frac{\left(\frac{c_1 + c_2}{2w_0}\right) \cdot \left(1 - \frac{c_1 + c_2}{2w_0}\right)}{1 - \frac{1}{2e_0}}$$

Then:

$$P = \frac{p_1 - p_2}{\left(S^2 \cdot \frac{2}{e_0}(1-r)\right)^{\frac{1}{2}}}$$

The least significant difference test

For independent (non-overlapping) samples:

For any set of columns defined with an *elms=* keyword, let:

df be the degrees of freedom, calculated as:

df = N - ncols

where *N* is the number of observations in all means, and *ncols* is the number of columns in the set.

t(df) be the critical value of *t* at *df* degrees of freedom at some confidence level defined by the user.

s be the square root of the mean square within the columns defined with *elms*=:

$$s = \left(\frac{\sum_{i=1}^{ncols} xsq_i - \frac{\sum_{i=1}^{ncols} x_i x_i}{n_i}}{N - ncols} \right)^{1/2}$$

where:

xsq_i is the sum over all observations in column *i* of the 'squares of *x*'.

x_i is the sum of the values over all observations in column *i*.

n_i is the number of observations in column *i.*

N is the sum over all observations in column *i* of n_i (that is, the total number of observations in the *elms*= set of columns).

h is the harmonic mean of the number of observations in each group, calculated as:

$$h = \frac{ncols}{\sum_{i=1}^{ncols} \frac{1}{n_i}}$$

Then, the Least Significant Difference is calculated as:

$$LSD = t(df) \bullet s \bullet \left(\frac{2}{h}\right)^{1/2}$$

For overlapping samples:

The significance is tested using the normal tstat method. The LSD is then computed as follows:

For each pair of columns defined with an *elms*= keyword, let:

df	be the degrees of freedom, calculated by subtracting 1 from the number of observations in the first column.
t(df)	be the critical value of *t* at *df* degrees of freedom at some confidence level defined by the user.
se	be the standard error of mean difference calculated as in the standard tstat computation (section 7.10, 'The T-test on column means').

Then, the Least Significant Difference is calculated as:

$$LSD = t(df) \bullet se$$

The Newman-Keuls T statistic

Quantum performs the following steps:

- Calculates the formula for each pair of columns.
- Calculates the sum of these formulae for each column.
- Sorts the columns in ascending order of the sum of the formulae.
- Compares the significance of each pair of columns with the appropriate value in a lookup table.

The formula for two columns *a* and *b*, is:

$$Q_{ab} = \frac{\frac{x_a}{n_a} - \frac{x_b}{n_b}}{\sqrt{V_\beta \bullet \frac{1}{2} \bullet \left(\frac{1}{\tilde{n}} + \frac{1}{\tilde{n}} - \frac{2 \bullet corr \bullet ekl}{e_a e_b} \right)}}$$

Where:

For the test of means:

$$V_\beta = \frac{\sum_{c=1}^{k}\left[x_c^2 - \frac{(x_c)^2}{n_c}\right]}{\sum_{c=1}^{k}(e_c - 1) \bullet \frac{w_c}{n_c}}$$

where

k represents the number of columns tested.

n_c represents the number of observations contributing to column c.

w_c represents the sum of squared weights for the observations contributing to column c.

x_c represents the sum of values in column c, and is calculated as:

$$x_c = \sum_{i=1}^{n_c} x_{ic}$$

x_c^2 represents the sum of squared values in column c, and is calculated as:

$$x_c^2 = \sum_{i=1}^{n_c} x_{ic}^2$$

e_c represents the effective base in column c, and is calculated as:

$$e_c = \frac{n_c^2}{w_c}$$

$$e_\beta = \sum_{c=1}^{k} \frac{n_c^2}{w_c}$$

$\tilde{n}$ represents the harmonic mean and is calculated as

$$\tilde{n} = \frac{k}{\sum_{c=1}^{k} \frac{1}{e_c}}$$

df represents the degrees of freedom and is calculated as:

$$df = \sum_{c=1}^{k} \frac{n_c^2}{w_c} - k$$

corr and *ekl* are the overlap corrections and are calculated as:

$$corr = \frac{\sum_f x_{fa}x_{fb} - \frac{\left(\sum_f x_{fa}\right)\left(\sum_f x_{fb}\right)}{f}}{\sqrt{\left(\sum_f x_{fa}^2 - \frac{\left(\sum_f x_{fa}\right)^2}{f}\right)\left(\sum_f x_{fb}^2 - \frac{\left(\sum_f x_{fb}\right)^2}{f}\right)}}$$

$$ekl = \frac{f^2}{\sum_f w_c^2}$$

where *f* represents the number of observations in overlap contributing.

For the test of proportions:

$$V_\beta = \frac{\left[\frac{\sum_{c=1}^{k} x_c}{\sum_{c=1}^{k} n_c}\right]\left[1 - \frac{\sum_{c=1}^{k} x_c}{\sum_{c=1}^{k} n_c}\right]}{1 - \frac{1}{\sum_{c=1}^{k} \frac{n_c^2}{w_c}}}$$

where

k represents the number of columns tested.

n_c represents the number of observations in the base of column *c*.

w_c represents the number of observations in the squared weights row in column *c*.

x_c represents the number in a row in column *c* (that is, the count).

e_c represents the effective base in column *c*, and is calculated as:

$$e_c = \frac{n_c^2}{w_c}$$

$$e_\beta = \sum_{c=1}^{k} \frac{n_c^2}{w_c}$$

$\tilde{n}$ represents the harmonic mean and is calculated as

$$\tilde{n} = \frac{k}{\sum_{c=1}^{k} \frac{1}{e_c}}$$

df represents the degrees of freedom and is calculated as:

$$df = \sum_{c=1}^{k} \frac{n_c^2}{w_c} - 1$$

corr and *ekl* are the overlap corrections and are calculated as:

$$corr = \frac{n_{fab} - \frac{n_{fa} n_{fb}}{f}}{\sqrt{\left(n_{fa} - \frac{n_{fa}^2}{f}\right)\left(n_{fb} - \frac{n_{fb}^2}{f}\right)}}$$

$$ekl = \frac{f^2}{\sum w_c^2}$$

where *f* represents the number of observations in overlap in the base.

