# The Application Guide

Version 3.0X March 1998

# *The Application Guide*

**Version 3.0X March 1998**

# Table of Contents

*Chapters*



*Technical Appendices*

# 1

*The design philosophy of QueryCalc*

# An Introduction

## A Simple but Powerful Idea

At the heart of every HP3000 lies a very simple idea. The HP3000's database, IMAGE, was designed to be no more than an electronic filing cabinet, and it was orginally meant to be just as easy to use. Somehow, over the years, this simple idea has been confused by a river of jargon. From this point on, we would like you to forget the stack of manuals you have on the shelf describing IMAGE and the intricacies of DBGETs, DBFINDs and DBPUTs. None of that information is particularly necessary to understanding why you would want to use IMAGE databases, nor how you might use existing databases to your advantage.

You bought your HP3000 to replace your steel filing cabinets with electronic ones. Now you should be able to get the information out at least as easily as you would have from the old filing cabinets. QueryCalc was designed to allow you to create management reports with ease, without a great deal of fuss or memorization. QueryCalc combines two very basic office ideas, spreadsheets and filing cabinets, into a powerful tool. All that's changed is that now the spreadsheets and filing cabinets are electronic.

This ease of use doesn't mean that anything's hidden from you. Quite the opposite, in fact. If you're ever really going to understand what's going on, you must be able to see what's in your databases. QueryCalc opens IMAGE up and makes your information visible to you.

The success of spreadsheets on computers has been historic, especially on personal computers. A spreadsheet encourages a *"What if..."* form of analysis. But this is not the primary manner by which you are going to use QueryCalc. QueryCalc allows you to extract information directly from the databases resident on your HP3000 and manipulate that information in the ways you require. If the computer is ever going to pay for itself, at least in the way you imagined it would before you bought it, this is where it's going to do it.

The single feature that made the electronic spreadsheet a success was *visible calculation.* You can see the relationships between the numbers on the page and you can change them. This feature alone accounts for much of the poplarity of spreadsheets on personal computers.

QueryCalc is a 26-page, true three-dimensional spreadsheet. And QueryCalc can be used simply as a spreadsheet. Operations may occur over individual cells scattered over multiple pages, columns, rows, vectors through pages or as complete cubes. QueryCalc has all of the features and behaviors you would expect from the very best of the personal computer spreadsheets. But that's only the surface; QueryCalc is much more than that.

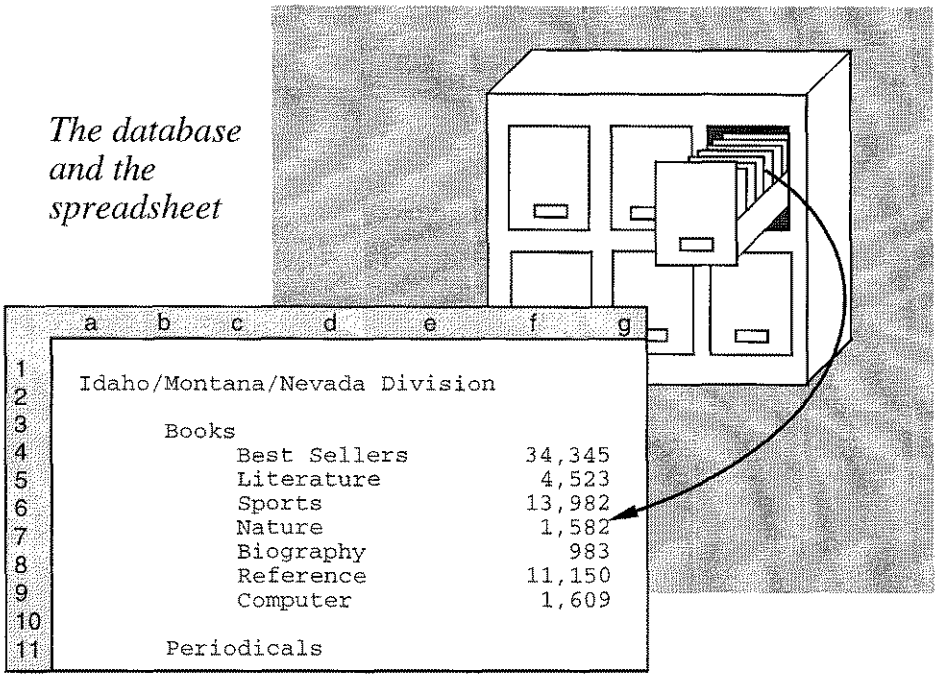
## The Spreadsheet as a Report Writer

A normal spreadsheet is a large, flat plane of rows and columns. If you wanted to enter the sales figures for January, February and March, you would have to go off to the side, add the sales receipts yourself and type the answers in. Or you might use some form of a program to get that same information out of your IMAGE databases and transfer it into your PC by down-loading an HP3000 file to a PC. Doing either procedure is cumbersome, tedious work where nothing is immediately intuitive or transparent. And it will be just as much trouble to do it next month when a newly calculated, updated report is needed again.

Nevertheless, it's obvious that there is a strong pressure to do something exactly like that. Probably someone in your organization is already using one or the other procedures, trying to get information out of your IMAGE databases into a form that he or she can understand and is familiar with on a personal computer.

## Power Software on the HP3000

QueryCalc is different. Where a normal spreadsheet is thin and flat, QueryCalc has depth because of its links into IMAGE. Every cell in QueryCalc's 26 pages can be a database inquiry question into any one of 10 IMAGE databases. Defining sums and relationships between the information extracted from the databases is simple because of the spreadsheet nature of the report. Text formatting and reorganization take on word processing-like attributes. QueryCalc is a "what-you-see-is-what-you-get" *(WYSIWYG)* report writer. No trial formats or trial compilations are necessary. You know what the output will look like before you print it because it's right there on the screen.




QueryCalc was designed so that everything is visible while you are putting a report together. The report you create, while you are creating it, is like putty. If you don't like what you've done, you can change it immediately. The data you retrieve from the IMAGE databases is completely open for your inspection. Reports can often be assembled 5 to 20 times faster than they could be using any other method. And yet the entire process is simple enough that you can learn to use it by mimicking someone else's report. QueryCalc can then be scheduled to run in the middle of the night, gathering and building your standard reports, week after week, without any further operator intervention.

## How a Typical Report is Built

Consider the report on the following page. QueryCalc has a variety of calendar functions so that date and time equations may be placed in cells, as they have been in the upper left hand corner of this report. QueryCalc readily allows you to transmit any escape sequences necessary for your particular printer to print in bold, italics or enlarged print, as was done for the title headings shown.

The text that fills the left hand column of the report is nothing more complicated than text labels, typed in by the person composing the report as he or she wished them to be.

But it is the right hand column of numbers that defines the power of QueryCalc. The numbers in the right hand column were not simply typed in, as would normally be the case, but are the results of database query questions which reside in their respective cells. The displayed results came directly out of IMAGE database(s) into the spreadsheet, were summed and formatted and are now ready to be printed.

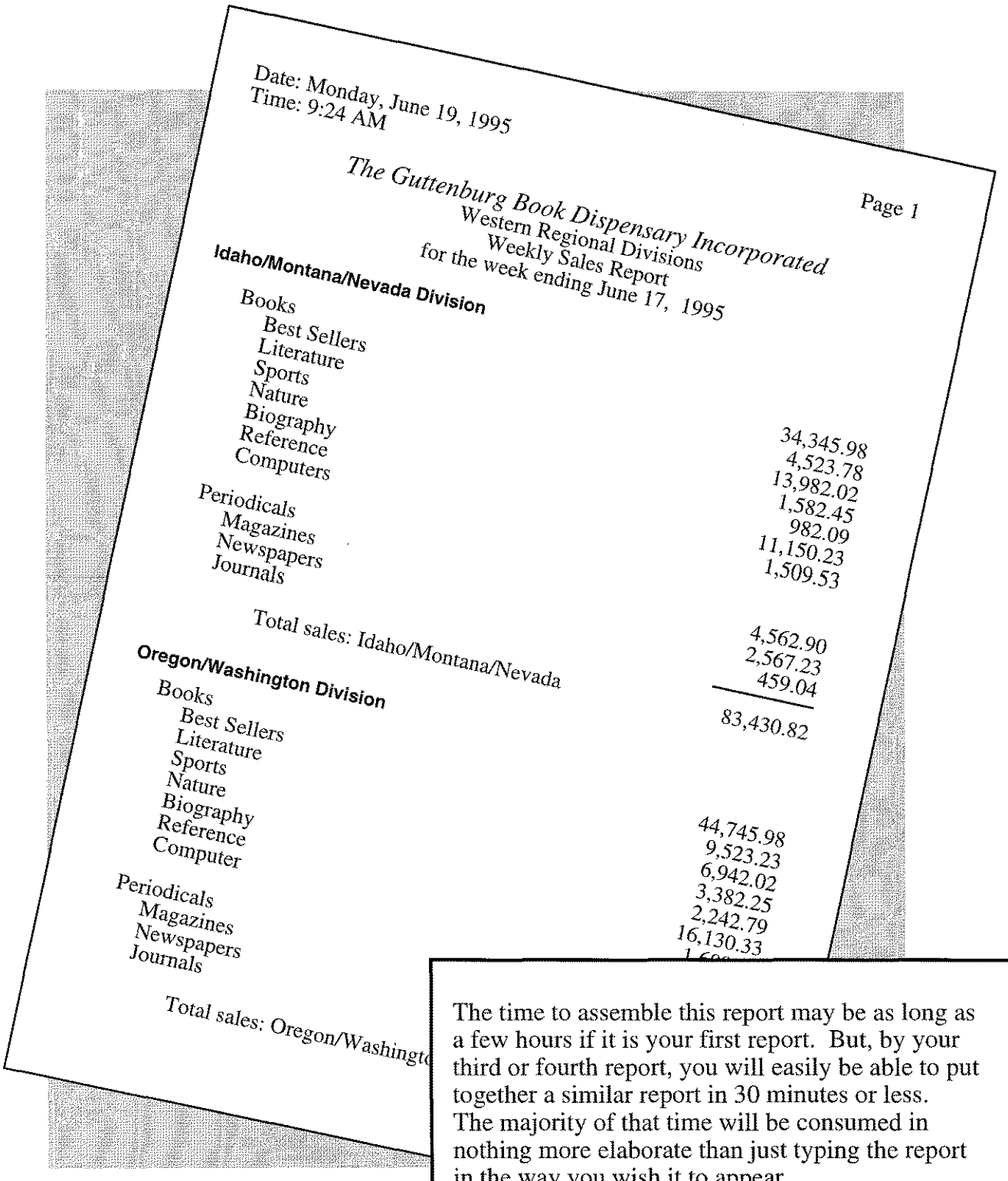
Creating the query questions is easily done. A uniform but powerful English-like syntax was created to allow you to ask just about anything imaginable. A question for one cell in this report might be *"(Show me the) sum of sales+receivables when date ib (is between) 19880611,19880617 and division is NW and category is 521".*

The next query question down the right hand column is quite likely to be very similar to the one above it, except that now perhaps the category is 522 instead of 521. Because of the spreadsheet nature of QueryCalc, a single query question may be replicated down a column. The query questions may then be modified using QueryCalc's on-line cell editor. More sweeping changes can be as easily accomodated with QueryCalc's search and replace function.

That's all there is to getting data out of IMAGE, KSAM or MPE databases. If you don't specify the dataset or database, QueryCalc will determine the proper database and dataset to get this information on its own. If multiple chained paths are available, QueryCalc will optimize the question to search down the shortest path. A variety of techniques were employed in the design of QueryCalc to make its database searches as fully self-optimizing as possible.

To complete the report on the right, only the column totals remain. Subtotals are created as column sums of the cells directly above [e.g., "SUM (E15:E21)"]. Displaying a grand total is no more difficult than composing a statement summing the subtotal cells, such as "E23+E34".

The report is now done. Putting an actual QueryCalc report together is no more difficult than it appears to be here.



Date: Monday, June 19, 1995
Time: 9:24 AM

*The Guttenburg Book Dispensary Incorporated*
Western Regional Divisions
Weekly Sales Report
for the week ending June 17, 1995

Page 1

**Idaho/Montana/Nevada Division**

| | |
|---|---|
| Books | |
| Best Sellers | 34,345.98 |
| Literature | 4,523.78 |
| Sports | 13,982.02 |
| Nature | 1,582.45 |
| Biography | 982.09 |
| Reference | 11,150.23 |
| Computers | 1,509.53 |
| Periodicals | |
| Magazines | 4,562.90 |
| Newspapers | 2,567.23 |
| Journals | 459.04 |
| Total sales: Idaho/Montana/Nevada | 83,430.82 |

**Oregon/Washington Division**

| | |
|---|---|
| Books | |
| Best Sellers | 44,745.98 |
| Literature | 9,523.23 |
| Sports | 6,942.02 |
| Nature | 3,382.25 |
| Biography | 2,242.79 |
| Reference | 16,130.33 |
| Computer | 1,6[illegible] |
| Periodicals | |
| Magazines | |
| Newspapers | |
| Journals | |
| Total sales: Oregon/Washingt[illegible] | |

The time to assemble this report may be as long as a few hours if it is your first report. But, by your third or fourth report, you will easily be able to put together a similar report in 30 minutes or less. The majority of that time will be consumed in nothing more elaborate than just typing the report in the way you wish it to appear.

## The Design Criteria of QueryCalc

It doesn't take a good manager long to realize that the information being accumulated in the corporation's databases can tell him much about his company, especially about what's making money and what's not. The reports that are needed are often spur-of-the-moment questions, such as shipping cost analyses, inventory turn-around times, profit and loss sheets for the various subdivisions. These reports weren't initially planned for, but now they could be eminently profitable.

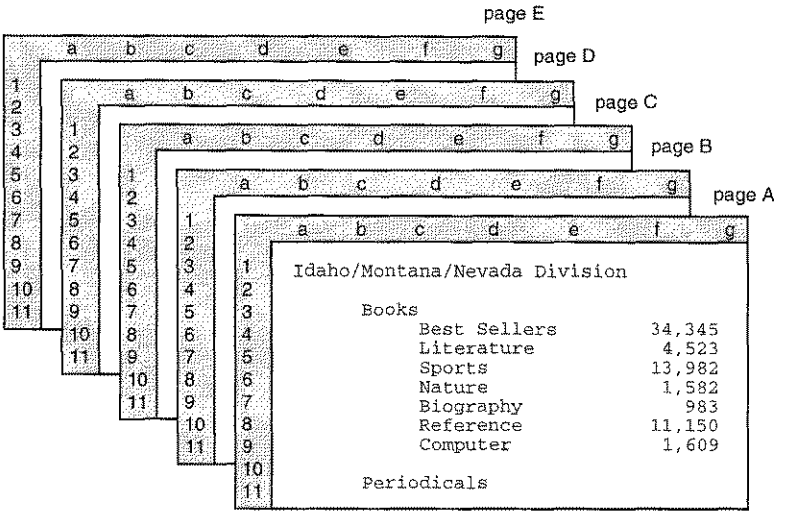
QueryCalc was designed from its inception to provide a mechanism so that members of the management staff could go to lunch, talk about what information they need, go back to the office and have the reports ready, with graphics, by about three in the afternoon. The report may be written by either a member of the programming staff or, even more likely, by one of the people at lunch. QueryCalc reports can be assembled so quickly that the final report can be put together while it is still being discussed.

Quite likely, better than 70% of the *ad hoc* reports that were designed on the spur of the moment have lasting value when regularly updated with new data, and will be used again. QueryCalc was designed to produce reports which are efficient, self-optimizing and capable of being run as regularly scheduled production reports.

## How QueryCalc Works

Each of QueryCalc's 26 pages is composed of 26 columns by 90 rows (2340 cells) for a total of 60,840 cells. If you've had any previous spreadsheet experience, you already have a very good idea how to use QueryCalc. Although QueryCalc performs all normal spreadsheet functions (indeed many functions like depreciation or cashflow are far better implemented than you've ever seen them before), QueryCalc was meant first and foremost to be an HP3000 IMAGE database report writer.

To be the most plastic report writer possible, QueryCalc was designed around the concept that characterizes the best of the word processors: "what-you-see-is-what-you-get". If you make a change in a column's width, move text, or insert rows, what you see on the screen will be what you will see on the printed page. The output from QueryCalc can be directed to any printer on the HP3000, your terminal's printer or to an MPE(flat) ASCII file where it can be picked up by HPWORD, DSG, or other such software for further processing or incorporation into another, longer document.




*The 3-dimensional nature of QueryCalc*

The column widths for each page are independently configurable. Thus each page can be formatted differently. However, QueryCalc is fully three dimensional. A normal QueryCalc cell reference might be Ag15, where the first letter specifies the page (A), the second the column (g) and the number the row (15) indicating where the data is to come from. A sum [e.g., "SUM(BG15:EN25)"] may range over a column, a row, a rectangle, a vector through the spreadsheet, or a cube as in this example. Any cell that exists anywhere on any page is referenceable, even if its width has been collapsed to zero so that it will not print.

## How to Use The *Applications Guide*

This *Applications Guide* is constructed in much the same fashion as QueryCalc. QueryCalc is a language-like program constructed of *primitives*. There are only four types of cells in QueryCalc (*numeric equations, text labels, text equations* and *query questions*) and only one command line. Everything in QueryCalc is built around these few primitives. What this means is that if you know how to do all of the simple procedures in QueryCalc, you'll have a good idea how to do something more complex.

Each chapter in the *Applications Guide* is directed towards a specific subject. Most of the chapters contain examples of actual reports, many of which reference the reports found in the demonstration account supplied with QueryCalc.

For the first several chapters, the keystrokes necessary to create each report

are listed, either in whole or in part. Practice entering and executing these reports. You will learn best by doing, not just by simply reading.

## Advice for the New User

QueryCalc was designed so that one useful report may be mimicked many times over. And that, by the way, is one of the three basic "tricks" known to every successful programmer. At the risk of giving away the sorcerer's secrets, there are just these three items worth remembering:


1. ***Be aggressive.*** This is the most important trick of programming you'll ever learn. QueryCalc was designed precisely so that you can be aggressive. QueryCalc opens your IMAGE, KSAM or MPE databases in a read-only mode without locking the datafiles. Because you can't modify, create or delete any information in the database, you can do no harm to the database. QueryCalc opens your databases with such a light touch that system backups can proceed while you are executing you reports.

   The worst harm that you can do is destroy your own work. But if sufficient time has elapsed since the report was first created, quite likely an earlier (if not identical) version of the report has been stored on a backup tape. If you work in a large organization and do not know your system managers well, such times are excellent opportunities to get to know them better.

   QueryCalc was designed to automatically generate efficiently executed reports and will guide you towards the creation of a well thought-out construction. How can you tell if your report is efficient? Generally, just by the time it takes to execute. If you have done something that is quite slow (which may imply some sort of inefficiency), you will especially notice it in session mode. Execution that is taking some time can always be stopped in mid-process by pressing the CNTL-Y keys. You may then examine the cells that are taking so much time. By the time you are ready to job-stream your reports, they are almost always quite well-constructed and efficient.

   The bottom line moral remains: *be agressive.* You can do no harm to the databases, to the HP3000, or generally do anything that is all that inefficient. Being agressive is the only way you will learn.


2. ***Program by imitation.*** The second great trick of programming is called *"ditto programming"*. Every good programmer knows the trick. When you look at someone else's reports, think abstractly. The report you see won't be precisely the same one you need, but it is probably more similar than different. You already know a great deal of information about your own databases. Even if you don't know what the databases are called or where they are, you have a very good idea of what's in them and how important that information is to you. And you know what information you need to see. Search through the *Applications Guide* and find reports which generate report structures similar to those which would be useful to you. You should be able to write comparable reports using your own databases in 2 or 3 days.


3. ***Know your databases.*** This third bit of advice is critical. There is a growing tendency among many companies which manufacture report writing programs to isolate the user from the database, to relieve him of the requirement of intimately knowing the databases from which he is extracting his information. If that is not an easy recipe for disaster, it is at least a quick recipe for confusion and error.

   A database is meant to be the *image* of a steel filing cabinet, filled with paper records. Nothing more. If you could have found the information you needed in a standard filing cabinet, you can find it in your databases. This is especially true of IMAGE on the HP3000, as you'll see in the following chapter. Almost all of the difficulty you'll have in using databases will come from determining what someone else called the items and finding out where they're located. But that's much the same problem you'd have when first using someone else's standard filing cabinets. The easiest solution is the obvious one: ask whomever is in charge of your databases for an explanation.

# 2

*Understanding the electronic filing cabinet*

# The Database

## What is an IMAGE Database?

The HP3000, with its IMAGE† databases, was bought to replace steel filing cabinets and paper records. The advantage that accrues in making your records electronic is the ease with which you can now search, collate and extract information. You now have the capability to generate analytical summaries which you never would have previously done using the paper records, simply due to the magnitude of the effort required.

IMAGE is a particularly simple, but powerfully and efficiently implemented database. IMAGE was designed from its beginnings to be the *image* of a steel file cabinet (hence its name). QueryCalc extends this basic idea as a combination of two procedures that are common to every office: filing cabinets and spreadsheets. The filing cabinets and spreadsheets have simply become electronic. Even so, procedures remain unchanged from those you would normally perform if you were to gather up the information from the paper records. With QueryCalc, you are going to flip through the records in filing cabinets, calculate sums and averages, and enter that information directly onto the spreadsheet, creating complex summary reports rather easily.

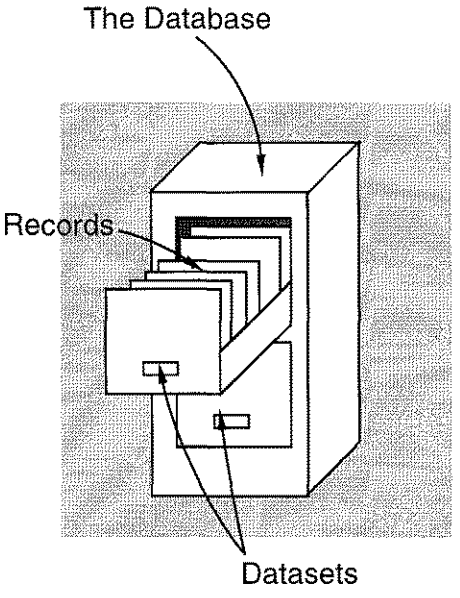
So, if QueryCalc is to be used effectively, it becomes important to understand how the filing cabinet is put together. To do this, we'll use the training database that was supplied with your copy of QueryCalc. This training database is a real database, containing 10,000 invoices and 6,000 labor tickets. The database was supplied by one of QueryCalc's user organizations, a construction company. The names of individual people, projects and companies have been altered, but otherwise, it's real.

---

†IMAGE is not the only database structure found on the HP3000, but it is the one most commonly used. Other common databases which QueryCalc supports are KSAM and MPE files (often called *flat files*). All database structures are fundamentally the same. And all database queries are identical in QueryCalc. Because IMAGE is the most commonly used database, this chapter emphasizes IMAGE. But, if you understand IMAGE, you automatically have a good feel for the others.

## Visualizing a Database

A *database* is the electronic equivalent of a filing cabinet. Each of its individual drawers are *datasets*. The names come from mathematical set theory, but the names are generally inconsequential. It would have been just as easy to call the database the filing cabinet and the datasets filing drawers. If IMAGE had been designed in today's "user-friendly" climate, the names would probably be different. But IMAGE was designed in the early 1970's when the principal users were data processing professionals with substantial mathematical training. Databases and datasets were appropriate names to their education and backgrounds. We want you to think of IMAGE as a steel filing cabinet filled with drawers.




What should you expect to find when you open a file drawer (a dataset)? Hundreds of manila folders. In IMAGE, the folders are called *records*. An electronic database differs from a paper-filled cabinet drawer in one very particular way. In any single real cabinet drawer you may have a mix of folder types: some personnel records, some invoices, and anything else that might fit in the drawer. That won't happen in a database. An electronic file drawer (a dataset) can contain only one type of record. The whole drawer must be dedicated to invoices (or payroll checks or employee records). The dataset (the drawer) may be very large or it may be very small, but it will always be consistent and uniform.

This same requirement of uniformity holds true for what's written into each manila folder (record). With paper records, you have the capacity to write information in the margins, scribbling notes anywhere where you can find room. Again, that can't happen in an electronic record. You can only write or read information from lines previously specified on the record. Every record in the dataset (the drawer) is identical in format. It's what's entered in those specific lines that makes up your database information. These line-by-line entries are called *dataitems* in IMAGE.

Who decided what dataitems make up a record? Quite often, it's you—or someone just like you sometime in the past. The electronic record in the dataset is generally no more than what you would have typed onto a paper

record if it had been up to you to generate a form. Presume that you are in charge of a construction company and it's your responsibility to generate a labor ticket form for the employees to fill out daily. What would you make the labor ticket look like if it had to be done on paper?

Quite likely, you would choose something like this:

```
                EMPLOYEE LABOR TICKET
          Name:_____________________________
          Date:_____________________________
    Job Number:_____________________________
 Regular hours:_____________________________
Overtime hours:_____________________________
```

## Designing the Database

The electronic version will be virtually identical, but with one important difference. Space in a computer is generally expensive. Text almost always requires more room than a number. Storing a person's name as text has two principal disadvantages. The first is the waste of space in storing a complete name where a number (employee number, social security number) would do. And the second is the ambiguity associated with names. Any one person can have a variety of things he or she may be called (Mary Smith, M.B. Smith, etc.). So rather than use names, it is almost always preferable to use a number to identify a person.

Thus, the computer-equivalent record would probably look like this:

```
                LABOR DATASET
SOCSECNUM:
     DATE:
    JOBNUM:
  REGULAR:
 OVERTIME:

      CAPACITY:10000    ENTRIES: 5792
```

While this looks a little computer-*ish*, the reasons for making it so are straightforward. Social Security Number, which now takes the place of the person's name, has been reduced to SOCSECNUM for three reasons: (1) spaces aren't allowed in a dataitem's name, (2) no dataitem name can be

longer than 16 characters, and (3) it's just simply easier to type. Other than that, the form of the record is identical to the paper form. This one form will be duplicated in the LABOR file drawer (dataset) perhaps 1,000 times, 10,000 times or possibly 100,000 times, depending on how many records you think you'll need to serve the time period the records should cover. The number of potential records in a dataset is called its *capacity*. The capacity of a dataset is not a fixed amount; it can be changed as more space is required.

## Retrieving Information from the Database

An IMAGE database (the filing cabinet) may have up to 199 separate datasets (drawers) in it, each with its own record format. For a construction company, other datasets we would need are for JOBS-IN-PROGRESS, EMPLOYEE RECORDS, and INVOICES.

Presume that current information fills the database. How would you find something in the database? Let's say the question you need to ask is:

*What was the sum of all invoices charged to job 8404*
*during the first half of 1989?*

The important question to ask is: how would you go about obtaining the information if all of the records were in paper form? You would open the file drawer labeled INVOICES, find the invoices for job 8404, flip through them and add up the total charges for those with dates which properly qualify. That's exactly what you're going to do with QueryCalc and IMAGE too. The question in QueryCalc looks like this:

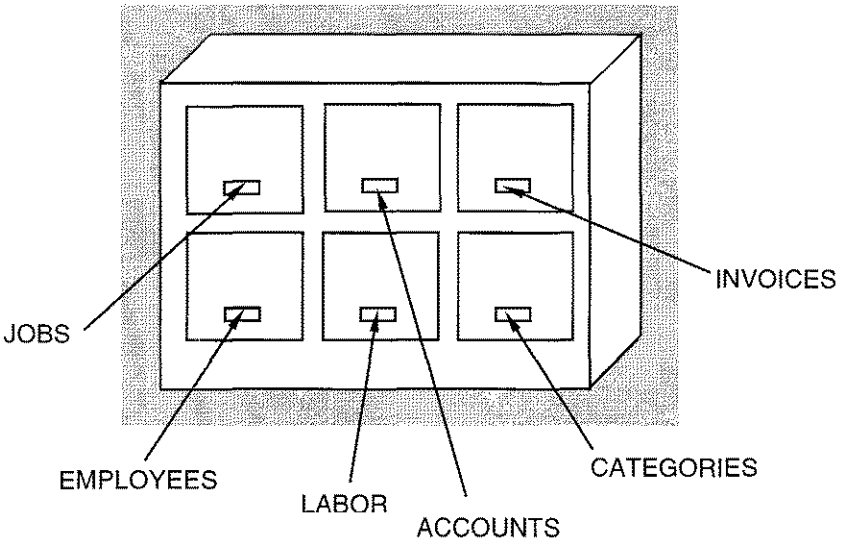
*@Using INVOICES, sum of AMOUNT when*
*JOBNUM is 8404 and DATE ib 890101,890630*

The question (called a *database query*) is written in a stylized English form both you and the computer can understand. It will always look something like this one, although you have a great many options in what you can ask.

The standard QueryCalc sentence will always occur in three basic parts: (1) the "*Using...*" phrase specifies which dataset (and optionally database) you wish to search through, (2) the "*sum of...*" phrase specifies what information you would like to retrieve, and (3) the "*when...*" phrase specifies the conditions that must be satisfied before the record is incorporated into whatever calculation is requested. As you'll see in a following chapter, the "*Using...*" phrase is not always necessary. QueryCalc can often deduce the

proper dataset (file drawer) from which to retrieve the information without it being fully specified.