For a more on the theory of overlapping samples, see Kish, *Survey Sampling*.

7.16 References

- Kish, L. *Survey Sampling*. New York: John Wiley and Sons. ISBN 0-471-48900-X.

8 Creating a table of contents

The table of contents is a list of the tables which have been produced in a run. At the end of this chapter there is an example of a table of contents for the sample tables in chapter 11, 'A sample Quantum job' in the Quantum User's Guide Volume 2.

The table of contents program, tabcon, uses intermediate files generated during the Quantum run which created the tables. Normally, Quantum deletes these files at the end of the run. If you want a table of contents you must keep these files. To do this, either use the version of Quantum which does not delete intermediate files, or use the *–k* option as part of your standard Quantum command.

8.1 Using the default layout

The sample table of contents in Figure 8.1 at the end of this chapter shows the default layout and content produced by the table of contents program. To obtain a table of contents, type:

```
tabcon -o tofc
```

This saves the formatted table of contents in a file called tofc.

There are a number of parameters you can use with tabcon to control how the table of contents is formatted and printed. The following table provides a full list of these options.

Option	Explanation
–f *file_name*	Format the table of contents according to specifications in the named format file.
–k	Keep all intermediate files. By default, tabcon deletes the intermediate Quantum files it uses. If you have previously flipped the job for use with Quanvert, and you then run tabcon, you will find that some of the files needed by Quanvert have disappeared. Using *–k* when you create the table of contents prevents it from deleting these files.
–o *file_name*	Write the table of contents to the named file. The default is to display the table of contents on your screen. On systems which allow it, you may use an output redirection parameter instead of *–o*. On Unix systems, for example, you could type: `tabcon > tofc`
–p	Send the formatted table of contents direct to the printer. The default is to display it on your screen. This option is not available under DOS.

Option	Explanation
–u	Underline using proper underline characters rather than hyphens (the suitability of this type of underlining will depend on the capabilities of your printer).
–x	Print a summary of how to use tabcon.

8.2 The format file

The format file determines the layout for the table of contents and the type of information it will contain. It consists of up to four statement types labeled *a, tt, ord* and *sel* respectively. If used, these statements must appear in this order. The format file may contain only one each of the *a*, *ord* and *sel* statements, and up to 32 *tt* statements. It may not contain any blank lines. If you want to use blank lines to improve readability, type a space or press TAB before pressing RETURN.

The a statement

Like an *a* statement in a Quantum program, the *a* statement in a format file determines the overall appearance and requirements for the table of contents. This includes such things as page length, page width, whether to print lines between entries, how to treat base titles, and so on.

The format of the *a* statement is:

a;*options*

where options is a list of keywords from the list below separated by semicolons.

Option	Explanation
basettl	Print table titles starting with Base under the base column rather than under the title column.
date	Print the date.
delim	Draw a line between entries.
dsp	Print a blank line between entries.
n10	Print the first *n10* or *n11* statement in the row axis under the base column.
page	Print the page number of the current table of contents page.
paglen=*n*	Set the page length to *n* lines.
pagwid=*n*	Set the page width to *n* characters.
roman	Print page numbers in roman numerals.

The following options may be preceded by *no* to turn off the defaults which they define:

basettl	*n10*
date	*page*
delim	*roman*
dsp	*section*

The default *a* statement built into tabcon is:

```
a;pagwid=132;paglen=66;nodsp;nodate;page;nodelim;n10;basettl;noroman
```

This produces a table of contents with the following characteristics:

- 132 characters per line and 66 lines per page.
- Single spacing.
- No date.
- Page numbers printed in arabic numerals (1, 2, 3, and so on) on each contents page.
- No lines between entries.
- The first *n10* element in the row axis is printed in the base column of the contents.
- Base titles starting with the word Base are printed in the base column.

The tabcon defaults are over-ridden by those in the format file if one has been specified using the -f parameter or if there is a format file named *tc.def* in the main Quantum directory, your home directory or the project directory. A standard *tc.def* file is distributed with Quantum.

☞ For further information, see section 8.3, 'Naming the format file'.

The tt statement

tt statements define titles to be printed at the top of each page of the table of contents. The format of these statements is:

tt*x**text*

where *x* is a letter defining the position of the text in the line. It may be either *l* for left justification, *c* for centered, or *r* for right justification.

You may define 32 lines of titles. If you do not define any, the table of contents will be labeled with the main title from the Quantum run itself.

The ord statement

The *ord* statement defines the content and layout of each line in the table of contents. Its format is:

ord;*options*

where options is a list of keywords from the list below separated by semicolons.

Option	Explanation
blank=*n*	Print *n* blank spaces between this and the next column. Use this keyword as many times as you need.
base[=*n*]	Print the table base in *n* spaces.
basettl[=*n*]	Print base titles in *n* spaces. This keyword is overridden by *non10* or *nobasettl* on the *a* statement.
page[=*n*]	Print the table's page number in *n* spaces.
table=[*n*]	Print the table number in *n* spaces.
title=[*n*]	Print, in *n* spaces, any titles selected by the *sel* statement.

You define the line layout by typing keywords from this list in the order you want the items to appear in the line.

For example, if you want the line to contain the page number in a field five spaces wide, the table number in a field three characters wide, and the table title, and if you want each field to be separated by three blank spaces, you would type:

```
ord;page=5;blank=3;table=3;blank=3;title
```

If your *ord* statement includes both *title* and *basettl*, but only one is defined with a column width, the other title will be printed in whatever space remains on the line. If both options are present without column widths, tabcon takes the amount of space remaining after other columns have been allocated and divides it equally between the two sets of titles.

The default *ord* statement built into tabcon is:

```
ord;page=4;blank=1;table=5;blank=1;title;blank=1;basettl;blank=1;base=6
```

This prints:

- The page number in four spaces.
- The table number in five spaces.

- Titles whose types are defined on the default *sel* statement.
- The base title.
- The table base in six spaces.

Each column is separated by one space.

If you reduce the page width on the *a* statement but do not define a new *ord* statement, tabcon automatically adjusts the program defaults so that they fit the new page width.

The sel statement

The *sel* statement determines what types of titles are printed. Its format is:

sel;*options*

where options is a list of keywords from the list below separated by semicolons.

Option	Explanation
flt	Print titles under *flt* statements.
high	Print titles from higher dimension axes.
side	Print titles from row axes.
tab	Print titles under the *tab* statement.
top	Print titles from column (breakdown/banner) axes.

The default *sel* statement built into tabcon is:

```
sel;flt;tab;side
```

It prints titles found under *flt* and *tab* statements, and titles defined in row axes.

Titles starting with the word base are normally printed in the base column. You may force tabcon to treat these titles in the same way as other titles by placing *nobasettl* on the *a* statement.

8.3 Naming the format file

tabcon has default settings built into it, and if these are satisfactory you need not use a format file at all. If you want a different layout for some tables, or you want to include more or different information about each table, you need to create a format file. If you call the file *tc.def* and create it in one of a list of directories that tabcon searches automatically, there is no need to name the file on the tabcon command line. Where you create *tc.def* depends on which tables it applies to:

- If you want all tables of contents to look the same (for example, if you have a house style), then create the file in the main Quantum include subdirectory:

  DOS %qthome%\include\tc.def

  Unix $QTHOME/include/tc.def

- If you have a style which you always use for your jobs, create the file in your home directory.
- If the layout or content is unique to one job, create the file in the project directory.
- You can also create format files with other names and in other directories. This is particularly useful if you do work for a number of clients who each have different requirements for their tables of contents. If you create a separate file for each client and keep it in, say, your home directory, you can call up the file for an individual client by naming it on the tabcon command line by using –f. For example:

  ```
  tabcon -f $HOME/tc.ben -o tofc
  ```

Since it allows many format files to exist, tabcon always searches for them in a fixed order:

1. Internal defaults
2. Main Quantum include directory
3. Your home directory
4. Project directory
5. File named with –f

tabcon reports the names of the format files it has used and the order in which it has used them as it runs.

In the example below, tabcon has used the installation format file and the project format file only:

```
Tabcon stage 1, version 9.1
Created Tue Oct 27 11:49:20 GMT 1992
Using formats from formatfile /qtime/v5d.1/include/tc.def
Using formats from formatfile tc.def
No errors found in formatfiles
Tabcon stage 2, version 9.1
Created Tue Oct 27 11:49:20 GMT 1992
```

Sometimes a file at a lower level overrides the same file at a higher level, at other times the information in the lower level file is additional to that in the same file at the higher level.

The table below shows what to expect for each format statement:

a New keywords at lower levels are additional to those at higher levels. If the same keyword is present with different values at different levels, the lowest level overrides the higher levels.

tt Lowest level is additional to higher levels.

ord Lowest level overrides higher levels.

sel Lowest level overrides higher levels.

26 May 1988

VISITOR SURVEY - BRITISH MUSEUM (NATURAL HISTORY)

Figure 8.1 Sample Table of Contents

9 Laser printed tables with PostScript

If you have a laser printer which recognizes the PostScript language, you can produce Quantum tables in this format by running a postprocessor, pstab, after the standard Quantum run. This provides you with:

- More flexible formatting capabilities for row text and column headings.
- Access to a wide range of different fonts.
- The ability to print pound signs.
- The ability to print logos.

The Quantum commands required for these facilities are described below.

The fonts which are used for printing Quantum tables on a laser printer are usually proportionally spaced. This means that each character is printed in the minimum amount of space required; so the letter ‘i’ takes up less space on a line than the letter ‘m’. In axes where you define the column headings with *g* statements, or where you lay out the row text in a particular way, the use of proportional fonts means that the printed output will not look the same as the layouts in your Quantum program. It is therefore necessary to insert some additional characters in these texts to define how they should appear on the laser-printed output.

✎ The *hitch* and *squeeze* facility is currently not available for PostScript tables.

9.1 Printing output with pstab

Quick Reference

To generate tables in PostScript format, run a standard Quantum tabulation job and then type:

pstab [**–x**] [**–d**] [**–s**] [**–f** *font_file*] [**–o** *output_file*] [**–p** *printer*] [**–t** *tabcon_file*]

To print output on a PostScript printer, first run the job as usual to create a tables file. Then run the program **pstab** to format and print the tables on the laser printer:

pstab [**–x**] [**–d**] [**–s**] [**–f** *font_file*] [**–o** *output_file*] [**–p** *printer*] [**–t** *tabcon_file*]

The following table provides details of the parameters.

Parameter	Explanation
–x	Print a summary of usage and exit.
–d	Stamp the word 'draft' across the tables.
–s	Translate $ signs in the file to pounds signs in the printed output.
–f *font_file*	Take fonts and logo definitions from the named file.