The database QCDEMO and some of its datasets



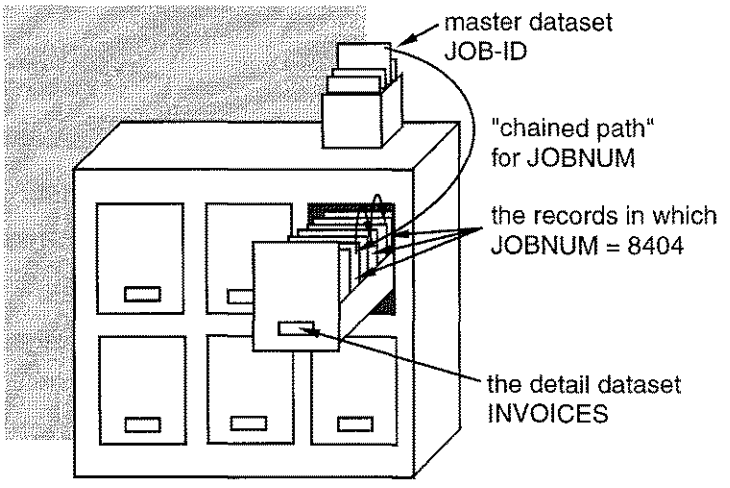

An actual database is likely to have more than a few datasets. Indeed, a database will probably have more file drawers than can be easily drawn. Six of the fifteen datasets of the training database, QCDEMO, are shown here. These are among the more important datasets in the database and will figure prominently in the exercises which will appear throughout the remainder of this *Guide.*

There are two ways to search a dataset for a particular piece of information. The first is straightforward. Open the dataset, begin at the beginning and read every record, front to back, marking those records which meet the qualifying criteria. This is called a *serial search.* This is a perfectly legitimate way to search a dataset, but it is also the slowest possible way to obtain a summary of qualifying records. If there are a 100,000 records in the dataset, this method may be too time consuming to be acceptable.

## High Speed Searches

The second way to search a dataset for a particular record or set of records uses indexing datasets (called *master datasets).* A master dataset sits on top of the filing cabinet like a file of 3x5 index cards, not unlike the indexing cards in a library file. But in this case, the 3x5 master dataset cards contain record numbers. If we wanted to find all of the invoices in the INVOICES dataset which are associated with the the job 8404, and the dataitem JOBNUM was a *search item,* the master dataset will point to first and last invoices which have JOBNUM = 8404. Each succeeding invoice itself will point to next and previous records which also have JOBNUM = 8404, thus forming a *chain.* The first record obviously can't point to a previous record, so it becomes the *chain head.* Likewise, the last record is the *chain tail.* Now, instead of searching the entire dataset, record by record, we can simply go to the 3x5 index card file, find the record for JOBNUM = 8404,

go to that particular record in INVOICES, starting optionally at either the chain head or the tail, follow the chain and read only those records which have the proper JOBNUM, as shown. In place of reading 10,000 records, we now may have to read only 30 records. This form of search will be much faster. The moral? *Always use a chained search if possible*. QueryCalc will do this automatically if one of the dataitems used in your query question is a search item. If there are multiple search items in your query question (implying multiple 3x5 card-like master datasets), QueryCalc will choose the chained path with the fewest number of entries. Alternatively, if your query question uses no search items at all, QueryCalc must use a serial search. QueryCalc will announce that too, specifying how many records will have to *serially* searched, thereby providing you with an estimate of how long the query question is likely to take.




## Selecting the Search Items

There are two kinds of datasets in an IMAGE database, *detail* datasets (the file drawers) and *master* datasets (the 3x5 index cards). And there are two kinds of dataitems. Technically, these are called either *key items* (search items) or *attributes* (non-search items), although these names aren't commonly used in reference to IMAGE databases. A search dataitem is a dataitem which has a chaining path back to a 3x5 card master dataset. The question is: which items should be search items. And which items should not be? The answer partly depends on your use of the database.

Consider the employee LABOR tickets dataset that we just constructed. Which items are good candidates for search items? The better way to ask that question is: how many ways are you likely to want to retrieve information from the LABOR dataset? Certainly, you would like to know the sum of hours that a particular person has worked. That means that SOCSECNUM must be a search item. You would also like summaries of all of the

labor tickets charged to a particular job, thus JOBNUM must also be a search item. Therefore, we can be sure that these two items on the LABOR dataset need to be *search* items. Are there any others?

```
                    LABOR DATASET

SOCSECNUM:          [search item]
     DATE:
   JOBNUM:          [search item]
  REGULAR:
OVERTIME:

        CAPACITY:10000    ENTRIES: 5792
```

Why shouldn't the dataitems for regular and overtime hours also be search items? Because you are rarely going to need to search for all of the records that have 13.1 hours of regular labor recorded in them. It's not the kind of question that you would normally ask. The only items that should be search items are those items that have some definite value, not an indeterminate value like 13.1. What about DATE then? It certainly has definite values. Should it be a search item? Possibly. Whether it should be or not wholly depends on what kind of questions you are likely to ask.

## Penalties Associated with Search Items

Are there penalties associated with making some items search items and others not? Yes, there are several, but they are generally small. IMAGE will only allow 16 items in any one dataset to be search items. This limit is not a severe restriction however. Good database design practice would generally not have more than about four, five or six search items in a single dataset. Each additional search item in a dataset requires that the dataset take up a bit more space on the disc; so there is a small penalty paid for disc space. A slightly more substantial penalty is that new entries placed into a dataset require greater processing time than they otherwise would. Existing search chains must be altered for each newly inserted dataset record. Or, alternatively, if a chain for the newly entered value does not already exist, a chain for the new value must be created. Again, these are generally not severe restrictions. The final penalty can occasionally be the most bothersome. A search item is a *key* item in the database and IMAGE will not allow key items to be modified or updated the way that non-key (*attribute*) items can be. The only easy way to modify a key item is to delete the entire record and re-enter it, thus automatically rebuilding all of the necessary chains.

*Search items will never be beneficial when entering data into a database.* In general, they can only slow data entry down. *But they will be invaluable when extracting and summarizing information from the database.* All report generators use available chained search paths, if possible. QueryCalc makes especially good use of the available chains. When, then, should search chains be defined? Fred White, one of the original members of the HP design team for IMAGE and now associated with Adager, offers these rules:

## Fred White's Rules for Chained Search Paths

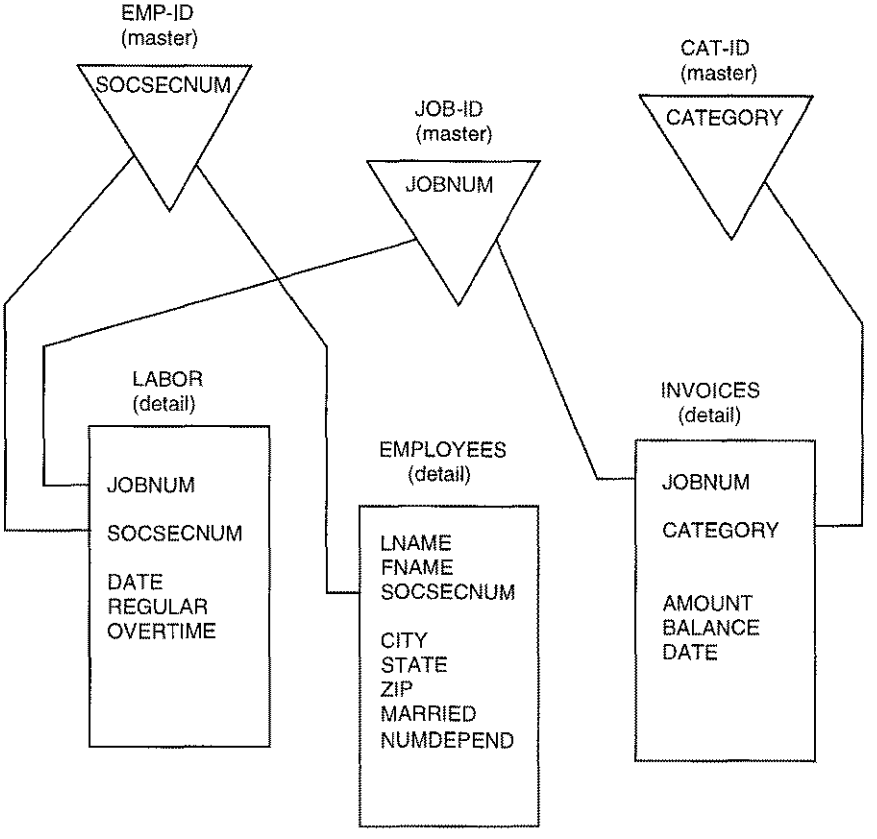
A chained path should be defined if:

- *it is necessary for the application,*
- *its speed of access is better than a serial search and it's frequently used, or*
- *its speed of access is so much better than a serial search that it's cost effective even if it's seldom used.*

If you are responsible for participating in the initial design of the database, the only question that usually arises concerns dataitems such as the DATE. Should DATE be a chained search item or not? Will it be used often enough to warrant a chain? Fred White's conclusion is: "*when in doubt, leave it out*". If the omission of a search item becomes a burden because of long serial search times, or searches are performed on the database in a manner not originally anticipated, that fact will become increasingly obvious. It's possible add a search path to an IMAGE database by using any one of the available products which restructure an IMAGE database (e.g., ADAGER from Adager, DBGENERAL from Bradmark Computer Systems, or DBCHANGE from Hewlett-Packard). But we politely disagree with Fred, who is otherwise a good friend. Our own experience suggests that if you have doubts, the item should be made a search item now rather than later.

## Schematizing The Database

An IMAGE database can quickly grow so complicated that attempting to draw it as a filing cabinet would become fairly messy. Thus, if you wish to draw the database, it becomes necessary to schematize it in the fashion shown on the facing page. Using standard symbols, master datsets (the 3x5 indexing cards) are drawn as triangles. Detail datasets (the file drawers) are drawn either as trapezoids (Hewlett-Packard's preferred notation) or as rectangles, as shown. Chained search paths are drawn as connecting lines from the master datasets to the detail datasets, landing on their respective




search items. But even the figure shown above is only a small fragment of the training database, QCDEMO; not all of the dataitems are shown for the detail datasets illustrated nor are all of the master datasets and their search paths shown. Most commonly, the structural form of the database is simply printed out rather than drawn.

How can you determine the form of your own databases? That's easily done using the @FORM command in QueryCalc. An examination of the structure of QCDEMO is a worthwhile brief exercise. Quite logically, you will not be able to fully utilize the information contained in your databases until you understand how the databases are constructed.

## A Walking Tour of QCDEMO

To see how QCDEMO, the construction company database, is put together, type the following at your terminal:

```
:HELLO USER.AICS
:RUN QC.QCPROGS.AICS
```

You are now in QueryCalc. Press RETURN to move into the first spreadsheet screen and type:

```
@OPENDB QCDEMO/FRONT
```

to open the database. The database password (FRONT) must be in all caps. To insure that the database was successfully opened, type:

```
@SHOWDB
```

If you have obtained proper entry into the database, QCDEMO will have an arrow pointing at its name, indicating that it is the *default* database (the database that will be chosen should there be several open at a time and you have not specified which you wish to use). *If the open has failed, the most likely cause is that QCDEMO has not yet been entered into QueryCalc's dictionary.* Press RETURN to return to the spreadsheet.

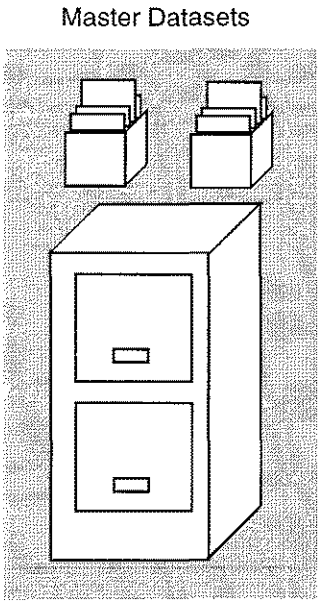
Now we are free to look at QCDEMO. Type:

```
@FORM
```

The first screen displays the datasets of QCDEMO and is duplicated on the opposite page. Just as 3x5 index cards rest on top of a filing cabinet, QueryCalc will place the master datasets on top of the detail datasets in the map of datasets. The information displayed is the name of the datasets, their current number of entries and their maximum capacities. Datasets more than 80% full are marked with an asterisk (the 80% full mark is often considered to be the maximum useful density of a dataset). Additionally, if the dataset is a master dataset, @FORM will indicate whether the dataset is a *manual master* or an *automatic master.*

## Automatic vs. Manual Masters

The difference between an automatic and a manual master is illustrated by the following example. If you were entering new employee labor tickets into the LABOR dataset and typed JOBNUM = 8912, and the indexing master dataset for JOBNUM was an *automatic master* and there were no previous entries in the master dataset for 8912, that job number would *automatically* be entered into the master dataset and a chained search path would be immediately established for the new value, 8912. This feature can be either useful or troublesome, depending on your perspective. If 8912 was typed in error, you would never know it. But if the master dataset (JOB-ID) had




SETS OF DATABASE QCDEMO

| MASTER DATASETS: | | ENTRIES | CAPACITY |
|---|---|---|---|
| CAT-ID | automatic | 120 | 501 |
| SUBCAT-ID | automatic | 70 | 101 |
| JOB-ID | automatic | 76 | 101 |
| ACCOUNT-ID | automatic | 483 | 1001 |
| EMPLOYEE-ID | automatic | 185 | 501 |
| KEYNAME-ID | automatic | 1 | 1 * |
| DUMMYSORT | automatic | 476 | 1001 |
| NAME-ID | automatic | 133 | 501 |
| **DETAIL DATASETS:** | | **ENTRIES** | **CAPACITY** |
| JOBS | | 53 | 102 |
| CHANGES | | 59 | 78 |
| JOBESTCOST | | 3598 | 4572 |
| TAXRATE | | 55 | 507 |
| CATEGORIES | | 120 | 510 |
| SUBCATS | | 67 | 114 |
| EMPLOYEES | | 181 | 505 |
| PAYRATE | | 737 | 1005 |
| LABOR | | 5796 | 7336 |
| PAYRECORD | | 4051 | 5137 |
| ACCOUNTS | | 482 | 1002 |
| INVOICES | | 9962 | 12610 |
| CHECKS | | 5445 | 6900 |
| SCHEDULES | | 0 | 1004 |
| FIXEDASSETS | | 82 | 504 |

*The master and detail datasets of QCDEMO*

been constructed as a *manual* master and the value 8912 had not already been entered *manually* into the master, a jobnum of 8912 would be rejected when you attempt entry into the detail dataset. All search item values entered into a detail dataset must be previously entered into their respective manual masters to prevent just this mistake.

Manual masters also have a few other attributes that differentiate them from automatic masters. An automatic master can never have more than one dataitem in it, the search item, and it must have at least one path to one detail dataset. Manual masters, in contrast, may contain many dataitems. And they may be "standalone" datasets (that is, they do not need to be associated with any detail dataset). However, neither form of the master dataset can ever have more than one search (key) dataitem in it..

## Mastering the Details

All of the master datasets in QCDEMO are automatic masters. That's often the case when the data entry (application) programs are written to check the validity of the data before it's entered into the database rather than use value checking features inherent to a manual master.

```
                    FORM OF DATABASE QCDEMO

 LABOR [detail dataset]

          ITEMS              DATA TYPE
           SOCSECNUM           U10  [text]   [search item]
           JOBNUM              I1   [num]    [search item]
           REGULAR             R2   [num]
           OVERTIME            R2   [num]
           DATE                U6   [text]   [search item]
           AMOUNT              R4   [num]
           SUBCAT              I1   [num]
           FICA                R2   [num]
           FUTA                R2   [num]
           NMESC               R2   [num]

        CAPACITY: 7336                ENTRIES: 5796
```

*The LABOR tickets dataset of QCDEMO*

To see what a dataset or dataitem looks like in detail, you can simply type the name of the dataset or dataitem once you are in @FORM. For example, type LABOR. What will appear on your terminal will be a map of the labor tickets dataset, as shown above.

This is the actual form of the LABOR tickets in QCDEMO, not all that different from the one designed a few pages earlier. The three chained search items, SOCSECNUM, JOBNUM, and DATE are marked. The maximum capacity of the dataset and its current entry count are shown, as are the dataitem types for each of the items. Only two types of data are stored in a database: text and numbers. QueryCalc is not particularly sensitive to the manner in which a number is stored. All numbers, regardless of their original data types, are automatically converted into high-resolution numbers (double reals) when they are imported from the database into the spreadsheet. Thus, displaying the numeric datatype is more for your information than for practical use. Although you are not likely to be greatly concerned about the actual datatypes of the individual dataitems, you'll find it useful to know what each data type means.

## Numeric Dataitem Types

Two basic types of number representations commonly exist in computers: *integers* (whole numbers) and *reals* (a "real" number is said to be a number with a decimal point) (Table 2-1). Both numeric types may be specified to be of various resolutions (that is, capable of resolving and accurately keeping track of perhaps a penny in a billion dollars). Real numbers are specified as either *short* (32-bit) or *long* (64-bit) numbers. Their respective data types are labeled R2 and R4 in IMAGE. Integer numbers (numbers without decimal points) may be either *short* (16-bit), *intermediate* (32-bit) or *long* (64-bit) integers, written as I1, I2 or I4 data types, respectively.

Slight modifications of the integer number types occur for the J1, J2 and J4 data types, which are used principally in COBOL, a common programming language. COBOL places additional restrictions on the range of values that an integer number may take. The J data types observe these restrictions. Another form of modified integers are the K data types, called *logicals*. A K1 or K2 number is identical to an I1 or I2 number, but is limited to positive values only.

Various other techniques have also been devised to store numbers. Two of these techniques involve the storage of numbers more like text than a normal numeric representation. Integer numbers stored in this fashion are called *packed* and *zoned integers* (P and Z data types, respectively). The resolution capacity of a packed or zoned integer is not fixed but is specified at the time of database creation.

## Text Dataitem Types

Only two types of text data types exist: either wholly upper case (the U data type) or both upper and lower case (the X data type). The length of the string of characters which can be entered into a text dataitem field is specified at the time the database is designed. Due to the manner in which text is stored, the length must always be an even number. Text dataitem lengths greater than 40 characters are rarely used.

## Printing the Form of the Database

If you are going to successfully extract information from your databases, you will find it useful to have printed copies of the structures of the relevant databases. In general, it is impossible to remember all of the dataitems and datasets if the databases are complex. Although the @FORM command is always available, printed copies of the database structures are a valuable reference.

| DATA TYPE | IMAGE TYPE SPECIFICATION | COMMONLY CALLED | LENGTH IN BITS | RESOLUTION |
|---|---|---|---|---|
| NUMBER | R2 | SHORT REAL | 32 BITS | 1 IN ±4 MILLION |
| NUMBER | R4 | LONG REAL | 64 BITS | 1 IN $\pm 2 \times 10^{16}$ |
| NUMBER | I1 | SHORT INTEGER | 16 BITS | 1 IN ±32,768 |
| NUMBER | I2 | MED. INTEGER | 32 BITS | 1 IN ±2 BILLION |
| NUMBER | I4 | LONG INTEGER | 64 BITS | 1 IN $\pm 2 \times 10^{19}$ |
| NUMBER | J1 | SHORT COBOL | 16 BITS | 1 IN ±10,000 |
| NUMBER | J2 | MED. COBOL | 32 BITS | 1 IN ±1 BILLION |
| NUMBER | J4 | LONG COBOL | 64 BITS | 1 IN $\pm 1 \times 10^{19}$ |
| NUMBER | K1 | SHORT LOGICAL | 16 BITS | 1 IN 65,536 |
| NUMBER | K2 | LONG LOGICAL | 32 BITS | 1 IN 4 BILLION |
| NUMBER | P | PACKED | 4 BITS/DIGIT | USER SPECIFIED |
| NUMBER | Z | ZONED | 8 BITS/DIGIT | USER SPECIFIED |
| TEXT | X | TEXT STRING | 8 BITS/CHAR | USER SPECIFIED |
| TEXT | U | UPPERCASE | 8 BITS/CHAR | USER SPECIFIED |

*Table 2-1. The data types used in IMAGE*

Three views of the database are recommended for printing. They are: function key [f1], *the sets of the database*; function key [f4], *the paths of the database*; and function key [f2], *the complete structure of the database*. To print these views to your system line printer, press first [f6], *print to the line printer*, and then in turn [f1], [f4], and [f2]. Press [f7], *print to terminal*, to redirect the output of @FORM back to your terminal's screen.

From these three views you have all of the information necessary to draw the database schema in the manner shown on page 2-9. All that needs to be done is to connect the master datasets with their respective search items in the detail datasets. The view [f4], *the paths of the dataset*, shows you how that is to be done. As you become more familiar with IMAGE, you will be able to draw the appropriate schemas more rapidly and will soon dispense with the need to draw them at all.

The complete printed @FORM of the training IMAGE database, QCDEMO, appears on the next several pages, printed in the order: Sets [f1], Paths [f4], and All [f2].

SETS OF IMAGE DATABASE QCDEMO

| MASTER DATASETS: | | ENTRIES | CAPACITY | |
|---|---|---|---|---|
| CAT-ID | automatic | 120 | 501 | |
| SUBCAT-ID | automatic | 70 | 101 | |
| JOB-ID | automatic | 76 | 101 | |
| DATE-ID | automatic | 951 | 3659 | |
| ACCOUNT-ID | automatic | 483 | 1001 | |
| EMPLOYEE-ID | automatic | 185 | 501 | |
| DUMMYSORT | automatic | 1 | 1 | * |
| KEYNAME-ID | automatic | 476 | 1001 | |
| NAME-ID | automatic | 133 | 501 | |

| DETAIL DATASETS: | ENTRIES | CAPACITY |
|---|---|---|
| JOBS | 53 | 102 |
| CHANGES | 59 | 78 |
| JOBESTCOST | 3598 | 4572 |
| TAXRATE | 55 | 507 |
| CATEGORIES | 120 | 510 |
| SUBCATS | 67 | 114 |
| EMPLOYEES | 181 | 505 |
| PAYRATE | 737 | 1005 |
| LABOR | 5796 | 7336 |
| PAYRECORD | 4051 | 5137 |
| ACCOUNTS | 482 | 1002 |
| INVOICES | 9962 | 12610 |
| CHECKS | 5445 | 6900 |
| SCHEDULES | 0 | 1004 |
| FIXEDASSETS | 82 | 504 |

* The indicated dataset(s) are now more than 80% full.

*The [f1] printout. A listing of the master and detail sets in the selected database.*

PATHS OF DATABASE QCDEMO

| this master dataset has _____ | paths to these detail datasets ___ | using this search item _____ | with this item sorting the chain |
|---|---|---|---|
| CAT-ID | CATEGORIES | CATEGORY | |
| | INVOICES | CATEGORY | |
| | SCHEDULES | CATEGORY | |
| | FIXEDASSETS | CATEGORY | |
| SUBCAT-ID | JOBESTCOST | SUBCAT | |
| | SUBCATS | SUBCAT | |
| | INVOICES | SUBCAT | |
| JOB-ID | JOBS | JOBNUM | |
| | CHANGES | JOBNUM | |
| | JOBESTCOST | JOBNUM | |
| | TAXRATE | JOBNUM | |
| | PAYRATE | JOBNUM | |
| | LABOR | JOBNUM | |
| | INVOICES | JOBNUM | |
| DATE-ID | LABOR | DATE | |
| | PAYRECORD | DATE | |
| | INVOICES | DATE | |
| | CHECKS | DATE | |
| ACCOUNT-ID | ACCOUNTS | ACCTCODE | |
| | INVOICES | ACCTCODE | |
| | CHECKS | ACCTCODE | |
| EMPLOYEE-ID | EMPLOYEES | SOCSECNUM | |
| | PAYRATE | SOCSECNUM | |
| | LABOR | SOCSECNUM | DATE |
| | PAYRECORD | SOCSECNUM | |
| DUMMYSORT | EMPLOYEES | ENTER-AN-X | LNAME |
| | ACCOUNTS | ENTER-AN-X | KEYNAME |
| KEYNAME-ID | ACCOUNTS | KEYNAME | |
| NAME-ID | EMPLOYEES | LNAME | |

The *Paths* printout compactly displays much the same information shown in the database schematic (page 2-9) and the filing cabinet drawing (page 2-6) for the master dataset *job-id* and two of the detail datasets (*labor* and *invoices*) it links to. All three views are equally valid. The moral? Never let your mental image of IMAGE become more complicated than the filing cabinet.

*The [f4] printout. A list of the chained paths which link the master datasets to detail datasets in the selected database.*

```
                    SETS OF DATABASE QCDEMO

-------------------------------------------------------------------------

CAT-ID [automatic master]

        ITEMS                 DATA TYPE
          CATEGORY              I1  [num]     [key item with 4 paths]

        CAPACITY: 501         ENTRIES: 120                 24% FULL

-------------------------------------------------------------------------

SUBCAT-ID [automatic master]

        ITEMS                 DATA TYPE
          SUBCAT                I1  [num]     [key item with 3 paths]

        CAPACITY: 101         ENTRIES: 70                  69% FULL

-------------------------------------------------------------------------

JOB-ID [automatic master]

        ITEMS                 DATA TYPE
          JOBNUM                I1  [num]     [key item with 7 paths]

        CAPACITY: 101         ENTRIES: 76                  75% FULL

-------------------------------------------------------------------------

DATE-ID [automatic master]

        ITEMS                 DATA TYPE
          DATE                  U6  [text]    [key item with 4 paths]

        CAPACITY: 3659        ENTRIES: 951                 26% FULL

-------------------------------------------------------------------------

ACCOUNT-ID [automatic master]

        ITEMS                 DATA TYPE
          ACCTCODE              I1  [num]     [key item with 3 paths]

        CAPACITY: 1001        ENTRIES: 483                 48% FULL

-------------------------------------------------------------------------

EMPLOYEE-ID [automatic master]

        ITEMS                 DATA TYPE
          SOCSECNUM             U10 [text]    [key item with 4 paths]

        CAPACITY: 501         ENTRIES: 185                 37% FULL

-------------------------------------------------------------------------
```

*The [f2] printout. A complete listing of all of the datasets, data-items, and dataset capacities for the selected database.*

```
DUMMYSORT [automatic master]

        ITEMS                DATA TYPE
          ENTER-AN-X           U2  [text]    [key item with 2 paths]

        CAPACITY: 1          ENTRIES: 1                  100% FULL
```

---

```
KEYNAME-ID [automatic master]

        ITEMS                DATA TYPE
          KEYNAME              U14 [text]    [key item with 1 path]

        CAPACITY: 1001       ENTRIES: 476                 48% FULL
```

---

```
NAME-ID [automatic master]

        ITEMS                DATA TYPE
          LNAME                U14 [text]    [key item with 1 path]

        CAPACITY: 501        ENTRIES: 133                 27% FULL
```

---

```
JOBS [detail set]

        ITEMS                DATA TYPE
          JOBNUM               I1  [num]     [search item]
          DESCRIPTION1         U40 [text]
          DESCRIPTION2         U40 [text]
          DESCRIPTION3         U40 [text]
          DESCRIPTION4         U40 [text]
          SITENUM              I1  [num]
          CONTRACTAMOUNT       R4  [num]
          BIDDATE              U6  [text]
          STARTDATE            U6  [text]
          MONTH                U2  [text]
          ESTCOMPDATE          U6  [text]
          ACTCOMPDATE          U6  [text]
          WCEXEMPT             U2  [text]
          BUILDERSRISK         U2  [text]
          OWNER                U20 [text]
          ADDRESS              U40 [text]
          CITY                 U16 [text]
          STATE                U2  [text]
          ZIP                  U10 [text]
          PHONE                U12 [text]
          JOBCOST              R4  [num]
          WORKCOMPLETE         R4  [num]
          PRIORCOST            R4  [num]
          PRIOREARNED          R4  [num]
          BILLED               R4  [num]
          RETAINAGE            R4  [num]
          GRT                  R4  [num]

        CAPACITY: 102        ENTRIES: 53                  52% FULL
```

```
CHANGES [detail set]

       ITEMS                DATA TYPE
         JOBNUM               I1  [num]     [search item]
         DESCRIPTION1         U40 [text]
         DESCRIPTION2         U40 [text]
         DESCRIPTION3         U40 [text]
         DESCRIPTION4         U40 [text]

       CAPACITY: 78         ENTRIES: 59                76% FULL
-------------------------------------------------------------------------------

JOBESTCOST [detail set]

       ITEMS                DATA TYPE
         SUBCAT               I1  [num]     [search item]
         JOBNUM               I1  [num]     [search item]
         ESTCOST              R4  [num]

       CAPACITY: 4572       ENTRIES: 3598              79% FULL
-------------------------------------------------------------------------------

TAXRATE [detail set]

       ITEMS                DATA TYPE
         JOBNUM               I1  [num]     [search item]
         SITETAX              R2  [num]
         STARTDATE            U6  [text]
         TERMDATE             U6  [text]

       CAPACITY: 507        ENTRIES: 55                11% FULL
-------------------------------------------------------------------------------

CATEGORIES [detail set]

       ITEMS                DATA TYPE
         CATEGORY             I1  [num]     [search item]
         SCHEDULE             U2  [text]
         DESCRIPTION1         U40 [text]
         AMOUNT               R4  [num]
         DATE                 U6  [text]

       CAPACITY: 510        ENTRIES: 120               24% FULL
-------------------------------------------------------------------------------

SUBCATS [detail set]

       ITEMS                DATA TYPE
         SUBCAT               I1  [num]     [search item]
         DESCRIPTION1         U40 [text]
         SUBCONTRACTED        U2  [text]

       CAPACITY: 114        ENTRIES: 67                59% FULL
-------------------------------------------------------------------------------
```

```
EMPLOYEES [detail set]

        ITEMS                DATA TYPE
          ENTER-AN-X           U2  [text]    [search item]
          LNAME                U14 [text]    [search item] [sort item]
          FNAME                U10 [text]
          SOCSECNUM            U10 [text]    [search item]
          ADDRESS              U40 [text]
          CITY                 U16 [text]
          STATE                U2  [text]
          ZIP                  U10 [text]
          PHONE                U12 [text]
          MARRIED              U2  [text]
          NUMDEDUCTIONS        I1  [num]
          STARTDATE            U6  [text]
          TERMDATE             U6  [text]
          TERMREASON           U30 [text]
          INSURANCE            R2  [num]
          FICA                 R2  [num]
          FUTA                 R2  [num]
          NMESC                R2  [num]

        CAPACITY: 505        ENTRIES: 181              36% FULL
```

---

```
PAYRATE [detail set]

        ITEMS                DATA TYPE
          SOCSECNUM            U10 [text]    [search item]
          JOBNUM               I1  [num]     [search item]
          CLASS                U2  [text]
          RATE1                R2  [num]
          RATE2                R2  [num]
          RATE3                R2  [num]
          RATE4                R2  [num]
          RATE5                R2  [num]
          RATE6                R2  [num]
          RATE7                R2  [num]
          RATE8                R2  [num]
          RATE9                R2  [num]

        CAPACITY: 1005       ENTRIES: 737              73% FULL
```