☞ For further information, see section 9.7, 'Fonts and logos'.

–o *output_file*	Save the PostScript output in the named file. The file may then be printed on a PostScript printer using the *lpr* command.
–p *printer*	Print the output on the named printer. This option uses the Unix lpr command to queue the print job. So this option does not work if the lpr command is not available on your system. This option is not available when running Quantum under DOS.
–t *tabcon_file*	Take the format for the table of contents from the named file. pstab automatically generates a table of contents file. This will be printed after the last table.

✎ pstab uses many of the intermediate files produced by the normal Quantum run. To keep these files for use with pstab, either use the version of Quantum which does not delete intermediate files or include the option –k on the standard Quantum command line. Also, do not clean the directory in any way after the run has finished.

9.2 Column headings

Quick Reference

To determine the justification of column headings above columns, use the characters:

^ Center the following text.

~ End of text block.

} Right-justify the following text.

{ Left-justify the following text.

! Optional break point in text (replace with space).

| Optional break point in text (replace with nothing).

Column headings that are generated automatically without the use of *g* and *p* statements are laid out so that the table extends across the full width of the page. Long texts are folded as necessary to create multi-line headings right-justified above the numbers in each column. You can use the ! and | characters to force breakpoints in element texts used for column headings. However, the ! and | characters only cause a breakpoint in PostScript output when the element text is too wide for the column, whereas they always cause a breakpoint in the standard output.

Column texts defined on *g* statements may be left-justified, right-justified or centered within a column. The position of a text in a column is determined by the use of the characters { }, ^ and ~ in the text where you would normally have a | or a blank space between columns:

} Start right-justified column text.

{ Start left-justified text.

^ Start centered text.

~ End of text block, for example, a column heading. If this is omitted, the column is assumed to end at the next { } or ^ character, whichever is the sooner. These special characters are the defaults. You may define your own characters which will be used in place of any or all of the defaults.

The characters { } and ^ mark the start of a column and ~ marks the end. Any text between these characters is justified according to the character which precedes them.

For example:

```
l color
col 32;Base;Red;Green;Yellow
g}     Base~ }    Red~ }  Green~ } Yellow~
p         x         x         x         x
```

requests that the column headings should be right-justified in the space between the } and ~ characters — that is, the right-most character of the column heading should be printed immediately above the right-most digit in that column, as specified by the *p* statement. Without the ~, the text would be right-justified between the } signs before and after the column text.

Similarly,

```
l color
col 32;Base;Red;Green;Yellow
g{     Base{     Red{   Green{  Yellow~
p         x         x         x         x
```

causes column headings to be left-justified above each column of numbers, whereas:

```
l color
col 32;Base;Red;Green;Yellow
g^           Color Preferred          ~
g}     Base^     Red^   Green^  Yellow~
p         x         x         x         x
```

indicates that the Base text should be right-justified while the remaining column texts should be printed centrally above each column. The overall axis heading will be centered above the column headings. Right-justified text above columns generally looks best, particularly for the lowest level of headings (the colors in our examples), so you'll probably use the } character in most of your headings.

You will notice from these examples that the column heading is set out exactly as it has always been; the only difference is the presence of the } and ~ characters at the start of each column where you would normally have typed a space. The position of the special characters is important since they determine the space in which the column headings will be justified. In our examples we've placed the special characters immediately to the left of the cell marker on the *p* statement so that the text is justified between the end of the previous column and the end of the current column. If you place the special characters in other positions, the text will still be justified between those characters even if this means that it no longer lines up with the columns themselves.

If you type text on *g* statements but omit the layout control characters, the tables will contain blank lines instead of column headings. Control characters do not affect your tables when they are formatted for printing on a non-Postscript printer; that is when you run Quantum but not pstab. The column headings retain their correct layout with the special characters being replaced by spaces.

In tables with a large number of column headings, it may occasionally happen that there is not enough room on the page to fit the individual column texts in the space allocated to each column. For instance, where you ask for a long word to be centered across a column, the column may not be wide enough to print the text in the font you are using. If this happens, Quantum will squash the characters until the text fits in the space available.

User-definable PostScript characters

Quick Reference

To define your own special characters create a file called qtform containing one line of just six characters (blank counts as a character). The first character is the replacement for the default ~ character, the second character is the replacement for the default ^ character, and so on.

Alternatively, you may define the six characters using the variable QTFORM.

The characters:

~ ^ { }

on *g* statements, and the characters:

| !

in element texts determine how the column headings will be laid out on the table. These characters are defaults which you may change if you wish. There are several ways of doing this, but in all cases you must always define all six special characters, in the order they are shown here, even if some of them are the same as the defaults.

Replacement of special characters may be defined globally for all jobs, or individually for particular jobs by placing the new characters in a file in one of the following locations:

DOS Project-specific defaults in qtform in the project directory.

Unix Installation defaults in $QTHOME/include/qtform or project-specific defaults in qtform in the project directory.

If you wish to define your own personal defaults, you may do so using the environment variable QTFORM.

Quantum searches first for the environment variable, then for qtform at the project level, and finally for qtform at installation level, stopping at whichever one it finds. If none of these exist, Quantum uses the default characters.

To define your own special characters, create a qtform file containing one line of just six characters:

char1 char2 char3 char4 char5 char6

where *char1* is the replacement for the ~, *char2* is the replacement for the ^, and so on. You may include blanks (spaces) in the list of characters, but these mean that the special characters they replace will have no special meaning at all to pstab.