---

```
LABOR [detail set]

        ITEMS                DATA TYPE
          SOCSECNUM            U10 [text]    [search item]
          JOBNUM               I1  [num]     [search item]
          REGULAR              R2  [num]
          OVERTIME             R2  [num]
          DATE                 U6  [text]    [search item] [sort item]
          AMOUNT               R4  [num]
          SUBCAT               I1  [num]
          FICA                 R2  [num]
          FUTA                 R2  [num]
          NMESC                R2  [num]

        CAPACITY: 7336       ENTRIES: 5796              79% FULL
```

---

```
PAYRECORD [detail set]

        ITEMS                DATA TYPE
          SOCSECNUM            U10 [text]    [search item]
          GROSS                R4  [num]
          FICA                 R2  [num]
          FUTA                 R2  [num]
          NMESC                R2  [num]
          FEDWITH              R2  [num]
          STATEWITH            R2  [num]
          DEDUCTION            R2  [num]
          CHECKNUM             I1  [num]
          AMOUNT               R4  [num]
          DATE                 U6  [text]    [search item]
          REGULAR              R2  [num]
          OVERTIME             R2  [num]
          INSURANCE            R2  [num]

        CAPACITY: 5137       ENTRIES: 4051              79% FULL
```

---

```
ACCOUNTS [detail set]

        ITEMS                DATA TYPE
          ENTER-AN-X           U2  [text]    [search item]
          DESCRIPTION1         U40 [text]
          KEYNAME              U14 [text]    [search item] [sort item]
          ADDRESS              U40 [text]
          CITY                 U16 [text]
          STATE                U2  [text]
          ZIP                  U10 [text]
          PHONE                U12 [text]
          ACCTCODE             I1  [num]     [search item]

        CAPACITY: 1002       ENTRIES: 482               48% FULL
```

```
INVOICES [detail set]

        ITEMS                DATA TYPE
          CATEGORY             I1   [num]     [search item]
          SUBCAT               I1   [num]     [search item]
          JOBNUM               I1   [num]     [search item]
          ACCTCODE             I1   [num]     [search item]
          INVOICENUM           I1   [num]
          AMOUNT               R4   [num]
          DISCOUNT             R2   [num]
          BALANCE              R4   [num]
          DATE                 U6   [text]    [search item]

        CAPACITY: 12610      ENTRIES: 9962              79% FULL
```

```
CHECKS [detail set]

        ITEMS                DATA TYPE
          ACCTCODE             I1   [num]     [search item]
          AMOUNT               R4   [num]
          DATE                 U6   [text]    [search item]
          CHECKNUM             I1   [num]

        CAPACITY: 6900       ENTRIES: 5445              79% FULL
```

```
SCHEDULES [detail set]

        ITEMS                DATA TYPE
          CATEGORY             I1   [num]     [search item]
          DESCRIPTION1         U40  [text]
          DESCRIPTION2         U40  [text]
          DESCRIPTION3         U40  [text]
          DESCRIPTION4         U40  [text]
          AMOUNT               R4   [num]
          DATE                 U6   [text]

        CAPACITY: 1004       ENTRIES: 0                  0% FULL
```

```
FIXEDASSETS [detail set]

        ITEMS                DATA TYPE
          CATEGORY             I1   [num]     [search item]
          DESCRIPTION1         U40  [text]
          DESCRIPTION2         U40  [text]
          AMOUNT               R4   [num]
          DATE                 U6   [text]
          SCHEDULE             U2   [text]
          YEARDEPR             R4   [num]
          NETVALUE             R4   [num]
          TERMDATE             U6   [text]

        CAPACITY: 504        ENTRIES: 82                16% FULL
```

# Concepts Introduced in Chapter 2

| | |
|---|---|
| DETAIL DATASET | the basic "file drawer" |
| MASTER DATASET | the 3x5 card index dataset |
| SEARCH ITEM | a dataitem in a dataset for which a chained search path exists |
| CHAINED PATH | the linking together of all of the records which share a common search item value |
| NON-SEARCH ITEM | any of the other dataitems in a dataset for which no search chain exists |
| AUTOMATIC MASTER | search item values (for which no chained search path currently exists) are automatically created when an automatic master is used |
| SERIAL SEARCH | a search of all of the records in a dataset, first to last |
| DATABASE FORM | the schematic drawing of the database's structure |
| REAL NUMBERS | numbers with decimal points |
| INTEGER NUMBERS | whole (non-fractional) numbers |
| RESOLUTION | the largest usable number after which accuracy will be lost |

# 3

*Using the spreadsheet as a report generation platform*

# The 3-D Spreadsheet

## Introduction to the Spreadsheet

Although the electronic spreadsheet is now extraordinarily popular and widely used, it's sometimes difficult to remember that it is not very old. The electronic spreadsheet was invented by Dan Bricklin in 1977 during his time as a graduate student at Harvard. As a business major, Bricklin was required to generate *pro forma* (paper spreadsheet) analyses of various commercial situations as part of his course work. Part and parcel of any spreadsheet is the characteristic that if you change one number, the effects ripple down the columns and across the rows. Bricklin invented the electronic spreadsheet simply because he grew tired of erasing holes in his papers due to constantly changing numbers.

The electronic spreadsheet makes an ideal report generating platform because of its inherent plasticity and its WYSIWYG nature. If you are not presently familar with spreadsheets, two attributes account for much of the spreadsheet's value. They are the ease by which columns and rows may be inserted, deleted and reformatted (while automatically adjusting all relevant cell references) and the capacity to replicate and edit blocks of cells. These two features alone eliminate a great deal of the effort necessary to put a report together. The design of QueryCalc especially emphasizes the report formatting features of the spreadsheet.

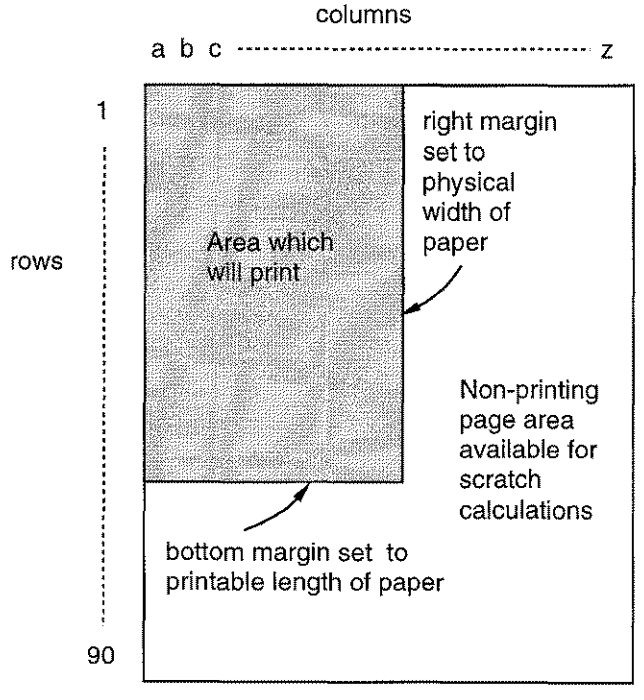
QueryCalc is a spreadsheet, and you can simply use it as a spreadsheet. But QueryCalc's more than that. QueryCalc is a combination of the most productive features of interactive computer languages, page-oriented word processors, spreadsheets and query languages. QueryCalc mimics a page-oriented word processing program as much as it does a spreadsheet. Where a standard spreadsheet is built as one very large, flat XY plane, QueryCalc is composed of 26, independently configurable pages.

You'll use QueryCalc differently than a standard spreadsheet. A PC-based spreadsheet is often used as a small database. Thousands of cells may be

used to store information such as outstanding invoices. If only eight cells in a row were used to list items such as customer name, account number, invoice number, invoice date, etc., a thousand cells will be consumed in listing of just 120 entries. One hundred entries obviously represents a very small database; nonetheless, 1000 cells take up a substantial area on the spreadsheet. You simply will not tend to use QueryCalc in such a fashion. IMAGE databases easily contain tens of thousands of records. A single cell in QueryCalc can request a summarization of all of those records.

## Printing the QueryCalc Page

QueryCalc also differs from standard spreadsheets in the manner it which printing occurs. QueryCalc will always print from the upper left-hand visible corner of the spreadsheet page out to the right margin and down to the bottom margin. A PC-based spreadsheet, by comparison, prints by specifying ranges of cells taken anywhere from within the body of the spreadsheet. *In QueryCalc, the upper left-hand corner of each spreadsheet page is the image of the physical page that will be printed. Although the cell widths on each page are independently configurable, the printing area remains the same on every page of the spreadsheet.* That's because the size of the paper you're printing to won't change, from page to page.




Only fifteen rows will be displayed on your terminal's screen at any one time, although 90 rows are available on the spreadsheet page. No readily available piece of paper is greater than 90 rows in length. The printable length of the spreadsheet page will be dynamically determined based on: (1) the paper size you specify, (2) the number of lines per page, and (3) the top and bottom margin settings. As you increase the size of the top and bottom margins, or add headers or footers, you simultaneously shrink the number of rows available to you in the print area. The physical size of the paper can't change, therefore the printable area must shrink (for more information, see Chap. 7, "Printing the Report").

*On a QueryCalc page, the printable area is defined by the right and bottom margin markers.*

## Learning to Become a Marginal Programmer

Traditionally, when numbers are entered into the rows and columns on an accountant's paper greensheet, they are placed there neatly and with precision. But the necessary scratch calculations are scribbled in the margins. We want you to use QueryCalc's margins in the same manner. All of the area to the right and below the margins on a page can be used for scratch calculations, temporary variables, or anything else you need to have available during report development, but don't wish to appear in the final printed form.

With a QueryCalc spreadsheet, *the margins don't even necessarily have to be at the sides.* Individual columns can be collapsed to zero width [using the /CWID command (see Ch. 4)]. Zero-width columns will no longer print, but the information in these hidden cells will recalculate normally and the results can be referenced as any other cell would be.

Cell definitions on any one of QueryCalc's 26 pages can reference cells anywhere in the spreadsheet. Sums (averages, variances or whatever) may range over columns, rows, rectangles, cubes or any combination of these. This feature accounts for the true three-dimensional nature of QueryCalc. But you'll find that the three-dimensional nature is more valuable than simply that. Although each page may be substantially different in what it is calculating and where it is obtaining its information, the easy reference to previously obtained values on other pages makes the preparation of complex reports simple.

## The Three Standard Report Forms

People tend to only write three kinds of reports on the HP3000. They are:

- *The indefinite detail list report*
- *The management summary report*
- *Graphics*

Each of these forms is easily done in QueryCalc. The first, the *indefinite detail list* report, is of this form: "Find me all of the people who owe us money and print me a list". The formating structure for the indefinite list report tends to be generally simple, but the length of the report is unknown before it's run, thus its indefinite nature. The second report form, a *management summary* report, is usually much more complicated to put together, and is generally quite complex in its formatting. Information is not simply repeated over and over as it is a detail list report. A management summary report is composed of many summarizations taken from many

different sources and then placed on specific areas of the page. The spreadsheet structure is obviously ideal for constructing the management summary report. The first and most obvious reason is the flexibility allowed in reformatting the page layout. But the most important reason is the interactive nature of the spreadsheet.

## The Four Cell Types of QueryCalc

Standard spreadsheets such as VisiCalc and 1-2-3 possess only two types of cells: *numeric equation* cells (which include simple numeric entries) and *text labels*. In contrast, QueryCalc has four cell types. They are the two VisiCalc cell types, *text equations* and *query questions*.

A **numeric cell equation** may be any one of the following forms:

```
34
sqr(g34)*1.0675*(g37+g38)
bd15*4
sum(aa1:cd13)
```

A **text label** is simply text written on the surface of the spreadsheet, not unlike this sentence. Text labels are not active cell types (that is, they do not change on recalculation). Examples are:

```
^Jan 1992
'Expense Report for FY94
```

A **text equation**, on the other hand, is an active cell type. The text equation extracts text information from other cells anywhere on the spreadsheet and recombines the text as you desire. Examples are:

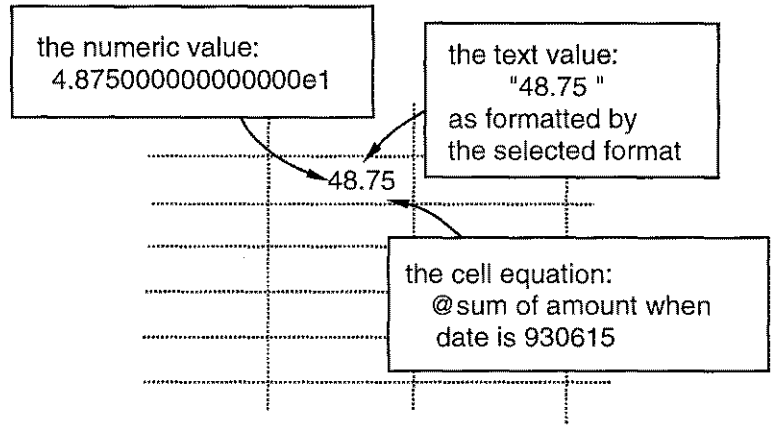
```
$"The value is: "+ups$(g34)
$a34+", "+a35
$dat$3(sysdate)+", "+hrs$(24)
```

A **query question** extracts and summarizes information from any of the databases (IMAGE, KSAM or MPE) which you presently have open. Examples are:

```
@Using invoices, sum of amount when
   category is 501 and date>890530
@Using qcdemo.labor, avg of overtime
   when jobnum is 8404
```

## The Triple Nature of Every Cell in QueryCalc

Every cell in QueryCalc, regardless of cell type, is composed of three parts: the *cell equation*, the cell's *text value* and the cell's *numeric value.*†




*QueryCalc is extremely weakly typed. That means you can add numbers and text together and get an answer.* Any text cell that contains only numeric characters takes on the numeric value of that text (that is, "3956" = 3956). Text cells which contain both alpha and numeric characters (or punctuation symbols) cannot be converted to a number, thus they are given a numeric value of zero ("-56A-" = 0). Conversely, all cells have a text value. Text from any or all of the four cell types can be combined in a single text equation.

A numeric equation, such as

```
b37*1.4
```

extracts and uses the numeric value from the cell B37. A text equation such as

```
$"Result: "+b37
```

extracts the text (formatted) value from cell B37. *Although the cell formatting routine will often round off the text in a cell to the specified number of digits, the actual numeric value is never rounded off. The effect is only superficial.* The numeric equation always works with QueryCalc's maximum resolution of 16.5 decimal points and never throws away information.

NOTE

†A detailed explanation of how a QueryCalc cell is constructed appears in the Technical Appendix, Section A: "How a QueryCalc Cell is Constructed".

The dual text/numeric nature of cells in QueryCalc is especially important to query questions. Text and numbers are the only two forms of items that can be stored in a database. To properly find (qualify) items in a database, text must be matched with text and numbers must be matched with numbers.

The following examples illustrate the point:

```
1. @Using invoices, sum of amount when
     category is [b7]

2. @Using invoices, sum of amount when
     category is [$b7]
```

For Example 1, presume that the dataitem CATEGORY is a numeric (I,J,K,P,Z or R) dataitem type. The equation in brackets which retrieves information off of the spreadsheet should therefore be a numeric equation (B7). A numeric value will properly be matched against a numeric value.

For Example 2, presume that CATEGORY is a text (X or U-type) dataitem. The equation in brackets must therefore be a text equation ($B7). The spreadsheet-extracted text will be matched against a text dataitem.

QueryCalc will do its best to properly match item types, but it may occasionally require your help. Should you ever have problems qualifying items in a database with values taken off of the spreadsheet, the problem almost always lies in mistaking a text field for a numeric field. *Text fields are used surprisingly often to store only numbers.* When numbers appear in a text field, they may look like numbers, but they're not. They're text characters and they must be treated that way.

Text and numeric equations which appear in query questions do not need to be simple spreadsheet data extractions. You may dynamically "manufacture" the patterns to be matched, as illustrated in the following examples:

```
@Using labor.tickets, avg of regular-hours
     when date is [190000+ab7*100+ab8]
```

*(where date is a numeric dataitem)*

```
@Using parts.partfields, find when partnum
     is [$"MJR"+hg34+"-00-1A"]
```

*(where partnum is a text dataitem)*

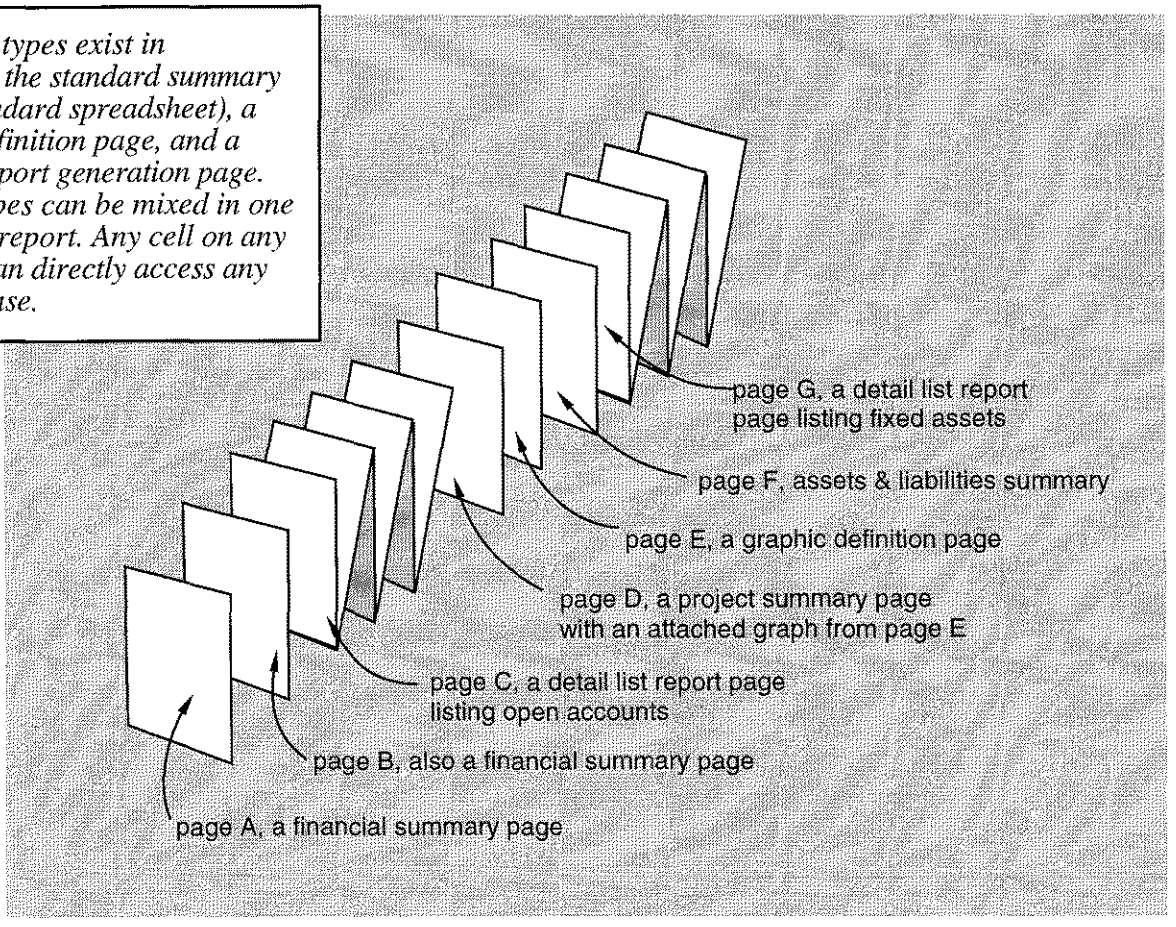
## The Three Page Types of QueryCalc

A QueryCalc report is composed of three distinct page types:

- *summary pages*
  (standard spreadsheets with or without query questions in the cells)
- *graphics definition pages*
  (used to define a graph)
- *detail list report generation pages*
  (used to create a detailed list of items).

The cells found in each of the pages, however, are identical. Any cell on any page can access any of the open databases, local or remote, to which you have access. The combination of the three page types allows the creation of virtually every possible form of report. Although the page types are distinct in their usages, the standard spreadsheet summary page is the basis of all three page types.

*Three page types exist in QueryCalc: the standard summary page (a standard spreadsheet), a graphics definition page, and a detail list report generation page. All three types can be mixed in one QueryCalc report. Any cell on any page type can directly access any open database.*



A *graphics definition page* is a standard spreadsheet summary page with a graphic template superimposed. By filling in the requisite information in the template, any one of six basic graph types (*standard bar, stacked bar, clustered bar, pie, point plot, and high/low/close chart*) is defined. Information may come from anywhere else in the spreadsheet or be calculated immediately in the definition cell itself (see Chap. 9, "Graphics").

The *indefinite detail list report page* is the most distinct page type of the three page types found in QueryCalc. Only four columns are used on the report page. You cannot increase or decrease the number of columns, nor can you change the column widths. The columns have pre-defined usages. The first column, column A, is used only for cell equations. The second column, B, is used to the name the extracted or calculated dataitems. Columns C and D are optionally used to define sort order and allocated print width, respectively. Rows are used to define groupings within the report (see Chap. 13, "Indefinite Lists").

When the completed report is to be printed, only the standard spreadsheet summary pages will print as they appear. The graphic definition pages cannot of course be printed directly. Rather the information specified on the graphic definition pages is transformed into the graphs they define. Nor will the indefinite list report pages print as they appear. Although a detail list report is composed on only one spreadsheet page, it may result in several to several hundred actual (paper) pages of output.

A QueryCalc report may be any mixture of the three page types you desire. An example is shown on the previous page. The first two pages are financial summaries while the third page (indefinitely) lists all open accounts. The fourth page is an additional financial summary with an attached graph. The sixth page is a standard assets and liability summary while the seventh page is another detail list listing all current fixed assets. A report as complicated as that shown is never more than a simple combination of simple ideas used over and over again. Each cell on each page type is identical. Most importantly, the procedure to put each report page together is the same. You enter the cell equation, check the results, and interactively repeat the process until you are satisfied.

# Concepts Introduced in Chapter 3

| | |
|---|---|
| SUMMARY REPORT | a standard spreadsheet page. Cells may directly summarize database information. |
| GRAPHICS PAGE | a modified spreadsheet page with a predefined template. Information filling the cells defines the graph. |
| DETAIL LIST REPORT | an indefinite list of items extracted from a database(s), totalled and sorted within groups. |
| NUMERIC CELLS | a standard mathematical equation which may be as simple as a single number. |
| TEXT LABELS | Text written on the "surface" of the spreadsheet. Text labels are non-calculating cells. |
| TEXT EQUATIONS | equations capable of active text reorganization during recalculation |
| QUERY QUESTIONS | database summarizations. |

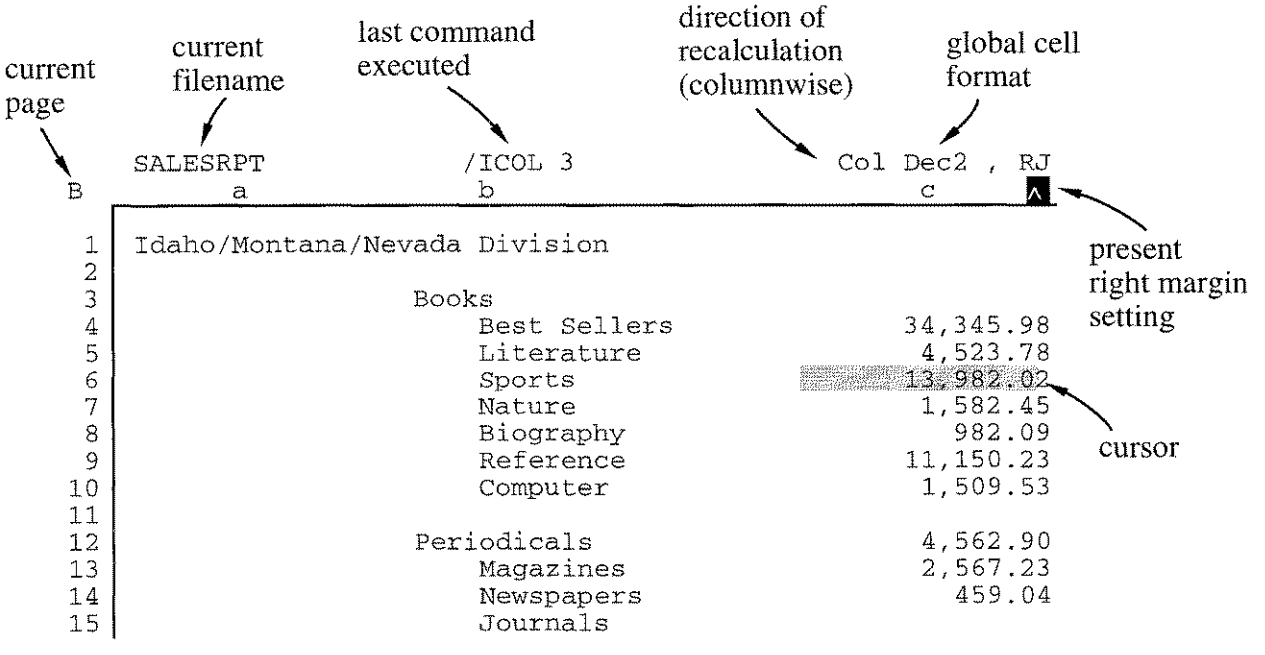
# 4

*Mastering the basic spreadsheet functions*

# Spreadsheet Operations

## Getting Around in QueryCalc

When you first enter QueryCalc, you will see a blank display showing the upper-left section of the first page, consisting of 15 rows. The cell cursor will appear on row 1, column A of page A. You may move the cursor around the spreadsheet in either of two ways. Pressing individual function keys ([F1] to [F8]) will shift the cursor up or down, left or right, jump from the current page to the next or previous pages, or scroll the screen up or down 15 rows at a time, as indicated by the function key labels. You'll find the function keys the easiest way to move around the spreadsheet within a local area.




*The standard features of the QueryCalc screen*

Alternatively, if where you wish to be on the spreadsheet is some distance away, you can type any of the following jump command forms:

| | |
|---|---|
| **/J GD14*** | jumps to page G, column D, row 14 |
| **/J B13** | jumps to column B, row 13 on the current page |
| **/J M** | jumps to page M (column A, row 1) |

Because the width of a column may be adjusted from 0 to 36 characters, you may occasionally jump to a zero-width column. Should you specify a jump to a zero-width column, QueryCalc will automatically move the cursor to the first non-zero column to the right. Because QueryCalc will not let you collapse all of the column widths on a page to zero, you can be sure that the cursor will be visible somewhere on page.

## *A Note on Cell References

Cells in QueryCalc are referenced by a two, three or four-character code indicating the cell's Page, Column, and Row position on the spreadsheet.

| | |
|---|---|
| CR34 | - indicates page C, column R, row 34 |
| G3 | - indicates column G, row 3 on the page you're now on |
| {DI2} | - an *absolute* cell reference to cell DI2 |

A cell reference which contains all three attributes (page, row and column) is said to be *fully qualified*. Optionally, when referring to cells on the current page, the page may be omitted. In this instance, the page value is *implied*. The *case* of the letters in cell references is unimportant: CR34 is the same as cr34.

An *absolute* cell reference is one that will not change when the cell is replicated. *Relative* cell references (cell references without braces surrounding them) will be adjusted by the magnitude of the offset between the cell's original position and its final resting place.

There is some advantage to using *page-implied* cell references for cells which are on the current page. Cell reference processing will be slightly faster. More importantly, you will find that insertion, deletion and replication of pages will generally be easier.

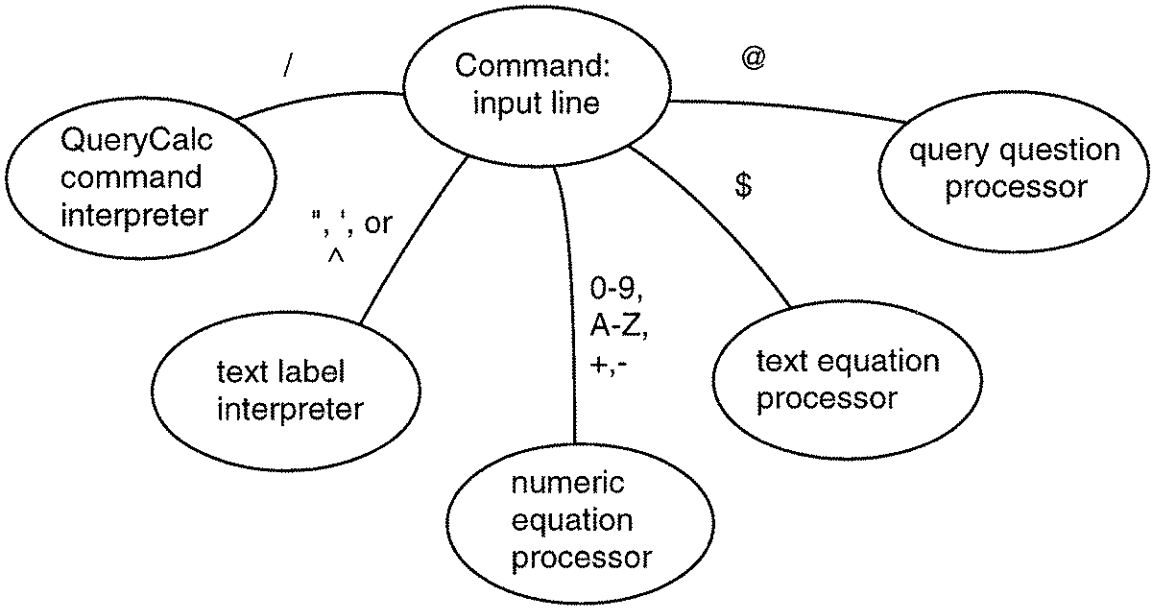
## Entering Text, Equations & Database Queries

Once you've positioned the cursor at the cell of interest, simply begin typing at the *Command:* prompt to enter text, numeric values, equations, or database queries into a cell. The cell highlighted by the cursor is the cell where your entry will appear.

QueryCalc is constructed throughout as a *finite state automaton.* You move from state to state depending upon which input symbol was received. This design philosophy is most evident at the *Command:* line. Every input to QueryCalc is preceeded by a prefix character.


*The prefix character is the key to the way QueryCalc works.* QueryCalc transfers your input to the appropriate subprocessor based on the prefix character that begins your input line.

The five classes of *Command:* line prefixes are shown below:




The most common mistake you'll make when first using QueryCalc will probably be due to either leaving the necessary prefix off or using the wrong prefix. But this phase won't last long. The prefixes used in QueryCalc are common to most spreadsheets and will soon become second-nature.

## Examining and Editing Cells

Examining the contents of a cell is straightforward. Move the cursor to the cell you wish to examine. The cell equation, format, print enhancements (if any) and recalculation errors (if any) will appear in the area just above the command line.

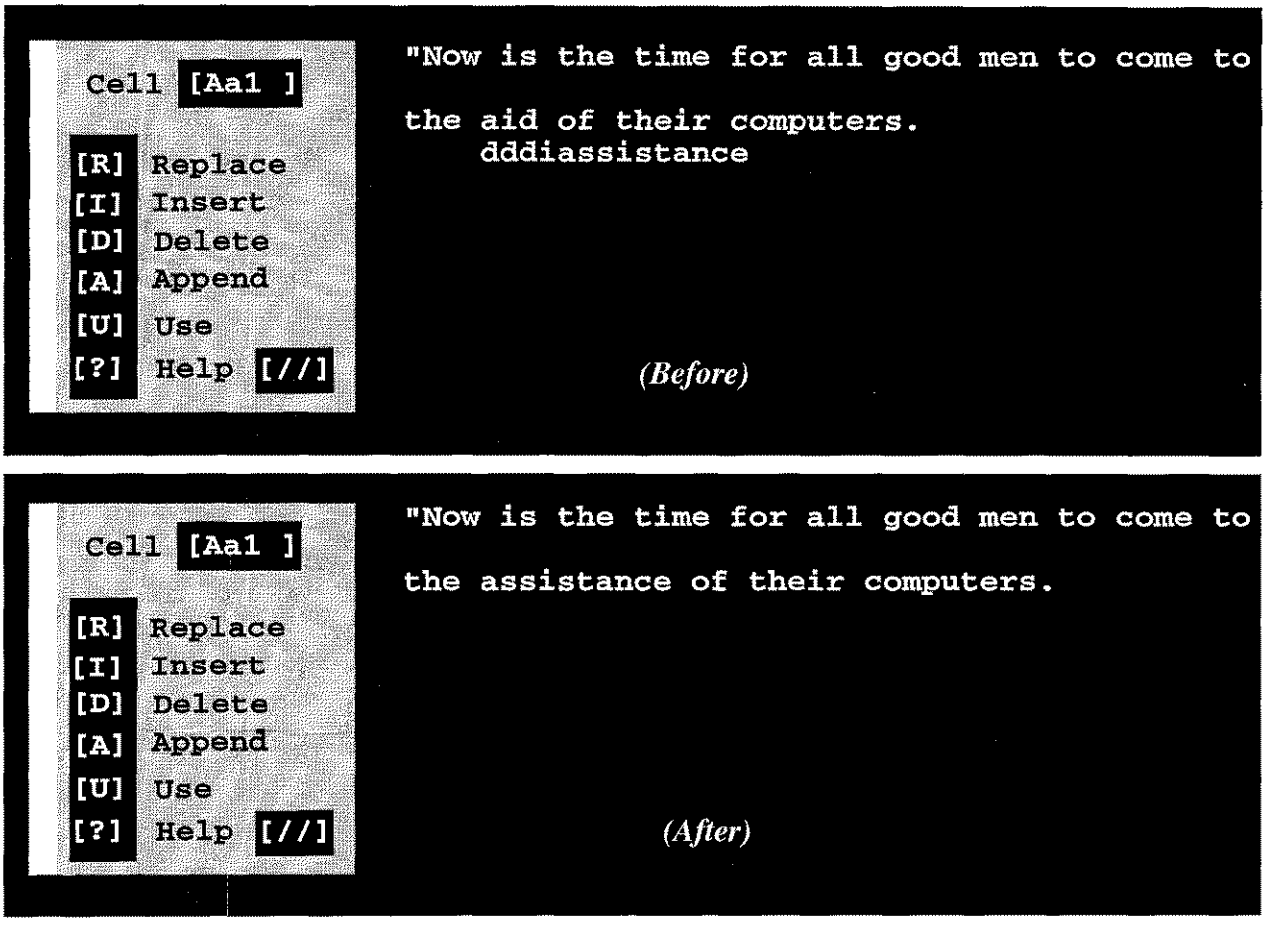
If you make a mistake when entering a cell equation, or wish to change a cell in your report, or redo your last command, you'll want to edit the cell's text. QueryCalc provides three editors for this purpose (which are in reality all the same editor). One editor edits only the contents of a cell. Another is specifically reserved to edit the last entry attempted, regardless of whether it was accepted by QueryCalc's syntax checking routines or not. The third editor allows you to edit the last command typed. The editors are accessed through one of these three commands:

| | |
|---|---|
| **/E** | to edit the contents of a cell |
| **/REDO** | to edit the last entry attempted |
| **/LAST** | to edit the last command given |

To edit a specific cell, move the cursor to the cell of interest and type **/E**. If the cell contains a text, numeric, or query equation or a text label, a window will appear listing the editor commands and the cell contents. If the text you wish to change is on the first line segment of the editor, *press the SPACE bar to move the cursor underneath the first character to be changed. Don't use the arrowed cursor keys when in QueryCalc.* The cursor keys are *local* keys only. They do not transmit to the computer. Although the terminal's cursor will move on the screen with each press of the cursor keys, the HP3000 and QueryCalc are unaware that you have hit any key at all.


NOTE

Once you're underneath the text you wish to modify, you have several choices (indicated by the editor command letters shown in the window). To delete text, press **D**'s under each character to be deleted. To delete and then insert text, press **D**'s under each character to be deleted and then press an **I** followed by the desired new text (as shown on the facing page). To replace text, if the new text does not begin with one of the editor's command letters, you may simply type the new text underneath the text to be replaced. If however your text does begin with a reserved letter, begin your entry with a **R** followed by the desired text. If you are familiar with MPE's :REDO editor or EDIT/3000 on the HP3000, you will immediately recognize the similarity in command forms.




*Using the cell editor to delete and insert new text in one pass*

Two slashes (*//*) will get you out of any QueryCalc subsystem with no effect. The editor follows this convention. A question mark (**?**) will get you help. Preceding your text with the command letter **A** will append new text to the end of the present edit line, regardless of where the cursor is presently positioned. You'll find the append feature useful in adding additional qualifying phrases to a long equation.

The edited line takes effect when you press **U**, for use. The text is interpreted by QueryCalc exactly as if you typed it in at the *Command:* line. The editor will display a cell's contents, up to the cell's maximum of 186 characters. The text may be broken into perhaps as many as four line segments.


NOTE

*If the portion of the text you wish to edit is not on the first segment, press the RETURN key once for each line segment you need to bounce past and then press the SPACE bar to position the cursor underneath the text to be modified.* [Pressing the RETURN key four times (sometimes less) is an easy way to resynchronize the editor screen with the cursor should something go awry.]

## Loading and Saving Files

While you are creating a spreadsheet, type **/SAVE** every so often. Doing this guarantees you won't lose your work. If the file name you wish to use already exits, QueryCalc will ask whether or not you want to purge the existing file with this name and create the new file. This question will occur whether the file is a QueryCalc spreadsheet or any other HP3000 file type. *If you did not know that there was another file in your logon group with this same name, prudent action would suggest that you do not answer YES until you determine what the other file is and how dispensible it might be.*

/SAVE will record the spreadsheet's cells, including all open databases, cell equations, page margins, formatting information, screen color, number representation selected (American or European), terminal width (80 or 132) and all other relevant information in a file within your current account and group. For reasons of simple security, you cannot save your spreadsheets in any other group on the HP3000 other than your own. You may however load a spreadsheet from any group on your machine (or even a remote HP3000), if you have standard MPE access to the necessary file. The SAVE command is like most commands in QueryCalc. You have two ways to use the command. The first is by simply typing /SAVE. Questions will follow asking for additional information (in this case for file name). Alternatively, you may type the command followed immediately by all of the necessary parameters.

Retrieving a spreadsheet is as easy. Type **/LOAD**, optionally followed by the name it was stored under. When a file is /LOADed, QueryCalc automatically restores all the file's ancillary settings (margins, screen color, etc). All of the databases which are open prior to the /LOAD are closed and the databases associated with the new file are opened.

Additional commands related to the files are **/KILL**, used to purge a spreadsheet from the HP3000, and **/INFO**, used to display the cell usage information for the currently loaded spreadsheet. The /KILL command is different from the standard MPE :PURGE command only in that it also automatically updates QueryCalc's directory. Should you purge a file using the MPE command while outside of QueryCalc, the name will remain in QueryCalc's directory as a ghost file. There is no penalty associated with the maintenance of ghost file name other than the obvious confusion it will engender. To exorcise the ghost file name, simply /KILL the ghost file while you are in QueryCalc.

## The File Commands

To load a spreadsheet report:

**/LOAD**
**/LOAD myfile[.group[.account]]**

To save a spreadsheet report:

**/SAVE**
**/SAVE myfile**

To see a listing of the QueryCalc files in your logon group:

**/CAT** *or* **/DIR**

To delete a QueryCalc file:

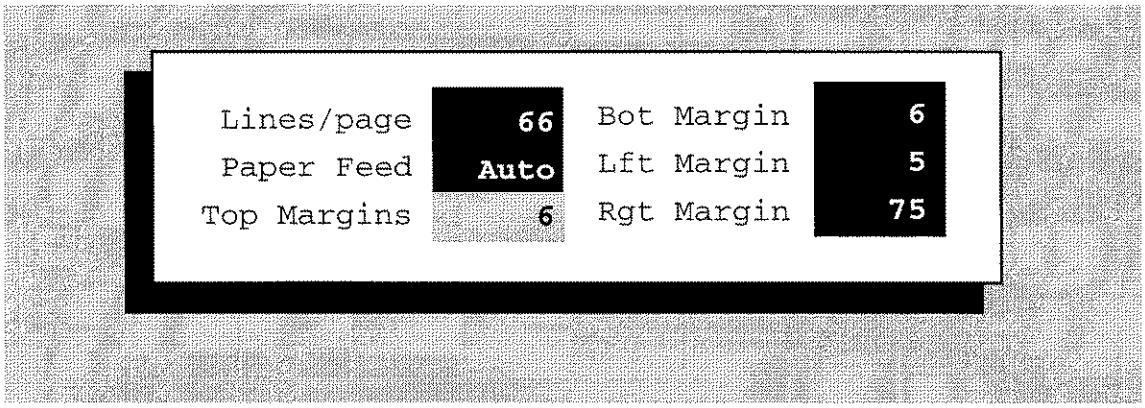
**/KILL**
**/KILL myfile**

To display information about the currently loaded spreadsheet:

**/INFO**

## Setting Page Margins

The size of the print area for each spreadsheet page in a QueryCalc report is set with the **/MARG** command. When the page margins are set in QueryCalc, you have described (1) the size of the paper you're using, (2) the margin sizes you wish to maintain at the top, bottom and sides of the paper, and (3) the number of lines per page your printer will print. A separate screen (a portion of which is shown below) is used to set each of these margin features. The use of the /MARG command is explained fully in Chapter 7, "Printing the Report".



## Every Input is Syntax Checked in QueryCalc

When an equation or a spreadsheet command is typed at the *Command:* line, the input is immediately checked for correctness. Syntax-checking exists to protect you from careless errors. QueryCalc will let you do very little harm to your currently active spreadsheet without warning you first .

But more is to be gained from extensive syntax checking than simple security. When your report is ready to be calculated, you can be virtually certain that the report will execute properly. Syntax-checking often represents a 10-20 times increase in programming productivity.

Only those equations which pass syntax check are entered into a cell. An equation which does not pass syntax check is rejected. More than 500 plainly-worded error messages are associated with the syntax checker. Should your input fail syntax check, the resulting error message should be clear enough to tell you precisely what's wrong with your typed input. Use the REDO editor to modify the equation until it's correct.

Equations which are accepted into a cell are marked either as:

- *recalculating cells*

or

- *non-recalculating cells*

The only cell types which can be *non-recalculating* are text labels and numeric equations which do not contain cell references. Examples are:

```
^Expenses for January, 1997
56*sqr(457+56)
```

All other cell types (query questions, text equations or numeric equations containing cell references) are marked as *recalculating* cells. Recalculating cells will be re-executed every time the spreadsheet is recalculated.


*Equations do not automatically calculate on entry.* A recalculating cell equation which does not end with an "!" is only syntax-checked. Query questions may take substantial time to execute. Once you become familiar with QueryCalc, you will not tend to request immediate calculation for every new query question. *An equation which has been syntax checked but which has not yet been calculated is marked as:*

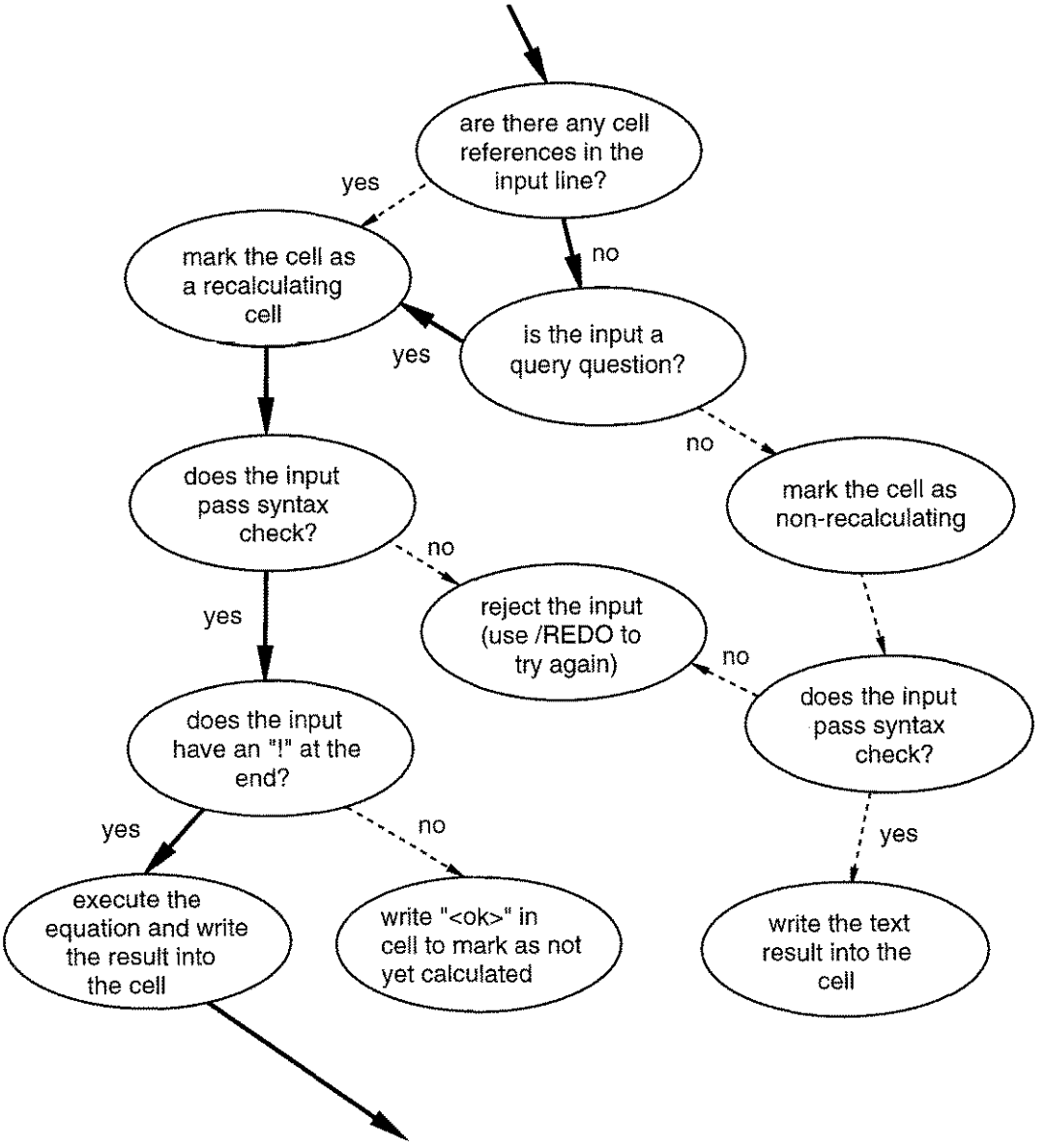
```
"<ok>"
```

## What Happens to a Cell Equation on Input?

The logical flow of syntax checking a cell equation is outlined below. The specific path travelled by the example shown is marked by the bold arrows. Because the query question in this example was terminated with an "!", the query question was calculated immediately on input rather than being deferred until global recalculation.

*Input at the Command line:*

```
@Using invoices, sum of amount when
        category is 501!
```




*Result placed in the cell:*

```
"23,984.97"
```

## Recalculating the Spreadsheet

QueryCalc has two modes of recalculation. The first recalculates only the surface of the spreadsheet. The second recalculates both the spreadsheet and extracts new data from the database(s). Recalculation in either mode may be specified for the entire spreadsheet or a selected range of pages or cells.

---

Command to recalculate the entire report *without* new database inquiry:

**!**

Command to recalculate the entire report *with* new database inquiry:

**!!**

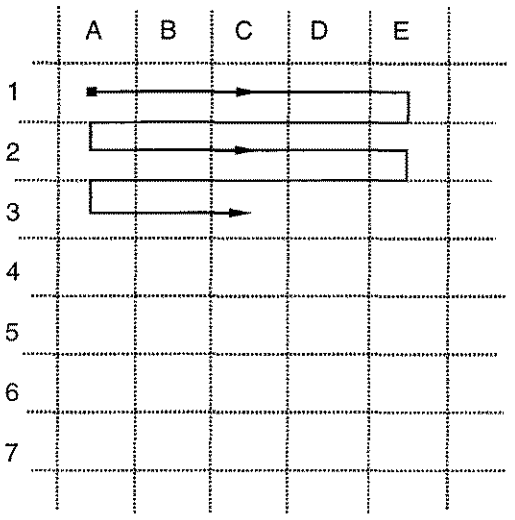
Command to recalculate any portion of the spreadsheet:

| | |
|---|---|
| **!e9,f10:f30** | recalculates only math and text cells in local page cells e9, f10 thru f30 |
| **!A:C,K,N:P** | recalculates only math and text cells in pages A thru C, K, and N thru P |
| **!!M:Q,Z,az5** | recalculates query questions, math and text cells in pages M thru Q, Z and the single cell AZ5. |
| **!!g34** | recalculates the query question in the single cell G34 on the current page |

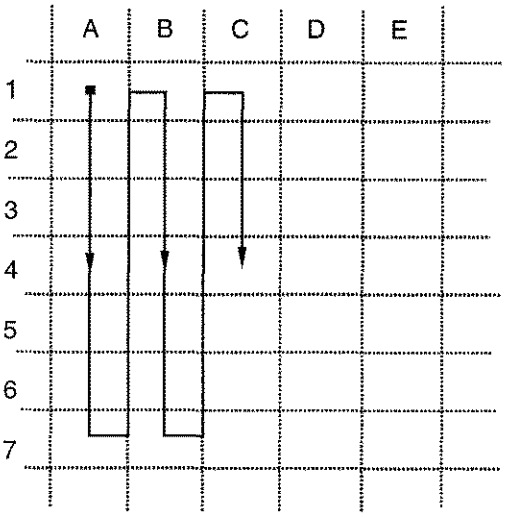
---

The recalculation of any spreadsheet begins at the upper left-hand corner of the spreadsheet and proceeds more or less diagonally down to the lower right-hand corner.

QueryCalc recalculates the spreadsheet in either one of two directions, *row-wise* or *columnwise*. QueryCalc does not employ automatic recalculation. Automatic recalculation would quickly become unacceptable due to the length of execution times of most query questions.




ROW-WISE
RECALCULATION




COLUMN-WISE
RECALCULATION

Database inquiries in QueryCalc may be written so that they are *dependent* upon the results obtained in a previous query question. Dependent query questions offer a simple mechanism to improve the efficiency of data extraction from databases (see Chap. 6, "Query Questions"). The dependency of one query question on another is sequential. On the spreadsheet, cellular position and the recalculation direction dictates the order of execution. If the direction of recalculation is row-wise, dependent query questions should lie basically to the right of the query question they reference. Conversely, if the direction is column-wise, dependent query questions should lie basically underneath the independent cells they reference.

The direction of recalculation may be independently specified for each page of the spreadsheet. The default direction is column-wise for all pages. To change the direction of one or several pages, the command forms are:

| | |
|---|---|
| **/ORDER** | the basic command. Questions requesting more information will follow. |
| **/ORDER ALL C** | sets all pages to column-wise recalculation order. |
| **/ORDER B:G,M R** | sets pages B thru G and M to row-wise recalculation order. |

## Setting Column Widths

The column widths on a page may be individually set to any width between 0 and 36 characters wide. A zero-width column will not print, nor will the cursor "land" on it. A zero-width column is in all other respects a normal column of cells. Its equations will calculate normally and may be referenced as any other cells. The default character width for a blank spreadsheet is 12 characters wide for all columns.

| | |
|---|---|
| **/CWID** | the basic command. Questions requesting more information will follow |
| **/CWID B:G,M,R 10** | sets columns B thru G, M and R to 10 characters wide |

You may examine the current column width settings for a page by typing **/COLS**. The current widths will be printed at the bottom of the page.

*Text that is too wide* for a cell's width will simply be extended rightward into the adjacent cells. If the cells to the right are empty, the text will be displayed. If they are not, the right-lying cells' contents take precedence over any extended text. *Numbers that are too large* are treated differently. If a numeric display is too large for a cell's width, the display you will see will be a series of asterisks:

"**********"

## Screen Settings

Other settable screen attributes are the color of the screen, the display width, and the form of the numeric display (American or European).

| | |
|---|---|
| **/COLOR** | sets the color (black, white, gray) of the border, background and cursor |
| **/80**<br>**/132** | sets terminal to 80 or 132 column display. Only available on newer terminals |
| **/US**<br>**/EUR** | sets numeric displays in common American or European formats |

## Information Requests

A number of informational screens are available. They display current memory usage, current page usage, column widths for the current page, and a calendar.

| | |
|---|---|
| **/INFO** | displays current memory usage and current marging settings |
| **/PAGS** | displays current cell count on each page in QueryCalc |
| **/COLS** | displays column widths for the current page |
| **/CALEN** | displays a Gregorian calendar. The calendar's range is October, 1582 to December, 9999 |
| **/T** | displays current time and date |

## Interrupting QueryCalc

Various processes in QueryCalc, such as spreadsheet recalculations and printing, may be interrupted by pressing CTRL-Y. This is called a *subsystem break*. If you are unfamiliar with the process, press and continue to hold the CTRL key. Then press "Y". The subsystem break has been enabled in QueryCalc everywhere that it was felt that it was necessary. The subsystem break will take effect immediately. However QueryCalc may not come to an immediate stop. It will come to a stop at the earliest opportune (and safe) moment.

Long listings may be temporarily stopped by pressing CTRL-S and resumed by pressing CTRL-Q. These key combinations form what are called the Xon/Xoff transmission protocol characters. (*Xon* means transmission on; *Xoff* means transmission off).

The system break key, BREAK, has been disabled everywhere in QueryCalc. However, you may temporarily return to the MPE operating system by typing **/SYSTEM** at the *Command:* line, as explained on the following page.

## Leaving QueryCalc

QueryCalc may be exited in three ways: (1) to temporarily return to the MPE operating system, (2) to run another program within QueryCalc, and (3) as a complete exit.

---

To momentarily return to MPE, type:

**/SYSTEM***

MPE commands which do not require invoking a new process (such as :LISTF, :PURGE, :BUILD, :HELP, etc.) may be executed.

**/SYSTEM FILE QCLISTF;DEV=103;CCTL***

A one-line version of the /SYSTEM command can be formulated as above. This form is macro-executable. Command returns immediately to QueryCalc on both successful and unsuccessful execution.

To suspend QueryCalc and run another program, type:

**/RUN EDITOR.PUB.SYS***

The name of the program must be qualified, if not in the user's signon group or account.

UDC's (user defined commands) and MPE's run command qualifiers are not supported.

Upon completion of the invoked program, QueryCalc resumes at the point of suspension.

To fully exit QueryCalc, type:

**/EXIT**

---

* *Note:* The /SYSTEM and /RUN commands may be disallowed by your HP3000 System Manager.

# Concepts Introduced in Chapter 4

| | |
|---|---|
| CURSOR | a movable marker on the spreadsheet indicating which cell is the active cell |
| CELL ADDRESS | a one- or two-letter/numeric address specifying the page, column and row |
| ABSOLUTE CELL | a cell address that does not change when replicated |
| RELATIVE CELL | a cell address that is altered when replicated. The cell reference is altered by the amount of the displacement from its original position |
| PREFIX | the initial character of an input at the *Command:* line. The prefix specifies how the text is to be interpreted |
| CELL EDITING | the capacity to easily modify the contents of a cell (or last attempted input) |
| SYNTAX-CHECK | a procedure where every input is checked for correctness before entry onto the spreadsheet. Syntax-checking is one of the primary methods to improve user productivity |
| CALC ORDER | the direction of recalculation of the spreadsheet. The choices are either rowwise or columnwise. Each page may be independently specified |

# InterChapter Reminder

The only way QueryCalc will become useful to you is for you to know what's in your databases. That means understanding how a database on the HP3000 is put together and what each dataitem in your own databases means. This advice doesn't apply solely to QueryCalc. It's only common sense. However, this is not difficult. If necessary, review Chapter 2. And then ask someone about the databases on your machine. If you are confused and you can't get help locally or from your applications program supplier(s), call us. We'll do our best to help. Our telephone numbers are:

(800) AICS-INC *(United States)*
(505) 524-9800 *(elsewhere)*
(505) 526-4700 *(FAX)*

Our hours are 8AM to 5PM, Mountain Time, weekdays. To relate our time zone to yours, if you are in London, subtract 7 hours from your local time. If you are in Sydney, add 7 hours to your local time.

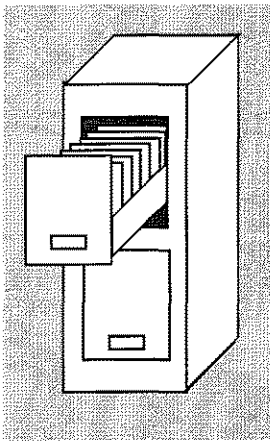
# 6

*Creating a summary of dataitem(s) from a single dataset*

# Query Questions

## The Single Dataset Query Question

The query question which summarizes data from a single dataset is the heart of QueryCalc. Although a variety of methods exist in QueryCalc to simultaneously extract data from multiple datasets, databases and multiple HP3000's (as explained in Chapters 10, 11, and 13), each of these methods is built around the question that gets data from a single dataset.



The single-dataset query question will always look something like this:

```
@Using invoices, sum of amount when
 category is 501 and date > 930101
```

The word "when" is the keyword in a query question. The phrase to the left of "when" specifies what statistics are to be summarized. The phrase(s) following "when" specify under what conditions retrieved records are to be added into the summarization. These restrictions are called the *qualifying phrase(s)*. The records in the dataset which pass these restrictions are commonly called the *qualifying entries*.

The statistics which may be summarized in a query question are these:

```
Sum of ....
Avg of ....
Max of ....
Min of ....
Var of ....
Dev of ...
Val of ....
```

The last statistic in the list should be read as: "get me the first value of...". This statistic is different than the others. The search of a dataset stops immediately once a single record has been qualified. The value returned is thus the "First Value". This form of query question will be predominantely used in dataset "rereadings" (an idea which will be explained shortly).

## The Query Question Explained

A complete query question will always look basically like this:

```
@Using qcdemo.invoices, avg of amount
when jobnum ib 8000,8100 and date>=19950601
```

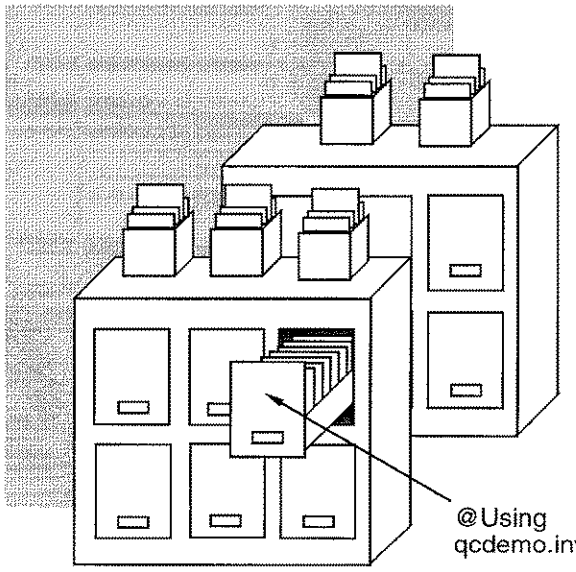
In order to explain what's possible in a query question, each phrase in the query question will be described in detail in the next few pages.

## The "Using..." Phrase

The initial "Using..." phrase at the beginning of a query question specifies which dataset in which database is to be searched. The phrase may be typed in any of four different fashions. Three of them are:

```
@Using database.dataset, ....
@Using database, ....
@Using dataset, ....
```

*Or the phrase may be left off altogether.* If the phrase is left off, QueryCalc will look only in the last-used database (called the *default* database). A query question is not complete until it is fully specified as to which database and dataset is to be searched. QueryCalc will automatically attempt to fill in whatever information you omit. If you specify only the database, as in the second example, QueryCalc will change *default* databases and look only there. If no single dataset contains all of these items, QueryCalc will announce that as an error. Should more than one dataset possess all of the items, QueryCalc will present you with a list of choices.