The example below replaces the curly braces and the pipe symbol with a colon, a semicolon and an 'at' sign respectively. All other characters are unchanged:

```
~^:;@!
```

The next example contains blanks for the { and } characters. If pstab reads these characters on a *g* statement it will print them as part of the column heading rather than reading them as left and right justification symbols:

```
~^  |!
```

The final example illustrates how to prevent the characters | and ! from having special meanings in element texts. You might wish to do this if you need to print these characters as part of the element text:

```
~^{}
```

✎ This change affects all axes in the run, not just the axis in which these characters form part of the element text.

If you wish to use the environment variable QTFORM rather than creating a file, just list the six characters as the value of the variable in the usual way. If the list of characters contains blanks you must enclose it in double quotes.

If the file or character list contains more or less than six characters Quantum issues an error message to this effect as the tables are formatted and uses the built-in defaults instead. If this is your only error, you may correct the value of $QTFORM and rerun the job with `quantum -o`.

9.3 Underlining column headings

Quick Reference

To underline text, type the following characters on a *g* statement:

– or **_**	to print a single thin line
=	to print a single thick line

Lines drawn with hyphens or underscores in column headings may still be printed. Thin lines may be drawn with – or _ characters, and a single thick line with the = character.

9.4 Text alignment in row axes

Quick Reference

To determine the justification of words or sections in row texts, use the characters:

}	right-justify the following text
{	left-justify the following text
^	center the following text
~	end of text block

and place the statement:

n03#& *character*

at the start of the axis, where *character* is one of the special characters shown above.

{, }, ^ and ~ may also be used with row texts which need to be laid out in a particular format. For instance, where the row texts are scores you may wish to print the factors directly underneath one another rather than just in the next free column, thus:

```
Excellent     (+2)
Very Good     (+1)
Average       (0)
Poor          (–1)
Very Bad      (–2)
```

In order to have row texts aligned in columns you need to start the axis with an *n03* statement of the form:

n03#& *X*

where *X* is one of the special characters { } or ^. The number of spaces between the & and the special formatting character depends on where you want the aligned column to start. For instance, if you want text to be aligned on column 20 you would type 19 spaces and then the formatting character in the 20th position.

Then in the individual elements you place a ~ immediately before the first character which is to be part of the aligned column (for example, the scores). Long texts before the ~ will be folded, but texts after it will be squashed if they will not fit within the remaining space in the side text.

Here is an example which lines up the first character of each score in column 20. Notice the { for left-justification on the *n03#&* statement and the ~ on the elements:

```
n03#&                    {
n10Base
n01Excellent ~(+2);fac=2;c=c237'1'
n01Very Good ~(+1);fac=1;c=c237'2'
n01Average ~(0);fac=0;c=c237'3'
n01Poor ~(-1);fac=-1;c=c237'4'
n01Very Bad ~(-2);fac=-2;c=c237'5'
```

Since this is a simple axis it could have been written using a *col* statement:

```
n03#&                    {
col 237;Base;Excellent ~(+2);%fac=2-1;Very Good ~(+1);
+Average ~(0);Poor ~(-1);Very Bad ~(-2)
```

9.5 Character sizes and fonts for titles

Quick Reference

To define the percentage by which titles should be enlarged or reduced in size, type:

#s*number*

at the start of the text on the *tt* statements. The standard size is 100 (i.e., 100%).

To define the font in which the a title should be printed, type:

#f*number*

at the start of the text on the *tt* statements. *number* is the number of the font you wish to use as it is defined in the font table.

Table titles defined on different parts of the table, for example, a *ttl* followed by a *ttc*, will be printed on the same line. A new-line will be produced when a subsequent title is defined for the same position as a previous one, for example, a *ttr* followed by a *ttr*. The texts are not checked as to whether there is sufficient space to fit them in. Therefore, the user must check whether this is likely and control the production of a new-line by means of blank *tt* statements.

All numbers and texts in a table are printed in a standard size using the fonts you request, as explained below. Text defined on *tt* and *n03* statements may be enlarged or reduced in size by preceding the text with the notation:

#s*n*

where *n* is a number defining in percentage terms the amount of enlargement or reduction required. The table title on the sample landscape table was generated by the statement:

```
ttc#s150VISITOR STUDY - BRITISH MUSEUM (NATURAL HISTORY)
```

The #s150 indicates that it should be printed 150% as large as the rest of the table text. A similar *n03* statement might read:

```
n03#s75Text at three-quarters the standard size
```

Quantum has a standard set of fonts that it uses for printing tables. The default is Helvetica. These fonts are declared in a font table, and each font is represented by a number based on the position of that font in the table. The first font in the table is font 0.

You might wish to print titles in a different font; for example, Helvetica Bold or Helvetica-Oblique (italic). To do this, type:

f*number*

in front of the text on the *tt* or *n03* statement.

For example:

```
ttc#f2General Election Survey
```

tells Quantum to print the title in font 2. If you use Quantum's standard fonts this is Helvetica Bold.

☞ For more information on fonts and the font table, see section 9.7, 'Fonts and logos'

9.6 Boxes in tables

Quick Reference

To suppress the automatic table border, type:

#nopagebox

in the font file.

To print boxes around sections of a table, type:

boxs	Start a box.
boxe	End a box.
boxl	Draw a line between each column or row within the box.
boxg	Start boxes above column headings on *g* statements (used with *boxs*).

Tables printed on the PostScript printer are automatically enclosed in a border. If you want unboxed tables, place the statement **#nopagebox** in the font file (specified with the –f option on the command line).

Additional statements have been added to Quantum to enable you to place boxes around selected cells in a table. For example, it is possible to print a table in which rows 3 to 5 and columns 1 to 4 are enclosed in a box.

The statements associated with boxes are:

boxs Start a box.

boxe End a box.

boxl Draw a line between each row or column within the box. In row axes this generates a horizontal line; in column axes a vertical line is drawn.

boxg Start boxes above rather than below column headings defined with *g* statements. This keyword requires *boxs* to be specified as well.

Boxes will only be drawn if these statements are present in both the row and column axes; if they are present in only one axis in a table, no boxes will be drawn. Additionally, a box extends the full width of the column rather than enclosing just the numbers themselves. Any *box* statements in illegal or irrelevant positions are silently ignored.

Here is an example:

```
tab rowax colax
l rowax
n10Base
n01Brand A
boxs
n01Brand B
n01Brand C
boxe
l colax
n10Base
n01Red
boxs
n01Blue
n01Green
boxe
```

This generates a table with 16 cells of which four are enclosed in a box, thus:

```
          Base      Red      Blue     Green
Base         x        x         x         x
Brand A      x        x         x         x
                         ------------------
Brand B      x        x  |      x         x |
Brand C      x        x  |      x         x |
                         ------------------
```

If you wanted to draw a horizontal line between Brand B and Brand C and a vertical line between Blue and Green, thereby giving the appearance of a boxed table, you must insert *boxl* statements in the appropriate places as follows:

```
l rowax
.
boxs
n01Brand B
boxl
n01Brand C
boxe
.
l colax
boxs
n01Blue
boxl
n01Green
boxe
```

This generates the following table:

```
          Base      Red      Blue     Green
Base         x        x         x         x
Brand A      x        x         x         x
                          ------------------
Brand B      x        x  |      x   |     x |
                         |----------|-------|
Brand C      x        x  |      x   |     x |
                          ------------------
```

9.7 Fonts and logos

Quick Reference

To introduce a list of fonts for the tables, type:

#fonttable

in the font file. Follow this with the names of the fonts to be used, one per line. Terminate the list with:

#endfonttable

To scale a font up or down from its default size, enter the font name followed by a space and the scaling factor.

The PostScript laser printer offers a wide variety of typefaces and type styles. The *font=* option on the *a* statement determines which parts of the table are printed in which typeface — for example, you may decide to print absolutes in the standard typeface, percentages in italic and the run title in bold.

Quantum provides a standard set of fonts which will be used if you use *font=* but do not name the fonts to use. These are:

font=0 : Helvetica
font=1 : *Helvetica-Oblique*
font=2 : **Helvetica-Bold**
font=3 : ***Helvetica-BoldOblique***
font=4 : Times-Roman
font=5 : *Times-Italic*
font=6 : **Times-Bold**
font=7 : ***Times-BoldItalic***

The first font in the list is the default font which will be used if no other fonts are defined, or if an invalid *font=* option is given (*font=8* when no font is defined for position 8), or if you misspell a font name.

You may also choose to print your tables using an entirely different font to the default — for example in the AvantGarde or Souvenir font. A full list of the fonts available may be obtained from your printer supplier. To print in a different font, create a font file starting with a line containing the word **#fonttable** and then list the names of the fonts you wish to use, one font name per line.

The list must be terminated with a line containing the word **#endfonttable**. If you want to print your tables using the Avant-Garde font, for example, your font file might be:

```
#fonttable
AvantGarde-Book
AvantGarde-Book
AvantGarde-Demi
AvantGarde-DemiOblique
AvantGarde-BookOblique
#endfonttable
```

Here, the first font corresponds to *font=0* (the default font), the second font corresponds to *font=1*, and so on. If the *font=* option is:

```
font=(a=2,tab=2,pc=3,numb=1,page=4,date=4,type=4)
```

titles following the *a* and *tab* statements will be printed in font 2 (AvantGarde-Demi), percentages will be printed in font 3 (AvantGarde-DemiOblique), all other numbers will be printed in font 1 (AvantGarde-Book), and the page number, date and output type will be printed in font 4 (AvantGarde-BookOblique). All other parts of the table will be printed in font 0 (AvantGarde-Book).

If you find that the character sizes used in PostScript tables are generally too small or too large, you may alter the default size by defining the font with a scaling factor in the font table section of the font file.

Scaling factors are proportions written as decimal values, so to make characters twice as large you would enter the scaling factor as 2.0. To make the characters one and a half times their default size you need a factor of 1.5, whereas to make the characters 95% of their current size you need a factor of 0.95.

To define a scaling factor, just type the font name followed by a space and the scaling factor in the font table. Here is an example that increases the standard character size for the Helvetica font by 35% and the character size for the Times-Roman font by 50%:

```
#fonttable
Helvetica 1.35
Times-Roman 1.5
#endfonttable
```

If you run tables using this font table, Quantum uses Helvetica as font 0 and Times-Roman as font 1. If you have any font statements that refer to other font numbers you will see error messages when Quantum prints your PostScript tables.

Logos are defined using the PostScript language and are included in the font file after the font definitions. They are printed at the same time as the tables. If you wish to include a logo on your tables, contact your SPSS MR support representative.

9.8 Positioning tables on the page

Quick Reference

To print small tables left-justified on the page, type:

#tableleft

in the font file.

Because the laser printer uses proportionally-spaced fonts and prints in a smaller type size than a line printer, tables which would normally fill a page of line printer paper will only partially fill the page when the table is laser printed. When the column headings are defined on *g* statements and the table is narrower than the page width, Quantum will print it centrally in the width of the page. If you want the table to be left-justified on the page, enter the command:

#tableleft

in the font file containing the font definitions.

You may also scale the table as a whole up to a larger type size by defining smaller values for *pagwid=* and *paglen=* on the *a* statement. For example, when printed using the default type size, the sample table occupies half a page. We have scaled it to fill the whole page by defining the page to be just a little larger than the table itself — that is, *pagwid=115; paglen=50.*

If you are printing more than one table, you will need to take the width of the widest table and the length of the longest one as guidelines for the overall length and width required. Since it is not possible to gauge exactly how many lines the table will occupy when it is laser printed, it is best to increase your line counts by a small amount as in the example above.