*The "@Using ..." phrase determines which database (file cabinet) and which dataset (file drawer) is to be searched by the query question.*

## The Item(s) to be Summarized may be Specified as an Equation

The item to be summarized in the query question can either be a single dataitem, as in this example:

```
@Using db.ds, sum of amount
```

or it can be a calculated combination of various dataitems:

```
@Using db.ds, sum of
amount*workcomplete/jobcost
```

Because of this feature in QueryCalc, the object of the summarization is called a *dataitem equation*. Both the item *amount* by itself and the phrase *amount*workcomplete/jobcost* are dataitem equations. The rules for use of dataitem equations are these:

1. All of the dataitems must come from the same dataset.

2. Only the five basic mathematical operators ( + , - , * , / , ^ ) are allowed. No functions are permitted. Function manipulation of the retrieved items can be handled using the user-defined query functions (see Chap. 11).

3. Parentheses are not allowed. (Parentheses are used for other purposes in specifying dataitems.) If you need to enter an equation such as

   ```
   A*(B+C)
   ```

   where A, B, and C are dataitem names, multiply the equation through and enter it as

   ```
   A*B+A*C
   ```

A common practice used by many database programmers is to create dataitem names which use some or all of the mathematical characters, but which were not meant to be interpreted mathematically. Hyphens are especially commonly used.

## Resolving Ambiguities in Dataitem Names

How can you tell what's what in a dataitem equation such as this:

```
@avg of current-age-age-operated-on
```

Surprisingly, QueryCalc can often work these equations out by itself. QueryCalc does this by examining each part of the dataitem equation piece by piece. If the specified database contains the item *current*, the next hyphen must be a minus sign. If *current* is not a dataitem, *current-age* is tried. If this dataitem name exists, the following hyphen is interpreted as a minus sign. This step-wise process of parsing dataitem names will fail however if there are two dataitems in the database, one named *current* and the other *current-age*. The second dataitem will never be seen and equation processor will unsucessfully attempt to subtract *age-age-operated-on* from *current*.

You can remove any ambiguities in your specified dataitem equations by placing backslashes (" \ ") around the dataitem names:

```
@avg of \current-age\-\age-operated-on\
```

Dataitem names specified in this manner are unambiguous. An added advantage of the backslashes is that the processing of the query question is slightly faster.

## The Qualifying Phrases

The qualifying phrase(s) following the word "when" in the query question determines which records are to be added into the accruing statistics during the search of the specified dataset. The word "when" is not always required. A query such as the following:

```
@Using invoices, sum of amount
```

will simply qualify every record in the dataset (but because no search items were specified to be matched, a serial search will be required). A more standard query question would look like this:

```
@Using invoices, sum of amount
    when category is 501
```

This query will extract only a subset of the records in the dataset. If the dataitem *category* is a search item, the retrieval time will generally be quite quick.

A qualifying dataitem may also be a dataitem equation, as in this example:

```
@Using invoices, sum of amount when
    amount-balance*1.25 > 10000
```

## The Three Parts of a Qualifying Phrase

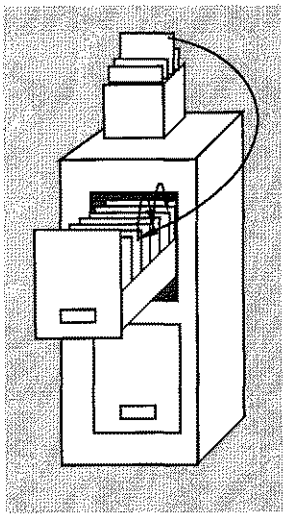
Every query qualifying phrase consists of three parts: (1) *a dataitem (or datatitem equation),* (2) *a relational operator* ("*relop*" for short), and (3) *a pattern* to be matched in the dataset's records.

| *dataitem* | *relop* | *"pattern"* |
|---|---|---|
| amount | > | 10000 |

The qualifying value ("10000") is called a "pattern" to emphasize the nature of a match in a database. You don't so much match numbers or letters when you qualify records as you match *bit patterns (strings of 1's and 0's).* Being a simple machine, that's all the computer can look for. Text forms a certain set of bit patterns, numbers another set. QueryCalc automatically generates the proper bit patterns based on the dataitem type being matched.

## Chained or Serial Search?

The qualifying phrases determine whether or not the search is to be *chained* or *serial.* For a chained search to be possible in IMAGE, at least one dataitem in the qualifying phrase must be (1) a search item, (2) matched with "=" or *is* relop, and (3) specified as a whole (no partial substrings).



```
@sum of amount when
category is 501 and date > 19930101
```

The dataitem *category* would meet those criteria in this example. *Date* would not qualify for a chained search in IMAGE, even if it were a search item.

The rules are slightly different for KSAM databases. KSAM is less restrictive about a precise pattern match than IMAGE is. For a keyed search to be possible in KSAM, at least one dataitem in the qualifying phrase must be (1) a search item, (2) the relop may be any relational operator other than *not equals* ("<>"), and (3), if a partial string is to be used, it must begin with the first character.

Otherwise, there is no difference between query questions for IMAGE and KSAM databases.

## The Seven Possible Relational Operators

There are only seven relational operators ("relops") possible in a query question: (1) equals, (2) greater than, (3) less than, (4) greater than or equals, (5) less than or equals, (6) not equals, and (7) "is between". The relational operators may be written in a query question in any of the following ways:

```
=    is   ie    eq
>    gt   igt
<    lt   ilt
>=   =>   ge    ige    inlt
<=   =<   le    ile    ingt
<>   #    ne    ine    isnot
ib
```

These are the most common standard representations for the seven possible relational operators. All of the relop forms on a row are equivalent. However, there is a restriction with the use of some of the relops. Those relops which start with a letter, or the"#" symbol, must be separated by a space from the dataitem name and the pattern to be matched. The reason is alpha characters and the "#" symbol are legitimate characters in IMAGE dataitem names. Without spaces, reliable sentence parsing becomes impossible, as in this case:

```
@sum of amount when ssn##452567191
```

QueryCalc will announce its inability to properly parse such a condition. The query question must be rewritten in this fashion:

```
@sum of amount when ssn# # 452567191
```

The qualifying phrase may be run together, without spaces, if the symbolic relops (=, >, <, etc.) are used. These symbols are illegal characters in IMAGE dataitem names, thus there is no ambiguity as to what part of the text is the dataitem name and what is not. An example:

```
@sum of amount when ssn#<>45267191
```

## Specifying the Pattern to be Matched

A qualifying phrase may be written as simply as this:

```
... when state is CA
```

where the pattern to be matched is CA. Quotes are not required around the pattern if the value to be matched is composed of only alphanumeric characters, absent of any form of punctuation or spacing. Quotes *are* required to surround the pattern to be matched if non-alphanumeric characters appear as part of the pattern, as in this example:

```
... when employee is "Smith, Joe"
```

The dataitems in these two examples, *state* and *employee*, are text dataitems. If a dataitem is defined as an "X"-type text field in the database, both upper and lower case text may appear in the dataitem's values (see Chap. 2). If the text dataitem is defined to be a "U"-type field, then only upper-case values are expected to appear in the dataitem's values. QueryCalc will automatically upshift all pattern text to match a "U"-type dataitem. *If however the dataitem is an "X"-type field, QueryCalc will leave the pattern to be matched as you typed it. Therefore you must be careful to specify a text pattern correctly, letter for letter, in its correct lettercase, if a match is to be found in the database.*

```
... when idnumber is 134527
... when pressure ib 1e5,7.3e6
```

Numbers require no such consideration. Regardless of how the number is stored in the database or how you represent the number in your query question, QueryCalc will automatically build the proper bit pattern to match the datatype used in the database.

## Matching a Pattern Off of the Spreadsheet

Much of QueryCalc's power is derived from its ability to extract pattern values off of the spreadsheet. Brackets ([...]) are used in a query question to form a "window" back into the spreadsheet's equation interpreters so that you can synthesize a pattern to be matched.

Examples of common numeric equation patterns are:

```
.... when productno is [ab12]
.... when idnumber is [1900+g13+g17/10]
... when startmonth is [190000+{az5}]
```

When a query question containing equations is replicated, the cell addresses used in the equations will be automatically adjusted by the amount of displacement traveled on the spreadsheet. The exception are cells surrounded by braces, which indicate absolute cell references (the third example on the previous page).

Text equations may be similarly created:

```
.... when name is [$g34]
.... when date ib [$"19"+{az5}],[$"19"+{az6}]
... when product=[$if$(ab7>ab8,"MRJ1501",g34)]
```

Any numeric or text equation that can be placed in a cell on the spreadsheet may appear with the brackets of a query question. In the third example, the pattern to be matched is made conditional on the relationship of two cells, Ab7 and Ab8.

## Subitems & Substrings†

Patterns may similarly be matched to subitems in a dataitem array. The subitem is referenced by a *single* index:

```
... month-profit(4) > 12000
... store-number(16) = [$r45]
```

The dataitem may be either text or numeric. It is of course necessary that the pattern to be matched be the same data type as the dataitem.

Substrings within a text dataitem may also be matched. Substrings are indicated by *two* indexes:

```
... date(3,5) = 051
```

The first index indicates the *start* position of the first character in the dataitem's text string; the second indicates the *stop* position. The example shown

---

†A *string* is simply a string of *text* characters. This sentence is called a string in the common parlance of programming. A *substring* is a specified subset of the whole string, with specific start and stop character positions. *Subitems* are different. A subitem is part of a *dataitem array*. Simple (non-arrayed) dataitems are written as I1, R2, Z10, X20, etc. Arrays are written as 30X6, 20R2, 10Z6, etc. The simplest way to imagine an array is think of post office boxes. All of the boxes are given the same (dataitem) name. It's the *index* number that is attached to the dataitem name that allows you to select the proper box. In QueryCalc, if you leave the subitem index off, you select data from the first subitem.

would declare a match whenever *date* equaled any of the following values: *840513, 20051012, MY05167.*

Matches searching for substrings within a subitem are indicated by *three* indices:

```
... acctcode(4,5,8) is GR7E
```

The first index indicates the *subitem.* The second and third indexes indicate the *start* and *stop* positions within the character string. This form of subitem matching can be used only with text dataitems.

All of the indexing values for a substring or subitem may be taken off of the spreadsheet, and thus be made variable rather than remain fixed values. The previous example could be rewritten as:

```
... when acctcode(c5,c6,c7) is GR7E
```

where cells C5, C6, and C7 contain the desired indexes. *The index values may only be cell references, not equations.* If the index value is to be calculated, it must be calculated in the referenced cell. An example which would violate the proper use of indices is:

```
... acctcode(c1*4+2,1,4)   (illegal)
```

Otherwise, fixed (constant) values, relative cell addresses, and absolute cell addresses may be freely intermixed:

```
... acctcode(c1,1,{d5})
```

## Text Matches Using Wildcards "@"

For text dataitems, wild cards may also be used:

```
... name is @ote
... name is ote@
... name is @ote@
```

The first example would find all name values which end in "ote". The second would find those that began with "ote". The third would find those that contained the string "ote" anywhere in their entry. The match is *case sensitive* for X-type text dataitems. It is *case insensitive* for U-type text dataitems.

# Table 6.1. Match Patterns Reviewed

| | |
|---|---|
| 1. *state is CA* | The pattern to be matched is CA. No quotes are required if there are no spaces or punctuation marks in the pattern value. |
| 2. *employee is "Smith, Joe"* | Quotes are required to surround the pattern if non-alphanumeric symbols (anything other 0-9,a-z) are used. If the IMAGE dataitem to be matched is a "U-type" text dataitem, the pattern will be upshifted to match the database entries. |
| 3. *productno is [ab12]*<br>*name is [$g34]* | Brackets indicate that the pattern to be matched will come off of the spreadsheet. In the first example, productno is matched with the numeric value taken from cell Ab12). A text dataitem must be matched with a text equation. In the second example, *name* is matched with the text value from cell G34. |
| 4. *acctcode(5) is P* | A dataitem which contains subitems is referenced with *one* index. If no subitem is specified for a subitemed array, the first subitem is assumed. |
| 5. *productnum(5,8) is [$CG7]* | A substring of a text dataitem (either X or U type) can be matched using *two* indexes. The indexes indicate that the substring to be matched begins with the fifth character and ends with the eighth. |
| 6. *@val of acctcode(9,3,5) when productnum(5,8) is 3415* | A substring of a subitem may be matched with *three* indexes. The substring of characters residing in character positions 3 to 5 of the 9th subitem of *acctcode* will be the value returned when the indicated substring of *productnum* equals 3415. |
| 7. *@val of month-profit(d5) when productnum(1,{d6}) is [$g34]* | Cell references (but not equations) may be used in place of fixed values for any of the indexing values. The cell references may be either *relative* or *absolute*. |
| 8. *name is Hol@*<br>*name is @ght*<br>*name is @erin@* | "Wild cards" (@) may be used in text string matches. In the first example, any text value beginning with "Hol" will qualify. In the second, any text ending-with "ght" will qualify. In the third example, the string of characters "erin" anywhere in the text entry will qualify. |

## Summing Subitems with the ":" Operator

For numeric items in a subitemed array, QueryCalc offers an easy way to sum a range of subitems using the ":" operator:

```
@sum of month-profit(1:6) when ....
```

Inserting a ":" between two subitem indexes is equivalent to summing all of the *month-profit* subitem values from the first month to the sixth, as shown:

```
@sum of month-profit(1)+month-profit(2)
    +month-profit(3)+month-profit(4)
+month-profit(5)+month-profit(6) when ....
```

Either or both of the *start* and *stop* subitems in the summation may be made cell references:

```
@sum of month-profit(1:d5) when ....
```

Using cell references for the indexes provides an easy mechanism for *rolling calculations*. For example, if the cell D5 contained an equation which automatically calculated the month number relative to the fiscal year, the year-to-date sum of an array could be calculated automatically.

## Using And's & Or's in the Qualifying Phrase

Up to 20 dataitem qualifying phrases may be appended together in one query question using "and's" and "or's".

```
@find when jobnum=8404 and date>850000
    or jobnum=8405 and date>850601
or jobnum ib 8406,8499 and contractamount>100000
                and date>850000
    or jobnum=8501 or jobnum=8502
```

The "and" takes logical precedence over the "or". That is, all of the "and'ed" items before the first "or" are evaluated as a unit. If the first "and" phrase is found to be true, the record is accepted. If any one of the "and" matches is not true in the first phrase, the second series of "and's" beyond the "or" is tried. The process is repeated until one complete "and" phrase is found to be true or the end of the sentence is reached. If none of the "and'ed" phrases are found to be true, the record is rejected.

## Implicit OR Lists

Because of their use elsewhere in query questions, *parentheses* cannot be used to group "and" and "or" phrases. The alternative is to use QueryCalc's *implicit or* lists. Virtually any query question can be formulated through the use of implicit or's.

```
@sum of hours*1.31+overtime
when facility is 701,711,734,707,749
and unit-number is 3409,3511,3613,4303
and job-number is 673,812,556
```

Up to 10 items may be specified in a list. The match patterns may be fully specified, incompletely specified (through the use of wild cards), taken off of the spreadsheet, or synthesized with an equation:

```
@find when last-name is
SMITH, ROBERT@,[$a3],[$a4+"GER"],@TON
```

## Dependent Query Questions

All of the query question forms discussed to this point have been *independent* query questions. That is, regardless of where the query questions are placed on the spreadsheet, the answers they extract will be independent of all prior queries.

QueryCalc also contains two related query question forms which are *dependent* on the last-asked standard query question. The forms are:

```
@ucs   (using current statistics)
@rereading, sum of ....
```

## The @UCS Form

When a standard query question calculates a requested statistic (sum, avg, max, etc.), all of the statistics are actually calculated. Thus it is a no-cost query to ask for the other statistical values. The calculated statistics may be requested by using any of these @UCS (*using current statistics*) forms:

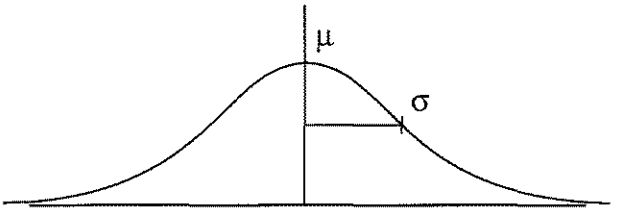
```
@ucs, sum          @ucs, avg
@ucs, dev          @ucs, var
@ucs, max          @ucs, min
@ucs, num          @ucs, pct
```

The statistic "Pct" returns the percentage of those records searched which were qualified. If the search were *serial*, the percentage would be of those

records which qualified out of the entire dataset. If the search were *chained*, the statistic would be the percentage that qualified on the chain.

Frequently, information is required to describe not merely a single statistic such as the sum or average, but instead the how the *population* of entries is distributed. Three statistics describe a Gaussian (Normal) distribution:

$n$, the number of entries qualified,
$\mu$, the mean (or average),
and
$\sigma$, the standard deviation.




*A standard "bell"-shaped Gaussian (Normal) distribution.*

These three statistics may be retrieved in QueryCalc by placing the following equations in cells immediately after a standard query question:

```
@ucs, num     (n)
@ucs, avg     (μ)
@ucs, dev†    (σ)
```

## The Recalculation Order is Important

If the direction of recalculation on the spreadsheet is *row-wise*, then the subsequent dependent query questions should lie somewhere to the right of the last-asked standard query question. If the direction is *column-wise*, then the dependent query questions should appear in the column below and close to the standard query question they reference.

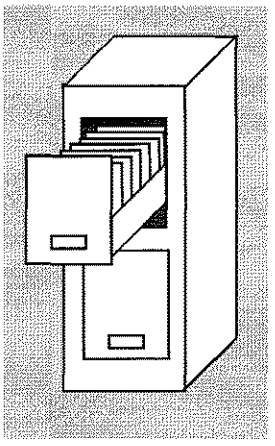
The calculated statistics (called the *current statistics*) remain in effect until either: (1) a new standalone query question, (2) text or (3) numeric equation is executed. Each of these cell types either clears the current statistics or resets them to new values. Intervening blank cells and text labels however have no effect on the current statistics.

---

†Standard deviation is the square root of the variance, which is also calculated (@ucs, var). The calculated variance is corrected by the factor $n/(n\text{-}1)$, which is commonly called the *unbiased estimator.*

## When a Record is Qualified, It's Marked

When a record is accepted into the accruing statistics of a standard, independent query question, its record number is recorded in a temporary file. *Marking* the records in this manner is a little like dog-earing all of the qualified file folders in the file drawer so that you can find them again.



Qualified Record Numbers

| | | |
|---|---|---|
| 12 | 213 | 363 |
| 25 | 215 | 373 |
| 39 | 217 | 374 |
| 67 | 218 | 375 |
| 68 | 223 | 382 |
| 73 | 245 | 423 |
| 76 | 247 | 456 |
| 79 | 267 | 512 |
| 80 | 289 | 523 |
| 103 | 301 | 578 |
| 104 | 303 | 580 |
| 156 | 356 | • |
| 185 | 357 | • |

## The @REREAD Form

The marked records allow the use of the second form of dependent query question: the *reread* of data from a qualified sublist. Rereading data from a qualified sublist often represents a substantial performance improvement over retrieving the data again. The form of the query question is:

```
@rereading, sum of balance
@rereading, avg of vacation when regular>=40.0
```

The first example will reread all of the last-qualified records and return the sum of the dataitem *balance*. The second example will reread all of the last-qualified records and return the average of the *vacation* days for only those records which had a *regular* hours greater than or equal to 40.

Rereading a previously qualified list using the @REREAD does not alter the list of qualified records. The current statistics are changed, however, to the new values calculated.

The @REREAD is also valuable in retrieving multiple dataitem values when one record has been isolated, such as obtaining the name, address, city and state of a single person. Dependent @REREAD's would be placed directly after a qualifying query question, as shown in the example at the top of the facing page:

```
@using employees, val of lname
     when socsecnum is [$g34]
@rereading, val of fname
@rereading, val of address
@rereading, val of city
@rereading, val of state
@rereading, val of zip
```

The record of interest is loaded into the HP3000's main memory with the execution of the first query question. Each @REREADING is a low-cost method of retrieving additional information from the same record. Finding the correct record on the HP3000's disc drives is the most "expensive" part of retrieving data. Once the record is in memory, it should be used to the greatest extent possible.

## Multilevel Rereads (Rereading Subsets)

In contrast to the single-record rereads just described, multilevel rereads allow you to progressively isolate records as subsets, thereby minimizing the need for repetitive serial searches. The technique is especially valuable for filling in tables of calculated values which are derived from a single dataset, and is discussed in detail in Chapter 7, "Summary Reports."

The syntax of a multilevel reread is:

```
@level 3 rereading, sum of amount
     when jobnum < 8500
```

where the level may range from 1 to 9.

Each @REREADING rereads the records previously found at the specified level and creates a new list of records one level down, based on the qualifying criteria of the rereading query. @LEVEL 1 REREADING is identical to the standard @REREADING.

Multilevel rereads are a simple but powerful search acceleration mechanism. The speed advantage occurs because each progressively lower-level reread must now read only a fraction of the qualifying records. But just as important, the original source pool of records is not discarded and does not need to be rebuilt by another serial search. A new subset of records can be redefined at any level by returning to that level and asking a new @LEVEL x REREADING query question.

## Query Forms Which Merely Mark Records

All of the query questions to this point in the chapter have summarized values taken from records in a dataset. There are three query question forms which do not create summaries of values. Rather, they merely mark the records so that they may be reread. The form of these query questions is:

```
@find when state is CA
@num when jobnum is 8404 and date>950000
@pct when keyname ib G,M
```

The first two forms, @FIND and @NUM, are completely synonymous. You should read the query questions as either "Find me all of the records whose state is California" or "Get me the number of records when the job number is 8404 and the date is in the year 1995 or greater." The value that is returned to the cell using either of these forms is the *number* of records which were found to qualify.

The third form, @PCT, similarly marks all of the records that meet the qualifying conditions, but it returns the *percentage* of records which qualified. If the search was a serial search, the percentage will be the percentage of qualifying records in the entire dataset. If the search was a chained search, the number returned will be the percentage of qualifying records on the chain.

While the utility of only marking records and not returning a summary value may not be obvious at first, the @FIND query question form will prove to be invaluable in user-defined query functions, "UDQF"s (Chap.11), and in detail list reports (Chap.13).

## Viewing the Qualifying Records

All query question forms (other than "using current stats") mark the records which pass the qualifying conditions. These individual records can be viewed after executing a query question by typing @SHOW. The @SHOW command reads down the list of marked records and displays the individual record values, one-by-one. To advance the display to the next record, press the RETURN key. You may stop the list prematurely by typing either Control-Y, " // ", or pressing the [F8] key.

Showing the qualifying records in this way allows you to quickly determine that the records you're finding are the records you want. Various command forms for the @SHOW are shown on the facing page:

# The @SHOW Command

To show records found during the last query:

**@SHOW** The show command shows all of the dataitems for all of the qualifying records found during the last database query.

**@SHOW PRODUCTNO,CITY,STATE**

A subset of the dataitems found during the last database query can also be specified to be shown.

To show randomly selected records in a specified dataset:

**@SHOW INVOICES**
**@SHOW INVOICES: AMOUNT, DATE**

Either all or a selected few of the dataitems will be shown for the dataset invoices, record-by-record, in the currently defined (default) database.

**@SHOW QCDEMO.INVOICES**
**@SHOW QCDEMO.INVOICES: AMOUNT, DATE**

Randomly chosen records in a dataset in any open database may also be shown. The specified database becomes the default database.

To show the contents of a search list:

**@SHOW !A** The dataitem values currently held in the specified search list will be displayed.

To show the list of currently open databases:

**@SHOWDB** The currently open databases are displayed. The default database is shown with an arrow ("<---<<").

## Statistical Sampling

Two clauses may be appended to any query question which modify the sampling nature of the query question. The reasons for sampling less than every qualifying record are:

1. The statistical sampling of recorded data for scientific and engineering purposes.

2. To simply limit the number of records drawn during the initial phases of putting a report together.

The modifying clauses are called LIMIT and SAMPLE. They are used in this fashion:

```
@Using database.dataset, avg of thickness
        when jobnum is 8404 and
 machinenum is 15061;limit=300;sample=.4
```

The two clauses are independent. They may be appended to a query question individually or in concert. The order of specification is not important. The LIMIT clause limits the number of records found. The SAMPLE clause specifies the sample frequency. A sample frequency of 0.4 indicates that approximately 40% of the qualifying records will be selected.

If both clauses are added to a query question, the order of execution is as follows: the record is first qualified for acceptance on the basis of whether or not it meets the standard conditions specified in the qualifying phrase. If the record is acceptable, and would normally be added in, a coin is tossed. If the result of the coin flip says that the record should be added in, the limit value is checked. If the number of records is less than or equal to the limit, the record is included in the accruing statistics. If this newest record hits the limit, the query search quits.

If only the SAMPLE clause is used, the entire set of records is searched. (Which records constitute the searched set is determined by the type of search employed, as it is in all query questions. The searched set may be either the entire dataset if the search is a serial search, or it may be only the set of records linked together by a common key value.) The selection of records is random. If you were to search a set of 100 records with a sampling frequency of 0.4, you will not necessarily get exactly 40 records. You may, on any one pass, see 38, 39, 40, 41 , or 42 records. This random-

ness is part of the nature of statistical sampling. If you desire equal sample sizes, the LIMIT clause allows you to specify the sample size. There is an important caveat to be considered, however, when both clauses are used simultaneously. This caveat is especially important when pseudoreplicating your data into distinct statistical "trials".

QueryCalc always searches a set in forward-read order. Normally, this order is indicative of the chronological order that the data has been entered (but not always). The first records are generally the oldest; the last records the newest. If the number of records in the set to be searched is 1000, and you specify a limit of 100 and a sample frequency of 0.3, the query will be satisfied by about the 300th record. Repeating the same query question again will randomly select a slightly different 100 records. But you are only reading the first third of the data. It is important to understand that you may be introducing some bias into your samples when you restrict the sample size. Earlier records may be somehow different from later ones.

## Pseudo-Replicating Data

H.G. Wells said that there are only three kinds of liars: (1) liars, (2) damn liars, and (3) statisticians. It's very easy to mislead someone else, and more importantly, yourself with statistics. Virtually everyone who has ever drowned in a river, has drowned in a river whose average depth was about six inches. A reported average without an accompanying variance is meaningless. Processes which generate identical averages may not be representative of the same physical process. One process may generate a mean with very little statistical variance. Such a process is highly predictable. Another process with exactly the same measured mean may have great variance, and thus be basically unpredictable.

*Pseudoreplication* is a simple statistical technique to determine how uniform your data truly is. Calculating the average and deviation of an entire set of records may be misleading if the process which produced the data is not particularly uniform. In the case of measuring the depth of a river, most measurements may be relatively shallow, but a few may indicate great depth. These exceptions are called *outliers*. A single average and deviation may not provide a true picture of the nature of the data. By dividing the sample set up into 10 approx. equal-sized subsets (`sample=0.1`), and recalculating the average and deviation 10 times, you can tell just by looking at the results if your data is uniform or not. Outlier data will randomly appear in some data subsets, but not others. Because of the smaller sample sizes, these outliers will tend to scatter the resulting 10 averages.

## Search Lists

The final query question form is the @STORE list. This query question form is important enough to be explained in detail (Chap. 10, "Search Sets"), so I'll only briefly mention its capabilities here. The @STORE form allows the creation of a set of search item values, which share a common membership in a class. The basic form of the @STORE query question is:

```
@using employees, store in !m socsecnum
        when state is VT
```

This query question will create a search list named M which will contain all of the social security numbers for those people whose home state of residence is Vermont. Twenty-six such search lists, labeled A to Z, may be created. *The search list is simply a list of search item values. It belongs to no dataset or database.* This independence is what gives the search list its value. You may view the contents of a search list by typing:

```
@show !m
```

You will notice that when you display a list, the search item values will be alphabetized. The alphabetization of the search list results from QueryCalc first sorting the list of retrieved values, and then eliminating all duplicate entries. *Search lists will always be composed of unique entries.*

Search lists are used in standard query questions in place of a single search item value. Instead of asking:

```
1. @using payrecord, sum of amount
    when socsecnum is 526687191
   or socsecnum is 585178564 ...
```

you would type:

```
2. @using payrecord, sum of amount
        when socsecnum is !m
```

What occurs in this second form, rather than search for a series of single dataitem values as in the first example, is that dataitem values are now taken one-by-one from the search list. The qualifying records are found, summarized, and added into the accruing statistics for each value in the list.

Please note that even in this simple example, two datasets were used. The item which qualified these social security numbers into a defined class was the state of home residence. The question that is being answered is a request for the sum of the total wages paid to the employees from Vermont, a request requiring information that must come from two different datasets.

## Using Search Lists to Accelerate Serial Reads

An important use of the @STORE form allows you to minimize the use of serial searches. Presume that *date* is a search item in a very large invoices dataset containing 10 years worth of invoices (say 100,000 invoices). Further presume that you need to report on only the invoices for a particular month (31 possible dates). Because *date* is a search item, there will be a master dataset for date, called perhaps DATE-ID. A master dataset, by its nature, contains only one entry for each search item value. Ten years of dates can total no more than 3,653 entries. Thus, it will be much quicker to gather a range of dates into a search class by serially searching a master dataset than it would be by serially reading the large detail dataset.

The two query questions used in such a search would be of this form:

```
@using date-id, store in !a date
 when date ib 19950101,19950131

@using invoices, sum of amount
        when date is !a
```

The first query question will necessarily be a serial search because of the "is between" relational operator. Although QueryCalc uses the fastest possible method to accomplish its serial reads, the second search will be faster yet. The second query question is a chained read, thus this search will access only those records which contain the desired dates in the search set and will generally be quite quick. The combination of these two techniques provides QueryCalc with a generic search technique that is comparable in speed to the very best third-party indexing techniques, with these added advantages: (1) no additional disk space is required, which is often quite extensive otherwise, (2) no interception of TurboIMAGE intrinsics is required, and perhaps most importantly, (3) the method is free and built into QueryCalc.

The further use of search lists is explained in Chapter 10, "Search Sets".

# USER EXERCISES

**Signing On**

1. Sign onto the practice account by typing **HELLO USER.AICS** at the colon prompt. Supply any necessary passwords. Passwords, if they are present, were put there by your system manager and he or she will know them if you do not.