9.9 Tables without a table of contents

pstab automatically creates a table of contents which it prints, starting on a new page, at the end of the tables.

Under Unix you can suppress the table of contents if you do not want it, by running qout and q2ps rather than pstab:

```
qout -p | q2ps -f format_file -o tables_file
```

9.10 Font encoding in PostScript tables

Font encoding is a method of defining and printing characters that are not part of the standard ASCII character set. You can think of it as a translation service whereby the printer reads a character from the tables or table of contents, looks it up in a translation table and prints the corresponding translation character.

Quantum comes with one encoding scheme already set up, which allows you to print characters from the Microsoft Multilingual (Latin 1) character set (in DOS this is known as code page 850), but you can define other translations if you wish.

Each font encoding scheme is held in a separate file in the QTHOME include directory. The files are called *name*.fen, where *name* is any name you choose (it is sensible to choose a name that reflects the name of the character set or code page). The name of the Microsoft Multilingual encoding file is cp850.fen.

Requesting font encoding

To request font encoding, include the option:

–e *encoding_scheme*

in your pstab command. Where *encoding_scheme* is the name of the encoding file without the .fen extension, so to request encoding using the Microsoft code page 850 scheme you would use the option `-e cp850`.

Defining your own encoding sets

If you are using a different character set you may create your own encoding files. For example, if you are working on a PC that uses the Slavic character set (code page 852) you may wish to create an encoding file for that character set so that your printed tables match those you display on your screen.

Creating encoding files requires some knowledge of PostScript, but the general procedure is as follows.

First, make a list of the characters in your character set that print differently to the way they appear on your screen (mostly, these characters will appear as blanks in the printed output). These are the characters for which translations are required. Normally these will be the characters 127 to 255 which form the extended ASCII character set: characters 0 to 31 are not used for keyboard characters, and characters 32 to 126 (the standard ASCII character set) are common to all code pages.

Create a new encoding file with an appropriate name and a .fen extension. You could use cp850.fen as a template.

For each character requiring translation, write an Encoding statement that defines the character's decimal value and the name of the PostScript character or symbol you want to print in its place. You'll find a list of character and symbol names for all standard PostScript fonts in the back of the Adobe® PostScript Reference Manual.

An alternative to creating an encoding file is to place all the encoding information in the Quantum job's PostScript format file as user-defined code, as described in the next section.

9.11 Personalized code in the PostScript format file

You may now include any PostScript code you like in the format file for pstab and pstabcon. Type:

#postscript

before the first line of your code and:

#endpostscript

after the last line. You could use this feature for declaring your own encodefont routine for font encoding.

VISITOR SURVEY - BRITISH MUSEUM (NATURAL HISTORY)

Absolutes/col percents
1 Sep 1999

How did you find your way round the Museum?
Base: All Leaving Museum

		Sex		Age				Completed Full Time Education		Visited Museum Before	
	TOTAL	Male	Female	11-20	21-34	35-54	55+	Yes	No	Yes	No
Base	301	171	130	57	152	71	21	237	64	156	145
Signposting	127	71	56	19	66	31	11	104	23	59	68
	42.2%	*41.5%*	*43.1%*	*33.3%*	*43.4%*	*43.7%*	*52.4%*	*43.9%*	*35.9%*	*37.8%*	*46.9%*
Wandered	120	76	44	28	58	25	9	93	27	52	68
	39.9%	*44.4%*	*33.8%*	*49.1%*	*38.2%*	*35.2%*	*42.9%*	*39.2%*	*42.2%*	*33.3%*	*46.9%*
Guidebook	22	6	16	2	14	4	2	20	2	9	13
	7.3%	*3.5%*	*12.3%*	*3.5%*	*9.2%*	*5.6%*	*9.5%*	*8.4%*	*3.1%*	*5.8%*	*9.0%*
Attendant	11	3	8	2	3	5	1	8	3	10	1
	3.7%	*1.8%*	*6.2%*	*3.5%*	*2.0%*	*7.0%*	*4.8%*	*3.4%*	*4.7%*	*6.4%*	*0.7%*
Brief description of route taken	4	2	2	1	1	1	1	3	1	2	2
	1.3%	*1.2%*	*1.5%*	*1.8%*	*0.7%*	*1.4%*	*4.8%*	*1.3%*	*1.6%*	*1.3%*	*1.4%*
With difficulty	16	10	6	3	7	5	1	12	4	11	5
	5.3%	*5.8%*	*4.6%*	*5.3%*	*4.6%*	*7.0%*	*4.8%*	*5.1%*	*6.3%*	*7.1%*	*3.4%*
Gallery plan	35	23	12	7	20	8	-	26	9	21	14
	11.6%	*13.5%*	*9.2%*	*12.3%*	*13.2%*	*11.3%*	*-*	*11.0%*	*14.1%*	*13.5%*	*9.7%*
With some who knew/already knew	30	14	16	7	13	8	2	23	7	28	2
	10.0%	*8.2%*	*12.3%*	*12.3%*	*8.6%*	*11.3%*	*9.5%*	*9.7%*	*10.9%*	*17.9%*	*1.4%*
DK/NA	3	2	1	-	3	-	-	3	-	1	2
	1.0%	*1.2%*	*0.8%*	*-*	*2.0%*	*-*	*-*	*1.3%*	*-*	*0.6%*	*1.4%*

A Options in the tabulation section

Keywords preceded by an asterisk may be used with the prefix *no* to switch off a global requirement for a specific table or element only. If the keyword is followed by an equals sign it is omitted when *no* is used (for example, *inc*= becomes *noinc*).

	Option	a/flt/tab/sectbeg	l	n/col/val/bit/fld
*	**acr100**	Yes	—	—
	anlev=	Yes	Yes	—
*	**axcount**	*a* only	—	—
	axreq=	*a* only	Yes	—
*	**axtt**	Yes	—	—
	baft	Yes	—	—
	base	—	—	Yes
	byrows	—	Yes	—
	c=	Yes	Yes	*n* only
	celllev=	*tab* only	—	—
	clear=	—	Yes	—
	clevel=	Yes	—	—
	coltxt=	—	—	*n03* only
	colwid=	Yes	Yes	Yes
	csort=	Yes	—	—
*	**date**	Yes	—	—
	dec=	Yes	—	Yes
	decp=	Yes	—	—
*	**dp**	*a* only	—	Yes
*	**dsp**	Yes	Yes	Yes
	dummy	—	—	*n01* and *n15* only
	effbase	—	—	Yes
	endnet	—	—	Yes
	endsort	—	—	Yes
	ex=	—	—	Yes
*	**exportmp**	Yes	—	—
*	**fac=**	—	—	Yes
*	**figbracket**	—	Yes	Yes
*	**figchar=**	—	Yes	Yes
*	**figpre=**	—	Yes	Yes
*	**figpost=**	—	Yes	Yes
	flt=	not *a*	—	—
*	**flush**	Yes	—	—

	Option	a/flt/tab/sectbeg	l	n/col/val/bit/fld
	font	Yes	—	—
*	**graph**	Yes	—	—
	group=	—	—	Yes
	hd=	—	Yes	not *n*
	hdlev=	—	—	*n23* only
	hdpos=	—	—	*n23* only
	hold=	—	—	Yes
	hitch=	Yes	—	—
	hug=	—	—	Yes
	id=	*tab* only	—	Yes
*	**ignorezeros**	—	—	*n07* only
*	**inc=**	Yes	Yes	Yes
	inctext=	Yes	Yes	Yes
	indent=	Yes	Yes	Yes
	keep=	*tab* only	—	bases only
	lang=	*a* and *tab* only	—	—
	levbase	—	—	Yes
	linesaft	Yes	—	—
	linesbef	Yes	—	—
	manipz	Yes	—	—
	maxim	Yes	—	Yes
	means	Yes	—	Yes
	median	Yes	—	Yes
	medint=	Yes	—	—
	minbase=	Yes	—	—
	minim	Yes	—	Yes
	missing=	Yes	Yes	Yes
*	**missingincs**	*a* only	—	—
	missingval	—	—	Yes
*	**netsm**	Yes	Yes	—
*	**netsort=**	*a* only	Yes	—
	nocol	—	—	Yes
	noexport	—	—	Yes
	nohigh	—	—	Yes
	nooverlapfoot	Yes	—	—
	noprint	Yes	—	—
	noround	—	—	Yes
	norow	—	—	Yes
	nosort	—	—	Yes

	Option	a/flt/tab/sectbeg	l	n/col/val/bit/fld
	notauto	Yes	—	—
	notstat	—	Yes	Yes
	notbl	Yes	—	—
*	**nsw**	*a* only	—	—
	ntot / nontot	—	—	Yes
	numcode=	—	Yes	—
*	**nz**	Yes	Yes	Yes
*	**nzcol**	Yes	—	—
*	**nzrow**	Yes	—	—
	op=	Yes	—	Yes
	overlap	Yes	—	—
*	**page**	Yes	—	—
	paglen=	Yes	—	—
	pagwid=	Yes	—	—
*	**pc**	Yes	—	—
	pcpos=	Yes	—	Yes
*	**pcsort**	Yes	—	—
*	**pczerona**	Yes	—	—
	percentile=	Yes	—	Yes
*	**physpag**	*a* only	—	—
*	**printz**	Yes	—	—
	rej=	—	—	Yes
*	**rinc**	Yes	—	—
*	**round**	Yes	—	—
*	**rsort**	Yes	—	—
*	**scale=**	Yes	—	Yes
	side=	Yes	—	—
	smallbase=	Yes	—	—
	smbase=	Yes	—	—
*	**smcol**	Yes	—	—
	smflag=	Yes	—	—
*	**smrow**	Yes	—	—
	smsup+	—	—	Yes
	smsupa=	Yes	—	Yes
	smsupp=	Yes	—	Yes
	smsupt=	Yes	—	Yes
*	**smtot=**	Yes	—	—
*	**sort**	Yes	Yes	Yes
	sortcol	—	—	Yes

	Option	a/flt/tab/sectbeg	l	n/col/val/bit/fld
	spechar=	Yes	—	—
	squeeze=	Yes	—	—
	stat=	Yes	Yes	—
	subsort	—	—	Yes
*	**summary**	—	Yes	Yes
	supp	—	—	Yes
*	**tabcent**	Yes	—	—
*	**title**	Yes	—	—
*	**topc**	Yes	—	—
	toptext=	—	—	Yes
*	**tstat**	Yes	Yes	Yes
	tstat	*tt*s	*tt*s	—
*	**tstatdebug**	*a* only	—	—
	ttbeg=	Yes	—	—
	ttend=	Yes	—	—
	ttord=	Yes	—	—
	tx=	—	—	not *n*
*	**type**	Yes	—	—
	unl	*tt*s	*tt*s	Yes
	uplev=	—	Yes	—
*	**useeffbase**	*a* only	—	—
	wm=	Yes	—	Yes
*	**wmerrors**	*a* only	—	—

Index

This index covers all four volumes of the Quantum User's Guide. The page references consist of the volume number followed by the page number; for example 2-6 is page 6 of Volume 2, 3-166 is page 166 of Volume 3, and so on.

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z