2. Type **QC** to run QueryCalc. If this does not work, type **RUN QC.QCPROGS**. Press **RETURN** to bypass the instructions screen. You are now in QueryCalc proper.

3. Type **@OPENDB QCDEMO**. Type **FRONT** (in all caps) in response to the password question. IMAGE databases are the only place in the HP3000 where passwords are case sensitive. You now have the database open.

**Viewing your Open Databases**

4. Type **@SHOWDB** to show your open databases. You should see QCDEMO as the only open database. Press **RETURN** to return to the spreadsheet.

5. Type **@FORM** to see the basic form of QCDEMO. Master datasets (3x5 card-like sets) are always arranged on top of detail datasets (file drawer-like sets). Type **LABOR** to see the labor dataset. Type **JOB** to see the JOB dataset. As you'll see, there is no JOB dataset in QCDEMO. QueryCalc will instead show you all of the dataitem and dataset names which are close to JOB.

6. Press the **[f6]** function key to direct your output to your system printer. The system printer is connected to the HP3000 and is generally located in or near the computer room. Press **[f1]** to print the sets of QCDEMO on your system printer. Press **[f4]** to print the paths of QCDEMO. Press **[f3]** to print the entire form of QCDEMO. Press **[f8]** or " **//** " to return to the spreadsheet. Retrieve your output from the printer before we go any further.

**First Query Question**

7. Move the cursor to cell B2. You can do this either by using the function keys or by typing **/J B2**. Type **@sum of amount**. Because the dataset isn't specified and because AMOUNT appears in a number of datasets, QueryCalc will ask you which dataset you wish to use. Type either the name **INVOICES** or the number **4** for the

INVOICES dataset and press **RETURN**. QueryCalc will announce that the search must be serial, that it must read 9962 records and ask whether or not you wish to proceed. Respond by typing **Y** (yes) or **U** (use). What will appear in the cell will be the phrase, "<ok>". The dashes indicate that your input has passed syntax check, but the cell has not yet been recalculated (see Chapter 4, page 4-8).

### Immediate Execution of the First Query Question

8. Retype **@sum of amount!**, but this time with an "!" at the end. The "!" means execute the query question immediately. Answer the questions in the same manner as Step 7. This time, QueryCalc will not come back to you immediately but will instead display a message saying "Calculating query". Because this is a serial search of nearly 10,000 entries, the serial read may take a minute or two. When the answer does appear, it will be a cell full of asterisks, "**********". The asterisks mean is that the answer is too big for the current cell width. Type **/CWID 20**. This will widen the cell from 12 characters to 20. The answer, -3,616,241.26, is negative because both invoices and monies received are entered into the same dataset as positive and negative invoices. If the sum is negative, it means you took in more money than you spent.

9. Press **RETURN** to see the equation that has been entered into the cell. Because QueryCalc operates off of terminals, and because not as much information per second can be transmitted over a terminal's connection as in a personal computer, the cell equations aren't automatically displayed. You must ask for them. What you should see is "@Using qcdemo.invoices, sum of amount". Notice that QueryCalc automatically entered the "Using..." phrase for you and supplied the name of the database.

### Viewing the Individual Records You Found

10. Type **@show** to see the invoices you found with the query question of Step 8. The display will show you all of the dataitem names in the INVOICES dataset, the values for the first record found, and which items are search and sort items. Press **RETURN** to advance the listing. Press either " **//** ", **Control-Y**, or **[f8]** to return to the spreadsheet.

# USER EXERCISES

**A Chained Search**

11. Now for a chained search. Move the cursor to cell B3 by pressing **[f6]**. Type **@using invoices, sum of amount when category is 501!**. The answer that is returned is 23,984.97. Notice the difference in speed from the previous serial search. CATEGORY is a search item in INVOICES. The purpose of a database is to find the data you need as quickly and directly as possible. Using search items in your qualifying phrases (to the extent possible) greatly accelerates data retrieval. Which dataitems are search items are marked on the database @FORM listing we generated in Step 6

**Synthesizing Search Patterns**

12. Move the cursor to cell A4. Type **"501**. This enters a text value of "501" into the cell. Now move the cursor to cell B4. Type **@using invoices, sum of amount when category is [a4]!**. The "[...]" construct is a "window" back onto the spreadsheet which allows you to manufacture the search value. Any numeric or text equation may be placed within the brackets. Simply remember that a numeric equation uses no prefix (as in this example) and that a text equation always begins with a "$". The answer that you get should be the same as in Step 11. The only difference is that the search value came off of the spreadsheet and wasn't hard-coded into the query question itself..

13. Move the cursor to cell A5. Type **"459445550** to enter a social security number into the cell. Move the cursor to cell B5. Type **@using labor, sum of regular when socsecnum is [$a5]!**. The answer that should be returned is 1530 hours. This example is similar to Step 12, except that SOCSECNUM is a text dataitem. The pattern synthesized in the brackets must therefore be a text equation. Numbers are commonly stored in text dataitem fields. They may look like numbers, but they're not. They're text characters and they must be treated that way.

**Dependent Query Statistics**

14. Move the cursor to B6. Type **@ucs avg!**. Although the query question in Step 13 asked only for the sum of regular hours, the average, variance, standard deviation, maximum, and minimum were calculated as well. Retrieving this information by using the current statistics (@UCS) is free. Virtually no system overhead is incurred. Move the cursor to B7 and type **@ucs max!**.

# USER EXERCISES

**Rereading Previously Found Records**

15. Every record that qualified in Step 13 was marked and may be repetitively reread. Doing this is generally much more efficient than finding the records again. Move the cursor to cell B8 and type **@rereading, sum of overtime!**. The returned answer will be 202 hours of overtime. The set of records you are rereading is the same set of records that is shown if you type @SHOW. Notice that the @UCS query questions did not disturb the list of records found in Step 13. The list of records remains in effect until another standard (non-@REREADING) query question is executed.

**Creating and Using Search Lists**

16. Move the cursor to cell B9. Type **@using employees, store in m socsecnum when numdeductions ib 4,6!**. We have just created a search list named M filled with employee social security numbers for those people who claim 4 to 6 deductions in their payroll taxes. As you can see, there are 57 such people. To see the values in this list, type **@show !m**. @STORE lists will always be composed of unique entries. Duplicate values are eliminated. The people represented by these social security numbers represent a *class* whose membership is defined by their number of deductions. Press **RETURN** to return to the spreadsheet.

17. To use the list, move the cursor to cell B10. Type **@using labor, sum of regular+overtime*1.5 when socsecnum is !m!**. This question will sum all of the effective labor hours charged by this group of people (48,555.21 hours). Because SOCSECNUM is a search item, this question will search down 57 different chains and add all of the hours together. @STORE lists allow you to collect data and statistics on classes exactly as you would on individuals.

**More Examples & Using Your Databases**

18. Additional query question exercises exist in Chapter 7, Chapter 10, Chapter 11, Chapter 12, and Chapter 13.

To ask these same kinds of question using your own data, you need to learn your own databases. The first step is to repeat Steps 3-6 with names relevant to your databases.

## Concepts Introduced in Chapter 6

| | |
|---|---|
| STANDARD QUERY | The query question which summarizes data from a single dataset. |
| "USING..." PHRASE | The first phrase in a query question. The phrase which specifies which database and which dataset is to be used. |
| RELOPS | The relational operators which match a dataitem value to a data "pattern". |
| "REREADING, ..." | A dependent query question which rereads the list of records previously qualified by a standard query question. |
| CURRENT STATS | The various statistics (sum, average, max, min, etc.) summarized by a query question. Although only one statistic may be asked for in a query question, all are calculated. |
| SUBITEMS | Dataitem values held in a dataitem array. Dataitem arrays are denoted by 30X6, 10R2, etc. |
| SUBSTRINGS | Specified subsections of a text dataitem's string of characters. |

# InterChapter Reminder

All numeric dataitems are converted to high-resolution real numbers when they are brought into QueryCalc, regardless of how they are stored in your databases. The I, J, K, P and Z datatypes cannot however store "real" numbers (numbers with a decimal point). In order to maintain accuracy to the penny when these datatypes are used, a programming "trick" quite often employed, especially by COBOL application developers, is to "offset" certain data fields by multiplying the number by 100 (or more) before it is entered into the database.

Some report writers, such as HP's QUERY, use an edit mask to correct this offset. The numbers are used in the report writer in their offset fashion. It's only on printing that a decimal point is reinserted to make the number look correct. QueryCalc doesn't do that. A numeric value is presumed to mean what it says. Because of this, data can be extracted and mixed from a great diversity of sources. If you must extract data from an offsetted field, the solution in QueryCalc is straightforward. Divide the value by its offset during extraction. Examples of the syntax for such division is:

```
@using parts, sum of amount/100 when ....
@using payrecord, sum of \gross-billing\/1000 when ....
```

For more information about mathematical operations available in a query question, please refer to pages 6-3, 6-4, 6-5.

# 10

*Operating on sets of dataitems as members of a class*

# Search Sets

## Defining Sets of Records

It's possible to define records in QueryCalc as sets on the basis of common attributes. This is one of the more powerful features of QueryCalc, and yet quite simple to use. You will find search sets useful for a variety of reasons, but search sets are particularly valuable for (1) statistical and financial analyses, and (2) high-speed generic searches. Using search sets, employees in a database can be divided and subdivided into classes for purposes of insurance risk analyses. Or school children can be broken into economic and ethnic classes for the purposes of demographic analyses. And manufacturing processes can be subdivided into distinct classes for purposes of reliability and yield analyses.

If you think of a class—as in a class-action lawsuit—every member of the class shares some common property. In the case of a class-action lawsuit, every member of the class has been adversely impacted by some common action. It is the presence of a common attribute that defines a class. A second class drawn from the same community of members would be defined by another feature. Two such classes could potentially have (1) a few members in common (*intersecting sets*), (2) have no members in common (*mutually exclusive sets*), or (3) one class could be totally contained within the other (a *proper subset*). Membership in one class does not preclude membership in another. Any one individual could be potentially be a member of an infinite number of sets.

Imagine the set, which we'll call Set A, of all people who received at least one paycheck in 1993. In QueryCalc, the set is created through the use of a modified query question:

```
@using payrecord, store in !a emp-num
      when date ib 930101,931231
```

Set A is denoted as *!a* in QueryCalc (as always, capitalization is unimpor-

tant). The set that is created by this query question will be a list of employee-id numbers. Most of the people in this class will, of course, have received a number of paychecks during the year, but it is the nature of the @STORE query question to *eliminate all duplicate entries* once the search set has been built. The process of duplicate elimination is accomplished by sorting the retrieved search values and recording only those which are different. Thus, when completed, Set A will contain only one id-number for each employee who received at least one paycheck in 1993, and by consequence of the duplicate-elimination algorithm used, the set will be presented in a quasi-ascending order.

Twenty-six such sets may be created, labeled A to Z. The sets may be reused and redefined any number of times during the course of the execution of a file.

## Using the Search Set

The use of a defined set in QueryCalc is straightforward. If we wish to determine the average number of dependents that Set A has, the query question would be:

```
@using employees, avg of numdeductions
        when emp-num=!a
```

The query process proceeds in the following manner. The "pattern" to be matched (an employee number in this case) is neither "hard-coded" into the query question nor taken off of the spreadsheet. Rather, the first employee number is taken from Set A's list and a keyed search is performed on that particular value. As normally done, all of the statistics appropriate to the query question are calculated. Once the end of the search chain has been reached, the next search value is taken from the list and its chain is searched. The calculated statistics accrue until all of the items' records in Set A have been searched. In this manner, the average of the deductions in Set A will be calculated as a class.

## The Rules

There are a few rules which govern the use of sets in QueryCalc. They are:

1. The search sets contain lists of search item *values*, not record numbers. Thus a search set belongs to no single dataset or database and may be used in any variety of database environments, so long as the dataitem type and length are identical. The dataitem name is unimportant.

2. A set may be formed using any dataitem (keyed or not), but when the set is used for pattern matching, the specified dataitem must be a search item in the dataset to be searched. This rule exists for purposes of speed and computational efficiency.

3. Only one dataitem may be recorded in a search item list. This rule is in force because the sets will only be used against one search item at a time.

## A Quick Exercise

To demonstrate the power of search sets, please type the following:

```
:hello user.aics,qcdemo
:run qc.qcprogs.aics
```

You are now in QueryCalc. Move to cell B3, and type:

```
@opendb qcdemo/FRONT
@using employees, store in !b socsecnum
    when zip is 88047!
```

The @STORE query question creates set B, which will be a list of the people who live in a certain community. There will be five people who meet this criterion. To see the list, type:

```
@show !b
```

To use the list to determine the gross wages paid to this particular set of people in 1984, move to cell B4 and type:

```
@using payrecord, sum of gross when
    socsecnum is !b and date ib 840101,841231!
```

To calculate the average number of regular hours each of these people worked, using a different dataset, move to cell B5 and type:

```
@using labor, avg of regular when
    socsecnum=!b!
```

## Defining a Set Using Multiple Datasets, Databases, or Even Multiple HP3000's

The formation of a set of search item values may often require extracting information from several different datasets, perhaps located in different databases or on different HP3000s. It's quite possible to do this, but it's done one dataset at a time, sequentially. The method used is that a set of search values defined from one dataset are applied to a second dataset, where the set is redefined and further qualified. In the example shown here, three databases are used. The first database is an MPE flat file, the second a KSAM file, and the third an IMAGE database. Set A will not be completely defined until the third query question has been executed.

```
@using mpedb.employees, store in !a idnumber
      when startyear ib 1980,1984

@using ksamdb.insurance, store in !a idnumber
      when idnumber=!a and disability=Y

@using imagedb.accidents, store in !a emp-number
      when emp-number=!a and accident-type>3
      and accident-date ib 19900101,19901231
```

The Set A created by this series of query questions is a list of id-numbers for those people who started with the company in the years 1980 to 1984 (information found only in the first dataset), who have disability insurance (information found only in the second dataset) and who suffered an accident greater than type 3 in a particular range of dates (information found only in the third dataset).

Notice that in the third query question, the employee id-number has a different name, yet the search list can still be used. That's because the search list contains only *values*. If the search item is of the same data type and length, you can use the search list, without regard to the item's name.

When qualifying a search list from multiple datasets and databases, you will find it most efficient to execute your @STORE query questions in that order which will eliminate as many search entries as possible in the first query, and then in the second, and so on. The only exception to this general rule is when the dataitem to be @STOREd is not a search item in one of the datasets. That dataset will always have to be the first dataset to be searched.

## Nested Subsets

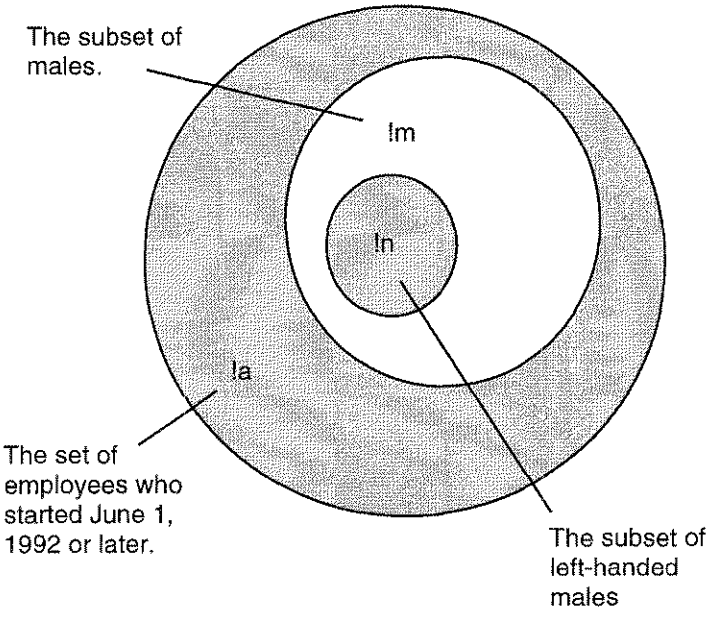
Nested subsets are created in an identical manner. It is quite easy to break a population of entries into a series of hierarchically-organized subsets for further statistical analyses. For example:

```
@using db1.employees, store in !a idnumber
when termdate<999999 and startdate>=920601

@using db1.employees, store in !m idnumber
      when sex is M and idnumber=!a

@using db1.employees, store in !n idnumber
when idnumber is !m and right-handed is N
```

These three @STORE query questions create three sets. Set M is a subset of the male employees, drawn from the list of people who began work on or after June 1, 1992 and continue to work now (Set A). The second @STORE creates set N, a subset of the set M, which is composed of only those males who are left-handed and meet the criteria of set A. By breaking an original population into these and similar smaller groups, you are offered the possiblity of performing high-speed, repeated statistical or financial analyses on selected groups. An example of such an analysis appears on the following page.



## An Example of a Nested Subset Analysis

The following analysis is an example drawn from real data and a program in actual use. The analysis represents a general estimate of the surgical risk associated with coronary arterial bypass-graft surgeries (CABGRISK) when broken into several categories of risk factors and intra- and post-operative complications.

The analysis begins on Page A with the creation of nine sets, labeled A through I. Because the data must be extracted from a variety of datasets, a series of query questions must be used. Set A isolates all of the patient numbers who had operations in the specified range of dates (cell Ad7). Set B is formed by using Set A's list of patient id-keys to determine those patients who had the proper form of operation while eliminating any patients which had complicating factors such as vascular disease (vsd) or additional valve surgery (cell Ad8). As occasionally happens, the length of the query question necessary to form Set B exceeds the 187-character limit of the cell. Thus, a second set, C, is formed as a subset of Set B, further refining the list of qualifying patient numbers (cell Ad9). This is not a particularly efficient procedure, but it is sometimes necessary.

The @STORE equations for Page A are shown on the opposite page. The actual number of items each equation generates are shown below.

```
        Risk factors in females having CABG

                    From:1981
                      To:1983

Date select List A:      1,407.00
     CABG-I List B:        846.00
    CABG-II List C:        837.00
      Female List D:       164.00
   Age > 60 List E:        105.00
     Obesity List F:        43.00
   Diabetes List G:         12.00
Hypertension List H:        10.00
      Smoker List I:         0.00
```

*The output that results from the construction of the nested subsets on Page A of the CABGRISK report.*

```
A |_____a_____|_____b_____|_____c_____|_____d_____|_____e_____|_____f_____
 1|
 2|                  Risk factors in females having CABG
 3|
 4|                                  From:     (Ad4)
 5|                                    To:     (Ad5)
 6|
 7|                Date select List A:      (Ad7)
 8|                     CABG-I List B:      (Ad8)
 9|                    CABG-II List C:      (Ad9)
10|                     Female List D:     (Ad10)
11|                   Age > 60 List E:     (Ad11)
12|                    Obesity List F:     (Ad12)
13|                   Diabetes List G:     (Ad13)
14|                Hypertension List H:    (Ad14)
15|                     Smoker List I:     (Ad15)
16|
17|


Equations for Page A
____________________


   (Ad4): 1981
   (Ad5): 1983
   (Ad7): @Using cardio.name, store in !a id-key when surg-date
          ib [d4*10000],[d5*10000]
   (Ad8): @Using cardio.op, store in !b id-key when id-key=!a and
          \coro-art-surg\=Y and \resection-va\<>Y and \repair-vsd\<>Y and
          \valve-surg\<>Y and \cong-surg\<>Y and \thoracic-surg\<>Y
   (Ad9): @Using cardio.op, store in !c id-key when id-key=!b and
          \peri-vasc-surg\<>Y and \graft-rep\<>Y
  (Ad10): @Using cardio.name, store in !d id-key when id-key=!c
          and sex=F
  (Ad11): @Using cardio.name, store in !e id-key when id-key=!d
          and age-operated-on>=6000
  (Ad12): @Using cardio.preop, store in !f id-key when id-key=!e
          and obesity=Y
  (Ad13): @Using cardio.preop, store in !g id-key when id-key=!f
          and diabetes=Y
  (Ad14): @Using cardio.preop, store in !h id-key when id-key=!g
          and hypertension=Y
  (Ad15): @Using cardio.preop, store in !i id-key when id-key=!h
          and smoker=Y
```

*The equations of Page A necessary to create the nine hierarchically nested subsets used in the CABGRISK report.*

# Surgical Risk Analysis by Class (Continued)

The use of the defined sets in the actual cardiovascular surgical risk analysis appears on Page B of the report. Five of the eight data columns of the report are shown below. A portion of the equations which comprise Page B are shown on the opposite page.

The first column (CABG ALL) refers to the patients of Set C. The next column (Females) references those in Set D. The following columns use the data in Sets E, F, and G respectively. The columns using Sets H and I, as well as some additional summary statistics, occur off-page.

| | CABG ALL | Females | Females over 60 | Females >60, obese | Females >60, obese diabetic |
|---|---|---|---|---|---|
| | | 164 | 105 | 43 | 12 |
| | | 19.6% | 12.5% | 5.1% | 1.4% |
| | | 5 | 4 | 1 | 0 |
| | | 3.0% | 3.8% | 2.3% | 0.0% |
| | | 0.6% | 0.5% | 0.1% | 0.0% |
| | | 7 | 5 | 0 | 0 |
| | | 4.3% | 4.8% | 0.0% | 0.0% |
| | | 0.8% | 0.6% | 0.0% | 0.0% |
| | | 1 | 1 | 0 | 0 |
| | | 0.6% | 1.0% | 0.0% | 0.0% |
| | 1.3% | 0.1% | 0.1% | 0.0% | 0.0% |
| Operative mortality | 12 | 3 | 3 | 1 | 0 |
| | 1.4% | 1.8% | 2.9% | 2.3% | 0.0% |
| | 1.4% | 0.4% | 0.4% | 0.1% | 0.0% |
| Myocardial infarction File IV | 0 | 0 | 0 | 0 | 0 |
| | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% |
| | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% |

*The data as extracted by class of patient. 837 of the 1407 surgeries performed during the two year period were coronary bypass grafts, without additional complicating factors. 164 of these operations were on females.*

Equations for Page B

```
 (Bb7): AD9
 (Bb8): 1
(Bb10): @Using cardio.post, num when id-key=!c and \kpl-d(3)\=Y
(Bb11): B10/B7
(Bb12): B10/{B7}
(Bb14): @Using cardio.post, num when id-key=!c and \kpl-d(4)\=Y
(Bb15): B14/B7
(Bb16): B14/{B7}
(Bb18): @Using cardio.post, num when id-key=!c and \kpl-d(12)\=Y
(Bb19): B18/B7
(Bb20): B18/{B7}
(Bb22): @Using cardio.post, num when id-key=!c and alive-or-dead=D
(Bb23): B22/B7
(Bb24): B22/{B7}
(Bb26): @Using cardio.postpost, num when id-key=!c and \kpl-d(4)\=Y
(Bb27): B26/B7
(Bb28): B26/{B7}
 (Bc7): AD10
 (Bc8): C7/B7
(Bc10): @Using cardio.post, num when id-key=!d and \kpl-d(3)\=Y
(Bc11): C10/C7
(Bc12): C10/{B7}
(Bc14): @Using cardio.post, num when id-key=!d and \kpl-d(4)\=Y
(Bc15): C14/C7
(Bc16): C14/{B7}
(Bc18): @Using cardio.post, num when id-key=!d and \kpl-d(12)\=Y
(Bc19): C18/C7
(Bc20): C18/{B7}
(Bc22): @Using cardio.post, num when id-key=!d and alive-or-dead=D
(Bc23): C22/C7
(Bc24): C22/{B7}
(Bc26): @Using cardio.postpost, num when id-key=!d and \kpl-d(4)\=Y
(Bc27): C26/C7
(Bc28): C26/{B7}
```

*The equations for two of the columns on the opposite page. The first column (spreadsheet column B) analyzes the risk of complications for the entire set of patients, Set C. The second column (spreadsheet column C) analyzes exactly the same risks for female patients, Set D. Because the columns are basically identical, column C is a /REPlicated duplicate of column B. Column C was modified using the search-and-replace command (/S&R), replacing "!c" with "!d" everywhere in the column. Each of the remaining columns was built in same manner. The time necessary, therefore, to construct the entire page was only 10-15 minutes.*

## High-Speed Generic Searches in IMAGE

An unanticipated benefit of @STORE sets in QueryCalc is that they provide a mechanism for high-speed generic searches of IMAGE datasets under specific conditions. Other database structures, such as ISAM, KSAM, and SQL†, often have advantages over IMAGE when a query question's relational operator (*relop*) is "greater than" or "is between". These relops do not imply a search for a single value, but rather a range of values. High-speed range searches, unfortunately, are normally impossible in IMAGE.

The mechanism by which a generic search is implemented in ISAM, KSAM and SQL is called a *b-tree*, short for *binary-tree*. IMAGE, in contrast, uses *hashed* keys. In a hashing algorithm, the search item value, even though it may be text, is considered to be a number. All data stored in a database is stored as 1's and 0's. Whether those bits are to represent text or numeric information depends only on the person defining the database, not the storage mechanism itself. It's therefore quite possible to take any value and view it as a number. In a hashing key scheme, this number will represent an address in a look-up table. The value held at the table address will be the record number of the first record containing the search item value. Being only a two-step process, hashing is the most efficient search algorithm known, but it suffers from a severe drawback in that the search item value must be known in its entirety before it can be used.

B-trees are different. A b-tree operates by asking a series of yes-no questions (hence the term, *binary*). The questions are of the form: is the search value is greater or less than a particular value? By navigating a b-tree, records may either be found (1) which possess exactly the search value specified, or (2) lie within a specified range of values. This latter form of search is called a *generic* search. The capacity for generic searches underlies the appeal of b-tree search indexes. However, this enhanced search capability is not without its cost. B-tree indexes are usually quite a bit slower than hashed searches.

Generic searches may be simulated in IMAGE databases using @STORE sets. But what you may find to be surprising is that generic searches in IMAGE using QueryCalc's @STORE sets are often significantly faster than KSAM b-tree searches. The trick is quite simple. Any search item in an IMAGE detail dataset must have a master dataset attached to it. A serial-

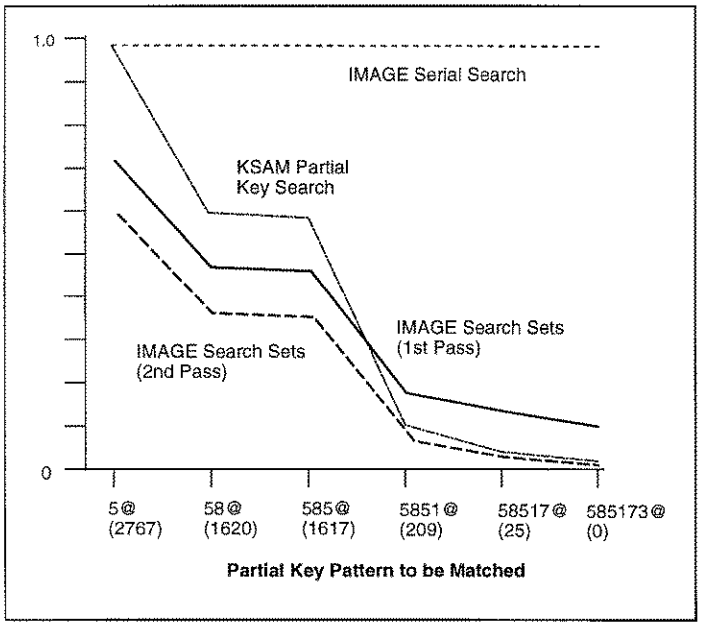
†ISAM stands for *indexed sequential access method.* ISAM is an IBM product. KSAM is an HP product and is very similar to ISAM. KSAM stands for *keyed sequential access method.* SQL was originally an IBM sublanguage, part of the DB2 database, but the use of the term has now become generic. SQL stands for *structured query language.*

search of the master dataset is almost always much less expensive in time and CPU resources than a serial search of the detail dataset. All of the search values which qualify in a partial-key serial search of the master dataset are @STOREd in a QueryCalc search set. That set is then applied in a search of the corresponding detail dataset. The query questions will be of this nature:

```
@using employee-id, store in !m socsecnum
      when socsecnum=58@

@using payrecord, sum of gross
      when socsecnum is !m
```

In this example, EMPLOYEE-ID is an IMAGE master dataset. PAYRECORD is a detail dataset. The acceleration advantage that search sets offer accrues because there are often a large number of entries attached to each search chain in an IMAGE detail dataset. Although serially searching the master dataset and building the search Set M requires some resources, the technique is actually faster than either serially searching the detail dataset or using a b-tree search against an identical KSAM dataset. Indeed, as shown by the figure below, the speed advantage can often be quite significant.




*The ratio of measured search times for various generic search techniques. The partial key pattern to be matched is shown along the x-axis. The "@" represents a wild card symbol. The number of records that qualified in each search is shown in parentheses.*

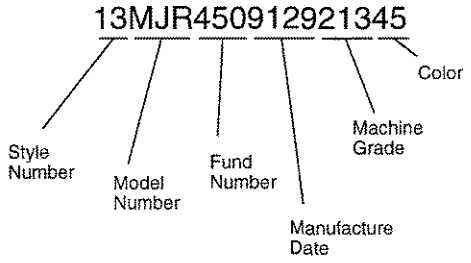
*A serial search of a detail dataset will generally be the slowest method. Its time-of-search will however be independent of the number of records that are qualified. When the number of records qualified becomes large enough (10-30%) of the dataset, serial searches often become faster than b-tree searches.*

*Two IMAGE search set curves are shown. The "1st Pass" curve includes both query questions shown in the text above (a serial search of the master dataset followed by a keyed search of the detail dataset). But once a search set has been built, it may be reused again and again without further overhead. The reuse times are shown in "2nd Pass".*

# High-Speed Partial Key Searches of Concatenated Keys

The capacity to search for only a portion of a key item's value is called a *partial key search*. The b-trees keys of KSAM and SQL allow you to do this quite easily, but only if you are searching for the first few characters of the key. Quite often however the need arises to seek out records on the basis of values lying somewhere in the middle of the key value. In this case, unfortunately, a b-tree will be of no use. However, IMAGE inquiries using QueryCalc's @STORE sets will continue to work as efficiently and offer distinct advantages over b-tree generic searches.

Intermediate partial key searches become an important topic if your database uses concatenated keys. A *concatenated key* is one where a number of separate characteristics are put together in a single key. An example of a concatenated key is shown here:




Concatenated keys can often be quite useful. They allow you to locate specific records quite quickly. But just often, they represent a significant barrier to efficient report writing. If you must search for an item in the middle of the key, such as *fund number* in this example, generally the only recourse in KSAM, IMAGE, and SQL is to use a serial search.

The search-accelerating method discussed on the previous two pages works just as well for intermediate partial key searches, but only when IMAGE databases are used. The technique will be exactly the same: (1) first perform a serial search of the master dataset, storing the qualified key values, and then (2) apply the formed set to the detail dataset, as shown:

```
@using part-master, store in !a part-key
      when part-key(6,8) = 450

@using part-detail, sum of qty-on-hand
when part-key is !a and stock-region is AZ
```

This technique will not work with MPE flat files, KSAM, or SQL datasets

because they do not have directly accessible master datasets. The presence of such accessible key files (master datasets) in IMAGE is the reason the method works.

If the key item is text, there are a number of ways to search for pieces of the key item value. The search may proceed either by absolute position or through the use of "wild cards". Please see Chap. 6, "Query Questions" for a complete list of partial search capabilities.

## When to Use the Method

The generic-search technique of using search sets is often quite advantageous, but like most things, it is no universal panacea. There are specific instances when it should and shouldn't be used. The rules on when to use the technique are these:

1. *Use the method if you are likely to qualify less than 20-30% of the records in the dataset.* Qualifying more records than this percentage and a single serial search of the dataset becomes faster than multiple chained searches. In these circumstances, write the query question in the normal fashion, as shown here:

```
    @using part-detail, sum of qty-on-hand
when part-key(6,8) is 450 and stock-region is AZ
```

2. *Don't use the method if there is a more direct way to retrieve the records.* If you know some special information about your records, such as that the Arizona stocking region represents only a very small proportion of records in your database, and STOCK-REGION is a search item, then again write the query question as shown above. Searching down just one short search chain will be much faster than searching through multiple chains.

3. *Use the method when there are many records in the detail dataset for each master dataset key-item value.* The technique works best when a one-to-many ratio exists. The greater the number of records per chain in the detail set, the greater the speed advantage of the technique. But if only one record exists per chain, no advantage exists. Indeed, then there will be a small "cost" to the technique.

## Boolean Set Algebraic Operations

QueryCalc allows the extensive manipulation of search sets using standard Boolean algrebra set operations. The only requirement is that the sets must be of matching data types (e.g., I2, R4, Z8, etc.). Normally, you do not need to be aware of the data type of the search values you are manipulating. It becomes a matter of concern here simply because there is no easy way to combine text and numbers into a single set.

The syntax for the set manipulation algebra is:

```
@Using sets, !y=!a+!b
```

Set Y will be defined as the *union* of Sets A and B. Four set algebraic operators are possible and are represented in the facing figure, along with the resulting set. They are: *union*, *subtraction*, *intersection*, and *exclusion*. In all set algebra operations, duplicate entries are eliminated. No value will appear more than once.

If set operations are to be performed on multiple sets, the operations must be placed in a series of cells:

```
@using sets, !y=!a+!b
@using sets, !y=!y+!c
@using sets, !y=!y-!d
```

This sequence of set operations is equivalent to

```
!y=(!a+!b+!c)-!d
```

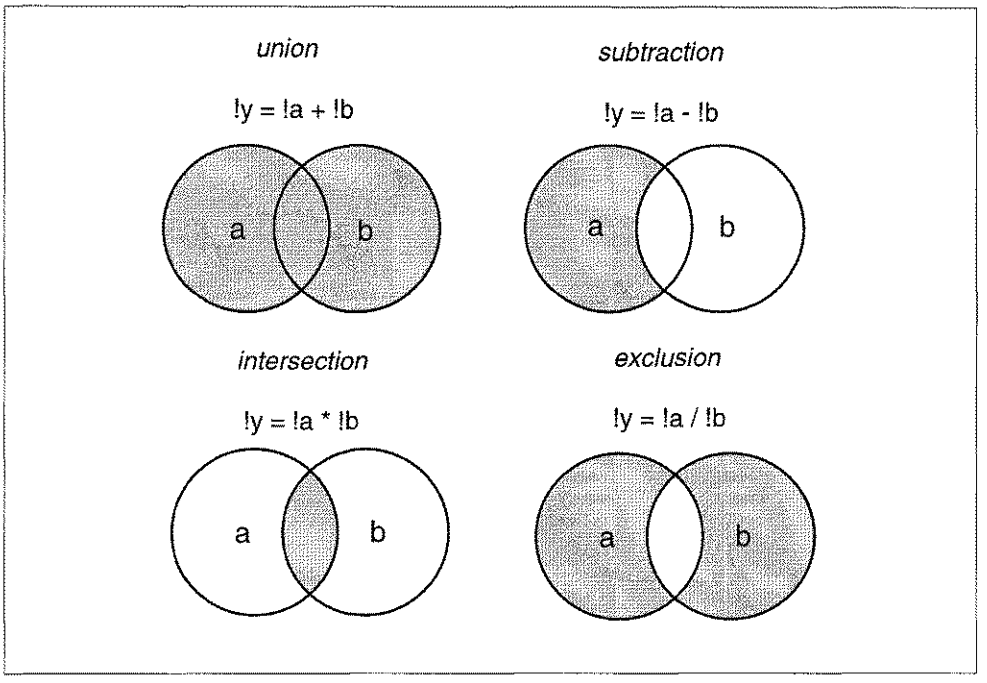
Using a series of cells, any Boolean algebraic manipulation is possible.

## Order Dependence

All of the algebraic operations [other than set subtraction (-)] are *commutative*, that is, the order is unimportant. For the commutative operations, each of the equations shown below are equivalent:

```
@using sets, !y=!a+!b    @using sets, !y=!b+!a
@using sets, !y=!a/!b    @using sets, !y=!b/!a
@using sets, !y=!a*!b    @using sets, !y=!b*!a
```

*Set subtraction is the exception.* Order is very important, as you can visualize by looking at the set diagram.




*The Boolean algebraic set operations of QueryCalc.*

## Manually Defining a Search Set

Search sets may also be manually defined rather than be assembled from data in the database. Examples of the syntax are:

```
@using sets, define !a as
  holland,"smith, john",
yankees,valley-plum;type=U20

@using sets, define !b as
  576,711,2303;type=I2
```

The rules for set assembly are simple. If the items to be defined to comprise the list are text, they must be surrounded by quotes if the item contains a comma or a semicolon; otherwise, they may simply be listed, as shown. Text will be upshifted if the dataitem type is specified to be "U" and left as is if the type is "X". If the dataitem type is specified to be a numeric data type, all of the items in the list must be numeric values. Duplicate entries will be eliminated and the list will be sorted, as is normal for all sets.

## Using External Files to Define Search Sets

Search sets may also be defined using external flat ASCII files. The syntax is:

```
@using sets, define !c from
filename.groupname.acctname;type=R4
```

Please note that the difference between these this set-definitional form and the previous one is the use of the "as" and "from" prepositions.

The italicized names specify the group and account of the file and are optional. The file must be a flat ASCII file containing only printable characters. If you can text in and read the file using EDIT/3000, then it meets these specifications. You may create this file using EDIT/3000, *but the file must be kept unnumbered.* The flat ASCII file containing the search item values must look like this:

```
abercrombie
wallace
mustafa
chin
hruska
miller
lucky
syvertson
   •
   •
```

Only one search item value should appear in each row. The item should be left justified. If leading blanks must appear as part of the key item value, type them in that way. They will be retained. All trailing blanks will be deblanked, however. The defined list will be sorted on entry and duplicate entries will be eliminated.

# Concepts Introduced in Chapter 10

| | |
|---|---|
| SEARCH SET | A list of search item values that belongs to no dataset or database. |
| NESTED SUBSET | A set of items which is derived from a larger set. All of the items in the subset appear in the larger set. |
| GENERIC SEARCH | A search which proceeds by looking for all of the records which lie between specified limits. The exact values of the search items are not known in advance, as they are in a specific search. |
| PARTIAL-KEY SEARCH | A search which proceeds by looking for a part of search item value. |
| SET ALGEBRA | The algebra of logical sets. The common operations are set union, intersection, subtraction. Union is equivalent to logical ORs. Intersection is equivalent to logical ANDs. |

# Technical Appendix: Comparing SQL to QueryCalc

## Introduction to SQL

SQL (*structured query language*) is a database/query language that was defined in the early 1970's by IBM. SQL was built around the idea of *sets*, and thus it is appropriate to compare SQL to QueryCalc at the end of this chapter. QueryCalc, in many ways, is surprisingly similar to SQL, especially in its query questions. But there are also significant differences. These differences are not only the result of different design purposes but also very different views on how the data should be presented and how much power should be given to the user.

SQL, in contrast to QueryCalc, contains its own database structure. QueryCalc accesses pre-existing databases, generally IMAGE and KSAM. SQL, on the other hand, is not a report writer *per se*. Although SQL has a well-designed interface which may be used to interactively test query questions or be used as an *ad hoc*, "quick-and-dirty" report generator, any complicated output that would normally be associated with a production-level report must be written in another language, such as PASCAL.

One of the design criteria of SQL was to hide as much of the database structure as possible from the user. Ideally, the user would not need to learn or know anything of keys, datatypes, or the like. In actual practice, as you might expect, that becomes impossible. The skilled SQL user must eventually become at least as knowledgeable about his database structures as a user of IMAGE.

SQL datasets are constructed as tables, as shown at the right. The dataitem names are listed at the top of the columns. The presentation order is rotated 90° from the way data is presented in QueryCalc. Each record entry is a row in the table. Two attributes differentiate SQL datasets from IMAGE: (1) No row (record entry) may be repeated in a table (this is a major difference)And (2), key item values in SQL may be the result of the concatenation of several adjacent dataitems, as shown in the SPJ table, a suppliers-parts-jobs table. The other tables in the example are the suppliers, parts, and jobs tables. Key items are marked where a value is encircled by a light box.

S

| S# | SNAME | STATUS | CITY |
|---|---|---|---|
| S1 | Smith | 20 | London |
| S2 | Jones | 10 | Paris |
| S3 | Blake | 30 | Paris |
| S4 | Clark | 20 | London |
| S5 | Adams | 30 | Athens |

P

| P# | PNAME | COLOR | WEIGHT | CITY |
|---|---|---|---|---|
| P1 | Nut | Red | 12 | London |
| P2 | Bolt | Green | 17 | Paris |
| P3 | Screw | Blue | 17 | Rome |
| P4 | Screw | Red | 14 | London |
| P5 | Cam | Blue | 12 | Paris |

J

| J# | JNAME | CITY |
|---|---|---|
| J1 | Sorter | Paris |
| J2 | Punch | Rome |
| J3 | Reader | Athens |
| J4 | Console | Athens |
| J5 | Collator | London |
| J6 | Terminal | Oslo |
| J7 | Tape | London |

SPJ

| S# | P# | J# | QTY |
|---|---|---|---|
| S1 | P1 | J1 | 200 |
| S1 | P1 | J4 | 700 |
| S2 | P3 | J1 | 400 |
| S2 | P3 | J2 | 200 |
| S2 | P3 | J3 | 200 |
| S2 | P3 | J4 | 500 |
| S2 | P3 | J5 | 600 |
| S2 | P3 | J7 | 400 |
| S2 | P5 | J2 | 100 |
| S3 | P3 | J1 | 200 |
| S3 | P4 | J2 | 300 |
| S4 | P6 | J3 | 500 |
| S4 | P6 | J7 | 300 |
| S5 | P2 | J2 | 200 |
| S5 | P2 | J4 | 100 |
| S5 | P5 | J5 | 500 |
| S5 | P5 | J7 | 100 |
| S5 | P6 | J2 | 200 |
| S5 | P6 | J3 | 100 |
| S5 | P6 | J4 | 900 |
| S5 | P6 | J7 | 100 |

*A simple SQL-like database for suppliers, parts, and projects.*

## Query Questions in SQL & QueryCalc

Query questions in the two languages are quite similar. For example, to find the records which meet a specified set of selection criteria, the syntaxes are:

```
@using parts, find when status=30 (QC)

Select * from parts where status=30; (SQL)
```

The "*" in SQL means retrieve and display all columns (dataitems) in the table. SQL organizes the records it finds into a new table, called a *result table*. QueryCalc marks its qualified records in basically the same way. A @FIND is followed by a @SHOW command. In SQL, you may specify that the retrieved record table only contain certain dataitems:

```
Select pname,color from parts
   where status=30;   (SQL)
```

In QueryCalc, the command following an @FIND would be:

```
@show pname,color (QC)
```

In both query languages, statistical calculations may be performed on the data in a dataset as a search proceeds. Examples of the two syntaxes are:

```
@using parts, sum of weight*454
      when status=30 (QC)

Select sum(weight*454) from parts
      where status=30; (SQL)
```

## SQL is a Relational Database

SQL was built around the "Relational Model" of Dr. Edgar Codd. The word "relational" in a relational database has nothing to do with the common use of the English word, where you might suspect that items in different datasets can somehow be easily related to one another. Rather, the word is derived from the mathematical definition of a *relation* in set theory. A relation $R$ is a $n$-tuple defined as

$$R = < d_1, d_2, d_3, \ldots . , d_n >$$

where $d_1 \in D_1$, $d_2 \in D_2$, ...., $d_n \in D_n$, where $D_1$, $D_2$, ..., $D_n$ are the *domains* for each of the $d \in D$. A domain specifies the allowed set of values

which each element in the *n*-tuple may assume. A relation *R* is simply one table in an SQL database. The various $d_i$ are the column entries (color pname, city, etc.). Entry values in an SQL table are "drawn" either from the set of allowed values (the domains) or classified as either one of two *missing* categories, important or not important. In contrast, IMAGE and KSAM do not intrinsically impose limits on what data may be entered into a dataitem's field. Nor do KSAM or IMAGE allow marks for missing data.

## Structured Query Questions

The word *structured* in Structured Query Language comes from the capacity in SQL to nest a sequence of query questions. Each query subpart establishes a qualifying set of records which may then be used to qualify another set of records in another table (dataset). In SQL, if you wished to list all of the suppliers who manufacture red parts, three tables (suppliers, parts, supp-parts) might be used. If so, the syntax would be:

```
Select suppliername
from suppliers
where suppliernum in
      ( Select suppliernum
        from supp-part
        where partnum in
            ( Select partnum
              from parts
              where color='Red' ));
```

The syntax in QueryCalc is different, but the result is precisely the same:

```
@Using parts, store in !a partnum
           when color=Red
@Using supp-parts, store in !b suppliernum
           when partnum=!a
@Using suppliers, find
           when suppliernum=!b
@show suppliername
```

In SQL, the nested query sequence is read from bottom-to-top. In QueryCalc, the order is not only reversed, but the individual query parts must be placed in separate cells. Which is better? Beyond personal preferences, the differences between the two approaches speak to the core of the differences underlying the design philosophy of the two languages.

## The Concept of Joins

SQL allows you to combine columns from various tables into a new result table. This process is called a *join*. In most commercial SQL implementations, only a few types of joins are possible. In HP's ALLBASE/SQL, only *natural joins* and *outer joins* are supported. An example of a natural join is the SQL nested query on the previous page. A natural join only returns records for which there are exact matches in the joined tables. Outer joins allow the union of records where matches don't exist. Codd (1990, *The Relational Model for Database Management: Version 2*) has defined 43 different forms of joins, giving them names such as *recursive join*, *semi-theta-join*, and *symmetric equi-join*, while suggesting that the list is by no means complete. QueryCalc is completely differently designed at this level of organization than is SQL. QueryCalc uses no joins at all. Rather, it was designed (1) to read from only one dataset at a time, and then (2) relate that data to any other dataset in any way that the user might wish.

These philosophical differences do make substantial practical differences. Dr. Paula Hawthorne, director of applications and technologies at Hewlett-Packard, recently said, "I was at one point in charge of the quality assurance group... SQL as a language is very confusing to the users. At least half of our bug calls were because people didn't understand an SQL query and what the right answer to that query was. In fact, often reading the literature, we didn't understand it either. Part of the problem with quality in relational database system is a problem with SQL" (*Interexpress*, June 1991).

SQL was designed with the best possible intentions: to make the data presentation as simple as possible. No presentation can be simpler than a row-column table. But we have always felt that motives directed towards the isolation of the user from the data storage were a fundamental mistake. Such an approach not only confuses even the most astute user, because he does not know precisely what the program is doing for him, it places an immense burden on the query language designer. The designer must generate large-scale data operations capable of every possible manipulation the user might desire. The far preferable tack would seem to be to generate a few, very powerful primitives that can connected together indefinitely.

# 13

*Producing long detailed lists of items, sorted and summarized*

# Detail List Reports

## The Indefinite Nature of the List Report

If you were the manager of a library, you might want to create a list report like this: "Print a list all of the students who have books overdue. List the books for each of the students and calculate the value of the penalty for that book. Summarize the penalties for each student and give me a total for all of the students. The output should look something like this:"

```
Able, John                    505-52-6841
    The Grapes of Wrath            63          14.38
    The Witches of Eastwick        38           6.14
                                                -------
                                                20.45
Anderson, Beverley            568-45-7191
    Fish King of Alaska           128          56.45
                                                -------
                                                56.45
Asaud, Ron M.                 205-56-3682
    A Guide to DB2                 23           6.14
    Thoughts & Discourses           7           1.12
                                                -------
                                                 7.26
```

Such a report is composed of two *groups*: the outer group is the alphabetized list of students, the inner group is the list of books that each student has out, sorted by the length of time the book is out. We call this kind of report an *indefinite (detail) list report.* The indefinite nature of the detail report is due to the fact that initially you don't have any idea of how long a list you're going to generate. That indefinite nature is in direct contrast to the fixed row/column format of a financial spreadsheet report where summaries will occur in specific, preassigned locations.

## Construction of a Detail List Report

A detail list report format will always be reminiscent of the outline format you used in your high-school History class:

```
I. major subject heading
    A. minor subject heading
        1. sub-subheading
            a. example
            b. example
            c. example
        2. sub-subheading
            a. example
            b. example
    B. minor subject heading
        1. sub-subheading
            a. example
            b. example
II. major subject heading
    A. minor subject heading
                .
                .
                .
```


Each indention represents a new group. Four groups occur in the example above. In the abstract, the information found in each group is identical, although the values will obviously differ. *The information presented within each indented group must be somehow directly related to information appearing in the group level immediately above it.*

The idea of a group is key to understanding a detail list report. The group is where items are listed, sorted and summarized. In QueryCalc, 50 different summarizations (averages, maximums, sums, counts, etc.) can be simultaneously maintained for a single detail list report. Items which appear in lower levels can be summarized in all higher group levels, allowing you to create grandtotals (or grandaverages, or whatever) at every successively higher level.

QueryCalc repeats the basic pattern of the History-class outline in its report layout structure, even to the point of automatically indenting the report in its default layout setting.

## What is a Group?

A group occurs every time a new, more detailed list of items is desired. To illustrate, let us presume that we would like a detailed list of all of the generals who participated in all of the wars that the United States has fought since the Revolutionary War, listed battle-by-battle.

The outermost group (Group 1) will be a list of the wars. The second group (Group 2) will be a list of each of that war's battles, and the third group (Group 3) will be a list of the generals for each battle. The format of a multi-group report will always be somewhat similar to the outline form shown on the opposite page.

Let us presume that the information we will need for the report appears in three different datasets, WARS, BATTLES and GENERALS. If you are familiar with the concept of JOINs and MULTIFINDs, we want you to forget everything that you know. The way QueryCalc is put together is much simpler and much more powerful. In QueryCalc, a query question extracts information out of only one dataset. Once that extracted information appears in a cell, you may use it as part of the qualifying information for your next query question. In QueryCalc, the process of linking or joining datasets together is very straightforward and intuitive.

## How a Report is Built

The first command of any list report is **/REPORT**. This command converts a standard spreadsheet page into a detail list report page. The first query question on the /REPORT page might be:

```
@using wars, find when year > 1775
        and country is US
```

What QueryCalc will do when this question is executed is find every record in the WARS dataset that meets the specified criteria. The qualifying records will be remembered, and once the initial search pass has been completed, each record will be brought back up, one at a time. All of the information that you wish to get from that one record is then read onto the spreadsheet by a series of @REREADING query questions:

```
@rereading, val of war-code
@rereading, val of war-name
@rereading, val of start-date
@rereading, val of end-date
```

## The Report's Query Questions

The second group, which is to be a list of all of the battles for a specific war, will be found in the same manner. And the same is true of the third group, the generals who commanded each battle in each war. The sequence of query questions will always look something like this:

```
A1: @using wars, find when year > 1775
       and country is US
A2: @rereading, val of war-code
A3: @rereading, val of war-name
A4: @rereading, val of start-date
A5: @rereading, val of end-date

A6: @using battles, find when war-code is [a2]
A7: @rereading, val of battle-code
A8: @rereading, val of battle-name

A9: @using generals, find when war-code is [a2]
       and battle-code is [a7]
A10: @rereading, val of lname
A11: @rereading, val of fname
A12: @rereading, val of command-strength
```

Each new group is defined by an @FIND. In this example, Group 1 begins in Cell A1, Group 2 begins in Cell A6, and Group 3 begins in Cell A9.

The first line of a /REPORT page must always be an @FIND, and it is normally stand-alone (that is, it does not depend on any other cell for its information). But the subsequent @FINDs on the /REPORT page should always be written such that they back-reference information in the earlier groups. In this example, Group 2 (Cell A6) finds the appropriate battles for the war that is currently displayed in Group 1. And Group 3 (Cell A9) finds the appropriate generals for the war and battle currently displayed in Groups 1 and 2.

## Reading the @FIND Lists

QueryCalc maintains three lists in a 3-group report. The first group's list, which is a list of wars, is read very slowly. The second group, the battles, is read more quickly. And the last group, the generals, is read very quickly indeed. When the list of generals for the first battle is exhausted, the next battle for the current war is read from the Group 2 list and a new list of generals is found. When this new list is exhausted, the next battle is read. The process repeats until the list of all of the battles for the war is done, and then the first group moves forward one war. The process simply repeats until all of the values in all of the lists have been read.

## The Resulting Report

The resulting report may be formatted in an almost infinite number of ways. One possibility is shown here:

```
U.S. Civil War, 1861-1865
     July 1861, First Battle of Bull Run
          Johnston, CSA                13,000
          Beauregard, CSA              28,100
          McDowell, USA                31,800
     May 1862, Yorktown
          Lee, CSA                     63,600
          Johnston, CSA                47,200
          McClellan, USA               83,450
     June 1862, Seven Days Battle
          Johnston, CSA                41,400
          Lee, CSA                     57,300
          McClellan, USA               75,700
     Sept 1862, Antietam
          Lee, CSA                    103,650
```

## The Sequence of Commands

Four commands are used to build a list report. They are used in this order:

**/REPORT**
**!!**
**/LAYOUT**
**/PRINT**

Three new commands are unique to the detail list report writer:

**/REPORT** converts a standard spreadsheet page into a detail list report page

**/LAYOUT M** invokes QueryCalc's editor, allowing you to modify the basic formatting structure for page M. If the page is not specified, the current page is assumed.

**/NEWLAYOUT B** invokes QueryCalc's editor, but erases the previous layout and builds a new default layout for Page B. If the page is not specified, the current page is assumed.

## The Report Page

A **/REPORT** page consists of four prespecified columns. You cannot increase or decrease that number, nor can you change the widths of the columns. Each column is used for a specific bit of information.

```
   DEMOREPT              /report                Col Dec2 , RJ
B |   equation (a)     |   name(b)   | sort(c)   |   width(d)
--+-------------------------------------------------------------
 1| ------------------find when keyname<>0;limit=100----------
 2| ZLEOSS              keyname          1a
 3| SANTOS LEOS         name
 4| 1313 EAST MADRID    address
 5| TAOS
 6| NM
 7| 88003
 8| TAOS,NM 88003       city
 9| 101                 acctcode
10| 20,652.82           amount
12|
13|
14|
15|
```

*The layout of a single group /REPORT page*

**Column A** may contain only equations (query questions, text equations, etc.) which will be used in extracting information from the database(s).

**Column B** will contain the names you wish to assign to the results. These names are important for two reasons. They will become the default column headers and the item value names used in /LAYOUT. The names also define which items can be sorted.

**Column C** can contain only sorting information. Within each group, items to be sorted are identified as "1a","2d","3a",etc. The number refers to the level of the sort (1=first level), "a" & "d" meaning ascending and descending, respectively. Each group level may have up to nine sort levels. Group 1 sort items take precedence over Group 2 items, Group 2 items take precedence over Group 3, and so on. The maximum number of sort levels within one report is 20. Sort level requests greater than 20 are simply ignored.

**Column D** will contain the widths you wish to associate with each item on the printout. Default widths are 36 characters for text items, 10 characters for numeric items. You may respecify these widths if you find them inappropriate.

Looking at the example on the opposite page, you'll notice that basically only columns A and B are filled in. These two columns must have something in them or you don't have a report. Columns C and D, on the other hand, are only optionally used.

The detail list report writer uses two auxiliary temporary files for each page in QueryCalc that is defined to be a /REPORT page. The first is a flat file which records all of the extracted data. This intermediate file is also used for sorting. Whenever the page is recalculated (!!), this temporary file is rebuilt. If you make any changes on the /REPORT page (other than in sort column C or simple formatting changes in column A), the file is purged and must be recalculated again.

Once the data file exists though, you may /PRINT the report page repeatedly without having to recalculate the data. This will prove to be particularly useful when reformatting the output using the /LAYOUT command.

Although the intermediate data file is not meant to be seen, it can be read by EDITOR/3000, and you may want to look at it. It's name is QCRPLST*x*, where *x* is the report page letter of interest.

The second temporary file stores the formatting information used in the formatting command /LAYOUT. Again, one file exists for each /REPORT page. Typing /LAYOUT invokes QueryCalc's internal editor and brings up the current layout file for the page. The first time a report page is laid out, QueryCalc will create a default formatting file for you based on the names and widths specified on the report page. Because of this default feature, it is possible—and many times desirable—to skip the layout step and simply print the results of the recalculated report page.

## Examples of Report Pages

To illustrate the operation of /REPORT pages, I will refer to example reports found in the training account, QCDEMO.AICS. The file of interest is called DEMORPRT. If you wish to follow along in QueryCalc, type:

```
:hello user.aics
:run qc.qcprogs.aics
   /load demorprt
```

All of the examples referred to in the remainder of this chapter are pages in DEMORPRT.

## Pages A & B: A Simple, Single Group Report

The demonstration database, QCDEMO, is an actual construction company's database. The first two report pages of DEMORPRT were put together to list all of the suppliers to the construction company, in the alphabetical order of their keynames, and the total amount of money spent with each supplier over the years.

```
A |___________equation (a)_____________|____name (b)_____|_sort (c)__|_width (d)
1|_____________________find when keyname<>0;limit=100_____________________
2|                  (Aa2)               keyname              1a
3|                  (Aa3)               name
4|                  (Aa4)               address
5|                  (Aa5)
6|                  (Aa6)
7|                  (Aa7)
8|                  (Aa8)               city
9|                  (Aa9)               acctcode
10|                (Aa10)               amount
11|
```

```
Equations for Page A
```

```
   (Aa1): @Using qcdemo.accounts, find when keyname<>0;limit=100
   (Aa2): @rereading, val of keyname
   (Aa3): @rereading, val of description1
   (Aa4): @rereading, val of address
   (Aa5): @rereading, val of city
   (Aa6): @rereading, val of state
   (Aa7): @rereading, val of zip
   (Aa8): $A5+", "+A6+"  "+A7
   (Aa9): @rereading, val of acctcode
  (Aa10): @Using qcdemo.invoices, sum of amount when acctcode is [a9]
```

*The /PRINTEQ equations for Page A*

Pages A & B of the file DEMORPRT are almost identical. The only difference between the two pages is that page A uses the default layout format created by QueryCalc while page B uses an edited formatting structure that was put together in just a few minutes. The two layout files (QCREPRTA and QCREPRTB) and their resulting outputs appear on the following pages.

There is an important take-home-lesson in this first example: *Do not confuse a single-group report with single-dataset access.* The example above accesses two datasets, ACCOUNTS and INVOICES. The datasets are related by the dataitem ACCTCODE, which appears in both datasets. A second group

would be necessary only if you wanted a detail listing of each invoice from each vendor, not simply a summarization as used here.

Cell A9 of the report page retrieves the dataitem ACCTCODE. This value is the "link" between datasets. Although its retrieval is necessary to join the information in the two datasets, it is not necessary that the linking item be given a name in Column B. If you did not need ACCTCODE to appear on the printed report, nor wanted to use it as a sorting key, then no reason exists for you to give it a specific name.

The names you choose for Column B are abitrary. They do not have to reflect the dataitems being retrieved. Those values which are named are the values which will be written onto the intermediate flat file. The values can be the direct results of query questions (as in line A2) or the calculated results of text and numeric equations (as in line A8).

The list generator in QueryCalc is built using the same basic ideas employed in the UDQFs (user defined query functions). Each group level follows this basic pattern:

- The first line of a new group will always be a @FIND query question. This query indentifies the records to be selected. *If the group level is greater than the first level, a portion of the qualifying requirements for the* @FIND *are expected to come from the information in the group level immediately above it.*

- Cells filled with @REREADINGs will then follow, extracting required dataitems from the dataset being searched by the @FIND. Before the /REPORT page is executed, QueryCalc compiles the page. *For the sake of computational efficiency, it is best to group all of the @REREADINGs together.*

- Following the @REREADINGs may come numeric equations, text equations and other summarizing query questions referencing other datasets and databases.

```
keyname                          name                                    ad

ADAMSMOULDING                    ADAMS MOULDING & LUMBER CO. INC.        13
ADDCO                            ADDCO MILLWORK                          13
ADVENTURE                        ADVENTURE TRAVEL AGENCY, INC.           13
AICS                             AICS                                    13
ALLEY                            ALLEY-CONNELL & ASSOCIATES              13
ALLREDSPECIAL                    ALLRED'S AUTO SPECIALTIES               13
AMERICANLINEN                    AMERICAN LINEN SUPPLY                   13
AMIGO                            AMIGO LEASING, INC.                     13
APSCO                            ALAMO PIPE & SUPPLY CO., INC.           13
ARGYLE                           ARGYLE WELDING & MEDICAL SUPPLY CO.     13
BLUPRINTS                        BLUEPRINTS & MAPS INC.                  13
BOTSFORD                         BOTSFORD LAND SURVEYING INC.            13
BURKE                            THE BURKE COMPANY                       13
CASH                             PETTY CASH                              13
CENTURYDOOR                      WDC DOOR CORPORATION                    13
CONSTRSPECIAL                    CONSTRUCTION SPECIALTIES SUPPLIERS      13
CORDELL                          CORDELL PAVING COMPANY                  13
DASAVINGS                        DONA ANA SAVINGS & LOAN ASSOCIATION     13
DGUILLEN                         DANIEL S. GUILLEN                       13
DODGEREPORTS                     F.W. DODGE DIVISION                     13
DONCUMMINGS                      DON J. CUMMINGS CO., INC.               13
DUMMY                            JOURNAL ENTRY                           13
ELDER                            ELDER EQUIPMENT LEASING, INC.           13
EPMACHINE                        EL PASO MACHINE & STEEL CO.             13
FALCON                           FALCON SPRINKLER SYSTEMS, INC.          13
FEDEXPRESS                       FEDERAL EXPRESS CORPORATION             13
FIESTAPOOL                       FIESTA POOLS                            13
FRONTIER                         FRONTIER ROOFING CO. OF EL PASO         13
G&GFLOOR                         G & G FLOOR COVERINGS                   13
GASCO                            GAS COMPANY OF NEW MEXICO               13
GE                               GENERAL ELECTRIC COMPANY                13
GLOVER                           GLOVER PLUMBING & HEATING CO., INC.     13
```

```
QCREPRTA
   10  GROUP 1
   20    H
   30    H,"keyname",1;"name",39;"address",77;"city",115;"acctcode",-160&
           ;"amount",-170
   40    H
   50    D,KEYNAME,1;NAME,39;ADDRESS,77;CITY,115;ACCTCODE,-160;AMOUNT,-170
   60    F,"---------",-160;"---------",-170
   70    F,ACCTCODE,-160,SUM;AMOUNT,-170,SUM
   80    F
   90  ENDGROUP
```

*The printout and default layout format (QCREPRTA) for Page A.* QueryCalc automatically trims the printed line to the width of the margins specified in /MARG. The 36-character widths associated with the first several items results from the default width being assigned to text items. Footer sums are automatically generated in the default layout format for all numeric items.

```
--------------------------------------------------------------------------------

ADAMSMOULDING       ADAMS MOULDING & LUMBER CO. INC.       ACCOUNT CODE: 346
                    1313 EAST MOCKINGBIRD
                    UNIVERSITY PARK, NM  88003
                    AMOUNT: 3,606.82

--------------------------------------------------------------------------------

ADDCO               ADDCO MILLWORK                         ACCOUNT CODE: 22
                    1313 EAST MOCKINGBIRD
                    UNIVERSITY PARK, NM  88003
                    AMOUNT: 44,403.79

--------------------------------------------------------------------------------

ADVENTURE           ADVENTURE TRAVEL AGENCY, INC.          ACCOUNT CODE: 463
                    1313 EAST MOCKINGBIRD
                    UNIVERSITY PARK, NM  88003
                    AMOUNT: 514.00

--------------------------------------------------------------------------------

AICS                AICS                                   ACCOUNT CODE: 115
                    1313 EAST MOCKINGBIRD
                    UNIVERSITY PARK, NM  88003
                    AMOUNT: 21,723.42

--------------------------------------------------------------------------------

ALLEY               ALLEY-CONNELL & ASSOCIATES             ACCOUNT CODE: 145
                    1313 EAST MOCKINGBIRD
                    UNIVERSITY PARK, NM  88003
                    AMOUNT: 1,276.40
```

```
QCREPRTB
   10  GROUP 1
   20    d,"------------------------------------------------------------&
           ------------------------",1;unify 6
   30    d
   40    D,KEYNAME,1;NAME,20;"ACCOUNT CODE:",-70;acctcode,72
   50    d,address,20
   60    d,city,20
   70    d,"AMOUNT:",20;amount,28
   80    d
   90  endgroup
```

*The printout and modified layout format (QCREPRTB) for Page B.* The header lines were eliminated and data item values were redistributed on the report. The report layout is now composed solely of detail lines. In a layout file, each line of H, D or F indicates one print line. A strict requirement is that each group's lines must have all of the H's together, followed by all of the D's and then all of the F's. The syntax checker in QueryCalc's editor will insure that this required order is observed. Notice that type case (upper or lower) is unimportant in the layout file.

# How to Build a Detail List Report Step-by-Step

1. Go to the spreadsheet page you wish to make a report page and type /REPORT.

2. Enter the first @FIND equation. You will find it easiest to work with real data, so terminate the @FIND with an "!" to cause your query to execute immediately. If the @FIND is likely to take some time, append a temporary LIMIT clause to the end of the query statement to limit the number of records you will find.

3. Proceeding down Column A, enter the @REREADING cells, terminating each query with an "!". Format the cells (using /FORM) in the fashion you wish the data to appear on the report.

4. Enter any other summary queries from other databases or datasets in Column A.

5. If you desire more complex results derived from several of the query values, enter the necessary equations in Column A.

6. If another group level of detail is necessary, enter the second @FIND equation. A portion of the qualifying requirements should come from information in the cells of the immediately preceeding group. Repeat the procedure in Steps 2 to 5.

7. Go to Column B and provide names (typed in as text labels) for those items you wish to appear on the printed report or which will be used as sort items.

8. Go to Column C and optionally enter the sort order (typed in as text labels). Nine sort levels (1 through 9) can be specified within each group.

9. Go to Column D and optionally enter any width values you wish to change from the default values of 36 characters for text and 10 characters for numbers.

**10.** Recalculate the report page by typing "!!"

**11.** /PRINT the report to see the results of your first try.

**12.** Type /LAYOUT to modify the default layout format.

**13.** /PRINT the modified report. Repeat steps 12 & 13 until you are satisfied with the results.

---

## The Types of Lines Which May Appear in a Layout File

The line types which are allowed in a **/LAYOUT** file are these:

| | |
|---|---|
| GROUP | BREAKON |
| H (header lines) | ENDBREAK |
| D (detail lines) | ONNEWPAGE |
| F (footer lines) | ENDPAGE |
| ENDGROUP | |

**Header lines** are composed of only text labels, print positions, and optionally, print enhancement requests.

```
H
H,"Date",30;"Amount",-50,b0,s2
H,"The Bayard Company, Ltd.",c
```

A single letter "H" indicates a blank print line in the headers. A header line with quoted text followed by a positive number indicates that the text is to be left justified, beginning at the position indicated. A negative number indicates that the text is to be right justified, the rightmost character to be placed at the position indicated, with the text going "backwards". Other acceptable print position values are R,L, and C. "L" left justifies text at print position of 1. "R" right justifies text at the current right margin (as set in /MARG). "C" center justifies the text between the present left and right margin settings. These print position conventions are used in all of three line types (H,D,F).

The optional *print enhancement codes* are the same codes used for spreadsheet cell enhancements (bold, underlining, font changes, etc.) (see Chap. 8, "Printing"). As with cell enhancements, the print enhancement tends to

stay on once its been turned on, although this behavior varies a bit from printer type to printer type. In general, you must turn the enhancement off at the next printable item if you do not want it to continue indefinitely.

**Detail lines** are basically the same as header lines, but with the optional addition of the item values named on the /REPORT page.

```
D
D,"Category:",-60;CATEGORY,62
D,"------item separator-------",c
D,date,14,nr;amount,-30
```

A detail line may be composed of either simple text (denoted by quotes) or item value names (no quotes). The format is otherwise identical to header lines, with one exception. The value "**NR**" may also be added to any detail line. "NR" means non-repeating. An example appears in the last line of the four lines above. Should the same date value immediately recur, the date value will be printed only the first time it appears. Because NR was not specified for the amount value, every value of amount will be printed. The result will look like this:

```
1/14/93        12,237.33
                6,781.01
               34,676.00
1/16/93            45.12
                  123.89
               18,345.40
```

**Footer lines** are similar to detail lines. They may be composed of either simple text or item value names.

```
F
F,"----------",-60;"----------",-80
F,"Grand Total: ",-48;AMOUNT,-60,sum;&
  BALANCE,-80,sum,b1;"",-81,b0
```

Simple text only has one necessary parameter: print position. A summarized report item value has two: (1) print position and (2) the summarization type to be accumulated (sum, num, avg, max, min, var, or dev). As with header and detail lines, footer lines may also be enhanced. In the example above, BALANCE is printed in bold. [Notice that the bold enhancement request (b1) is turned off in the next print item (b0), an empty text string].

## Layout Print Control Structures

Five print control structures for detail list reports exist in QueryCalc. Four of them are phrases used in header, detail or footer lines:

```
NEWPAGE
RESET 15
UNIFY 6
SKIPTO 5
```

The fifth is a conditional clause used in the /LAYOUT file:

```
ONNEWPAGE
   •
ENDPAGE
```

**NEWPAGE**, when placed anywhere within a printable line, causes the printer to skip to the top of a new page before the line is printed.

**RESET 15**, when placed anywhere within a printable line, causes the page number of the current page to be reset to the number specified.

**UNIFY 6**, when placed anywhere in a line, requires that at least 6 full print lines remain before bottom of page or the printer will be commanded to skip to the next page. The effect is that the present line and the following five lines will always be printed as a whole. Any positive integer may be specified in the UNIFY command. Negative numbers or numbers larger than or equal to the number of printable lines on the page (as set in /MARG) will be ignored. The UNIFY command stays in effect only for the duration of printing of the line in which it appears.

**SKIPTO 55**, when placed anywhere in a line, causes the printer to skip print row 55 before the line is printed. If the requested print row has already been passed, the printer will skip to the next page and go to the requested print row on that page. Negative line numbers or numbers larger than the number of printable lines on a page (as set in /MARG) will be ignored.


NOTE

Should you request both NEWPAGE and SKIPTO in the same line, the order of execution will be first NEWPAGE and then SKIPTO. That is, the printer will first go the top of the next page and then skip down to the requested print row before the line is printed. Should you request both UNIFY and SKIPTO in the same line, UNIFY overrides SKIPTO. That is, if you request a

skip to a print row within the number of page bottom rows protected by UNIFY, UNIFY will cause the printer to skip to the top of the next page.

## The Conditional Print Control Clause

**ONNEWPAGE** is a conditional clause. ONNEWPAGE causes a secondary header to be printed at the top of each continuation page. ONNEWPAGE takes effect only when the report fills the current page and the printer "walks" into a new page. That may occur because either too much information has been retrieved to fit on the current page or the SKIPTO phrase has caused the printer a skip to the next page. The ONNEWPAGE clause, it should be noted, is not activated on a NEWPAGE command. Indeed, NEWPAGE resets the ONNEWPAGE trigger.

A different ONNEWPAGE clause may be specified for each group in the report. If no ONNEWPAGE clause is specified for the group level currently printing, the last ONNEWPAGE encountered will be the one used.

The rules for using the ONNEWPAGE clause are these:

1. Only header line types are allowed in an ONNEWPAGE clause.

2. The ONNEWPAGE clause is always placed at the end of a group, just prior to the ENDGROUP command.

3. An ONNEWPAGE set of header lines used in a lower group can be cleared by setting a null ONNEWPAGE clause in a higher group (that is , the ONNEWPAGE command is followed immediately by an ENDPAGE command).

Examples of all of the print control structures are used in Pages C & D of DEMORPRT, sections of which appear in the next several pages.

## Global Headers & Footers

Global headers and footers for each report page (such as the footer page number in the example file, DEMORPRT) are set as they would normally be set for spreadsheet pages, in the /HEAD screen. And as they do for spreadsheet pages, headers and footers diminish the number of printable lines on the page by the number of lines they take up.

# Example 2: A Two-Group Expense Report

Pages C & D of DEMORPRT are identical pages, other than their layout files. The detail lists they generate are a list of all occupancy expenses for the year 1984. The report page is organized as two groups, one which extracts each category of expense, the other each invoice which occurred within 1984 for that category. The categories are sorted by name. The invoices are sorted by date. Three datasets are accessed by the report.

```
C |__________equation (a)_____________|___name (b)____|_sort (c)__|_width (d)
1|_____________________find when category ib 500,600_____________________
2|                  (Ca2)
3|                  (Ca3)
4|                  (Ca4)               category            1a          25
5|________________find when category is [a2] and date ib 8..._____________
6|                  (Ca6)               compdate            1a          10
7|                  (Ca7)               amount
8|                  (Ca8)               balance
9|                  (Ca9)
10|                (Ca10)
11|                (Ca11)               supplier                        30
12|                (Ca12)               date                            15
13|
```

Equations for Page C

```
   (Ca1): @Using qcdemo.categories, find when category ib 500,600
   (Ca2): @rereading, val of category
   (Ca3): @rereading, val of description1
   (Ca4): $NAM$(A3)
   (Ca5): @Using qcdemo.invoices, find when category is [a2] and
          date ib 840000,850000
   (Ca6): @rereading, val of date
   (Ca7): @rereading, val of amount
   (Ca8): @rereading, val of balance
   (Ca9): @rereading, val of acctcode
  (Ca10): @Using qcdemo.accounts, val of description1 when
          acctcode is [A9]
  (Ca11): $NAM$(A10)
  (Ca12): $DAT$4(A6)
```

*The /PRINTEQ equations for Page C*

Two cells, A2 and A9, are used to retrieve information (expense category number and account code) which is used to link datasets together. Neither cell is named in Column B. There is no need to name these cells because this linking information will neither need to appear on the printed report or be used in sorting.

```
QCREPRTC
  10     GROUP 1
  20       H
  30       H,"category",1
  40       H
  50       D,CATEGORY,1
  60       GROUP 2
  70         H
  80         H,"compdate",4;"amount",-23;"balance",-33;"supplier",36;"date",68
  90         H
 100         D,COMPDATE,4;AMOUNT,-23;BALANCE,-33;SUPPLIER,36;DATE,68
 110         F,"---------",-23;"---------",-33
 120         F,AMOUNT,-23,SUM;BALANCE,-33,SUM
 130         F
 140       ENDGROUP
 150       F
 160       F
 170       F
 180     ENDGROUP
```

*The default layout format (QCREPRTC) for Page C.* The default format automatically generates sums of all of the numeric items within a group. The default format also automatically indents each new group two spaces to the right.

Items are presented in the default format in the order that they appear on the /REPORT page. The item COMPDATE was given a name not because it was desired to have it on the printed report, but because its presence was necessary for sorting. One of the first items to be modified in the layout in Page D was to drop COMPDATE and move DATE to take its place.

category

Building Maintenance

| compdate | amount | balance | supplier | date |
|---|---|---|---|---|
| 840101 | 100.00 | 0.00 | Ccc Investments | JAN 01 84 |
| 840201 | 100.00 | 0.00 | Ccc Investments | FEB 01 84 |
| 840301 | 100.00 | 0.00 | Ccc Investments | MAR 01 84 |
| 840401 | 100.00 | 0.00 | Ccc Investments | APR 01 84 |
| 840418 | 10.40 | 0.00 | Marion H. Phares | APR 18 84 |
| 840429 | 51.21 | 0.00 | Home Centers/Western Region | APR 29 84 |
| 840501 | 100.00 | 0.00 | Ccc Investments | MAY 01 84 |
| 840524 | 71.96 | 0.00 | Ccc Investments | MAY 24 84 |
| 840531 | 166.27 | 0.00 | Ccc Investments | MAY 31 84 |
| 840701 | 100.00 | 0.00 | Ccc Investments | JUL 01 84 |
| 840724 | 152.21 | 0.00 | Hargraves Drilling | JUL 24 84 |
| 840801 | 100.00 | 0.00 | Ccc Investments | AUG 01 84 |
| 840802 | 34.35 | 0.00 | Metal Craft Company | AUG 02 84 |
| 840831 | 59.90 | 0.00 | Metal Craft Company | AUG 31 84 |
| | 1,246.30 | 0.00 | | |

Cleaning Service

| compdate | amount | balance | supplier | date |
|---|---|---|---|---|
| 840103 | 15.00 | 0.00 | Natalia Vasquez | JAN 03 84 |
| 840109 | 15.00 | 0.00 | Natalia Vasquez | JAN 09 84 |
| 840116 | 15.00 | 0.00 | Natalia Vasquez | JAN 16 84 |
| 840123 | 15.00 | 0.00 | Natalia Vasquez | JAN 23 84 |
| 840130 | 10.00 | 0.00 | Natalia Vasquez | JAN 30 84 |
| 840206 | 15.00 | 0.00 | Natalia Vasquez | FEB 06 84 |
| 840213 | 15.00 | 0.00 | Natalia Vasquez | FEB 13 84 |
| 840220 | 15.00 | 0.00 | Natalia Vasquez | FEB 20 84 |
| 840227 | 20.00 | 0.00 | Natalia Vasquez | FEB 27 84 |
| 840305 | 15.00 | 0.00 | Natalia Vasquez | MAR 05 84 |
| 840312 | 20.00 | 0.00 | Natalia Vasquez | MAR 12 84 |
| 840319 | 15.00 | 0.00 | Natalia Vasquez | MAR 19 84 |
| 840326 | 15.00 | 0.00 | Natalia Vasquez | MAR 26 84 |
| 840403 | 10.00 | 0.00 | Natalia Vasquez | APR 03 84 |
| 840409 | 10.00 | 0.00 | Natalia Vasquez | APR 09 84 |
| 840416 | 20.00 | 0.00 | Natalia Vasquez | APR 16 84 |
| 840423 | 15.00 | 0.00 | Natalia Vasquez | APR 23 84 |
| 840430 | 20.00 | 0.00 | Natalia Vasquez | APR 30 84 |
| 840507 | 20.00 | 0.00 | Natalia Vasquez | MAY 07 84 |
| 840515 | 10.00 | 0.00 | Natalia Vasquez | MAY 15 84 |
| 840521 | 10.00 | 0.00 | Natalia Vasquez | MAY 21 84 |
| 840529 | 15.00 | 0.00 | Natalia Vasquez | MAY 29 84 |
| 840604 | 15.00 | 0.00 | Natalia Vasquez | JUN 04 84 |
| 840611 | 20.00 | 0.00 | Natalia Vasquez | JUN 11 84 |
| 840618 | 15.00 | 0.00 | Natalia Vasquez | JUN 18 84 |
| 840627 | 15.00 | 0.00 | Natalia Vasquez | JUN 27 84 |
| 840702 | 10.00 | 0.00 | Natalia Vasquez | JUL 02 84 |
| 840710 | 10.00 | 0.00 | Natalia Vasquez | JUL 10 84 |

*The default printout for Page C.*

```
QCREPRTD
   100  GROUP 1
   110    d,newpage;"-------------------------------------------------------------------&
            --------------------",1
   120    d
   130    d,"UniCon Construction, Ltd",c,b1,s2
   140    d,"University Park, New Mexico",c
   150    d,"",1,b0,s0
   160    d,"-------------------------------------------------------------------------&
            -------------",1
   170    d
   180    D,"Category:",1;CATEGORY,11,m2;"",r,m1
   190    GROUP 2
   200      H
   210      H,"date",4;"amount",-23;"balance",-33;"supplier",36
   220      H
   230      D,unify 3;DATE,4;AMOUNT,-23;BALANCE,-33;SUPPLIER,36,m2;"",m1,r
   240      f,"---------",-23;"--------",-33
   250      f,"Total:",-13;amount,-23,sum;balance,-33,sum
   260      f
   270      f,skipto 50;"Count:",-53;amount,-63,num;balance,-73,num
   280      f,"Average:",-53;amount,-63,avg;balance,-73,avg
   290      f,"Maximum:",-53;amount,-63,max;balance,-73,max
   300      f,"Minimum:",-53;amount,-63,min;balance,-73,min
   310      f,"Variance:",-53;amount,-63,var;balance,-73,var
   320      f,"Deviation:",-53;amount,-63,dev;balance,-73,dev
   330      F
   340      ONNEWPAGE
   350        h,"-----------------------------------------------------------------&
                ---------------",1
   360        h
   370        h,"UniCon Construction, Ltd",c,b1,s2
   380        h,"University Park, New Mexico",c
   390        h,"",b0,s0
   400        h,"-------------------------------------------------------------------&
                -----------------",1
   401        h,skipto 9
   410      ENDPAGE
   420    ENDGROUP
   430    f,newpage;skipto 50;"-------------------------------------------------------
            ------------------------------",1
   440    f
   450    f,"Total:",-13;amount,-23,sum;balance,-33,sum;"GRAND TOTALS",r
   460    f
   470    f,"Count:",-13;amount,-23,num;balance,-33,num
   480    f,"Average:",-13;amount,-23,avg;balance,-33,avg
   490    f,"Maximum:",-13;amount,-23,max;balance,-33,max
   500    f,"Minimum:",-13;amount,-23,min;balance,-33,min
   510    f,"Variance:",-13;amount,-23,var;balance,-33,var
   520    f,"Deviation:",-13;amount,-23,dev;balance,-33,dev
   530  ENDGROUP
```

*The modified layout format (QCREPRTD) for page D.*

**UniCon Construction, Ltd**
**University Park, New Mexico**

Category: *Building Maintenance*

| date | amount | balance | supplier |
|---|---|---|---|
| JAN 01 84 | 100.00 | 0.00 | *Ccc Investments* |
| FEB 01 84 | 100.00 | 0.00 | *Ccc Investments* |
| MAR 01 84 | 100.00 | 0.00 | *Ccc Investments* |
| APR 01 84 | 100.00 | 0.00 | *Ccc Investments* |
| APR 18 84 | 10.40 | 0.00 | *Marion H. Phares* |
| APR 29 84 | 51.21 | 0.00 | *Home Centers/Western Region* |
| MAY 01 84 | 100.00 | 0.00 | *Ccc Investments* |
| MAY 24 84 | 71.96 | 0.00 | *Ccc Investments* |
| MAY 31 84 | 166.27 | 0.00 | *Ccc Investments* |
| JUL 01 84 | 100.00 | 0.00 | *Ccc Investments* |
| JUL 24 84 | 152.21 | 0.00 | *Hargraves Drilling* |
| AUG 01 84 | 100.00 | 0.00 | *Ccc Investments* |
| AUG 02 84 | 34.35 | 0.00 | *Metal Craft Company* |
| AUG 31 84 | 59.90 | 0.00 | *Metal Craft Company* |
| Total: | 1,246.30 | 0.00 | |

| | | |
|---|---|---|
| Count: | 14.00 | 14.00 |
| Average: | 89.02 | 0.00 |
| Maximum: | 166.27 | 0.00 |
| Minimum: | 10.40 | 0.00 |
| Variance: | 1,734.08 | 0.00 |
| Deviation: | 41.64 | 0.00 |



*The modified printout for Page D.*

## Printing to Preprinted Forms

The printout for Page D looks very much like the format which would be used to fill in a standard invoice. With the use of the SKIPTO command phrase, printing can be commanded to occur at any specific location on a preprinted form. Because you rarely know beforehand how many items will fill the invoice, the use of the UNIFY command phrase will allow you to set an artificial "bottom of page", thus protecting a print area which would normally contain summations, taxes, etc. Using the ONNEWPAGE clause, a continuation header can be printed at the top of each new page of an invoice which has too many items to fit on just one page.

Similarly, filling out other preprinted forms, such as W-2 reports, unemployement reports, wage withholding reports, etc. are as easily done.

## Check Registers & Repeating Dataitems

Check reconciliation registers are also readily programmable. In a check register, items from several groups are often desired to be printed on the same line. That's possible in QueryCalc, given a few constraints. The first constraint is that higher group level items can be printed in lower groups, but the reverse is not true (the reason: a higher group value will always be known by the time the lower group element is to be printed, but QueryCalc cannot know in advance the value of the lower group item before it is encountered). The syntax checker in /LAYOUT will prevent you from describing a format so that you would violate this rule. The second constraint is that all of the items must be capable of appearing on the same line(s). There is no way in QueryCalc for you to have a column of check amounts printed at the right and on the left print a shorter multi-line description of the account, such that both columns will be flush at the top. To do that would require multiple data files with synchronizing print control structures, a feature QueryCalc does not possess.

However the use of the non-repeating (**NR**) modifier offers a range of possibilities. The default layout of Page C (DEMORPRT) was modified to produce the output on the opposite page. The item value CATEGORY, a Group 1 item, has been moved down to Group 2. This will cause the current value of CATEGORY to be printed on every Group 2 line. By specifiying the NR modifier for CATEGORY, its value is only printed once at every group value change. This allows different group item values to be printed simultaneously on the same line. Notice too that the SUPPLIER item value is also printed with the **NR** modifier. This allows the opportunity to easily recognize a change in vendors.

```
Category                Date          Amount   Balance Supplier

Building Maintenance    JAN 01 84     100.00      0.00 Ccc Investments
                        FEB 01 84     100.00      0.00
                        MAR 01 84     100.00      0.00
                        APR 01 84     100.00      0.00
                        APR 18 84      10.40      0.00 Marion H. Phares
                        APR 29 84      51.21      0.00 Home Centers/Western R
                        MAY 01 84     100.00      0.00 Ccc Investments
                        MAY 24 84      71.96      0.00
                        MAY 31 84     166.27      0.00
                        JUL 01 84     100.00      0.00
                        JUL 24 84     152.21      0.00 Hargraves Drilling
                        AUG 01 84     100.00      0.00 Ccc Investments
                        AUG 02 84      34.35      0.00 Metal Craft Company
                        AUG 31 84      59.90      0.00

Cleaning Service        JAN 03 84      15.00      0.00 Natalia Vasquez
                        JAN 09 84      15.00      0.00
                        JAN 16 84      15.00      0.00
                        JAN 23 84      15.00      0.00
                        JAN 30 84      10.00      0.00
                        FEB 06 84      15.00      0.00
                        FEB 13 84      15.00      0.00
                        FEB 20 84      15.00      0.00
                        FEB 27 84      20.00      0.00
                        MAR 05 84      15.00      0.00
                        MAR 12 84      20.00      0.00
                        MAR 19 84      15.00      0.00
                        MAR 26 84      15.00      0.00
                        APR 03 84      10.00      0.00
                        APR 09 84      10.00      0.00
                        APR 16 84      20.00      0.00
                        APR 23 84      15.00      0.00
```

```
QCREPRTC
    10  GROUP 1
    20    H
    30    H,"Category",1;"Date",27;"Amount",-45;"Balance",&
            -55;"Supplier",57
    40    H
    50    GROUP 2
    60      D,CATEGORY,1,NR;DATE,27;AMOUNT,-45;BALANCE,&
              -55;SUPPLIER,57,NR
    70      F
    80      ONNEWPAGE
    90        H
   100        H,"Category",1;"Date",27;"Amount",-45;"Balance",&
                -55;"Supplier",57
   110        H
   120      ENDPAGE
   130    ENDGROUP
   140  ENDGROUP
```

*The printout and layout format for a modified Page C.*

## Control Breaks

Quite often, you will need to impose additional subtotals and print controls in the rep
Four items, ACCTCODE, DATE, AMOUNT, and YEAR, are generated on the repor
A control break may be typed into the layout file in the following fashion:

```
F |__________equation (a)_____________|___name (b)____|_sort (c)__|_width (d)
1|_____________________________find when jobnum is 8404_____________________________
2|                  (Fa2)
3|                  (Fa3)                                    2a          10
4|                  (Fa4)               amount
5|                  (Fa5)               company              1a          20
6|                  (Fa6)               date                             10
7|                  (Fa7)               year
```

```
(Fa1): @Using qcdemo.invoices, find when jobnum is 8404
(Fa2): @rereading, val of acctcode
(Fa3): @rereading, val of date
(Fa4): @rereading, val of amount
(Fa5): @Using qcdemo.accounts, val of keyname when acctcode
       is [a2]
(Fa6): $DAT$5(A3)
(Fa7): YEAR(A3)
```

```
D,COMPANY,1,NR;DATE,23;AMOUNT,-42
BREAKON COMPANY
   F,"----------",-42
```

```
QCREPRTF
   10    GROUP 1
   20      H,"-------------------------------------------------------",1
   30      H,"company",1;"date",23;"amount",-42
   40      H,"-------------------------------------------------------",1
   50      H
   60      D,COMPANY,1,NR;DATE,23;AMOUNT,-42
   70      F,"----------",-42
   80      F,"job total:",-30;AMOUNT,-42,SUM
   90      F
  100      ONNEWPAGE
  110        H,"-------------------------------------------------------",1
  120        H,"company",1;"date",23;"amount",-42
  130        H,"-------------------------------------------------------",1
  140        H
  150      ENDPAGE
  160    ENDGROUP
```

*The layout file without control breaks.*

| company | date | amount |
|---|---|---|
| ARTGLASS | 01-MAR-95 | 1,321.45 |
| | 30-APR-95 | 317.00 |
| BORDERMACHINE | 26-FEB-95 | 46.09 |
| BOTSFORD | 30-APR-94 | 256.94 |
| | 01-JUN-95 | 73.33 |
| BUILDERSBLOCK | 25-APR-94 | 158.06 |
| | 31-MAY-94 | 16.68 |
| | 25-JUN-94 | 2,487.22 |
| | 25-AUG-94 | 15,372.90 |
| | 25-AUG-94 | 6,441.00 |
| | 25-SEP-94 | 2,660.00 |
| | 25-SEP-94 | 11.18 |
| | 25-OCT-94 | 28,311.90 |
| | 25-OCT-94 | 12,001.35 |
| | 25-NOV-94 | 24,852.00 |
| | 25-DEC-94 | 5.50 |
| | 25-DEC-94 | 11,207.15 |
| | 25-JAN-95 | 11,044.70 |
| | 25-JAN-95 | 21.24 |
| | 25-FEB-95 | 22.36 |
| | 25-FEB-95 | 13,841.50 |
| | 25-FEB-95 | 1,500.32 |
| | 25-FEB-95 | 550.00 |
| | 24-MAY-95 | 9,424.23 |
| | 25-MAY-95 | 21.08 |
| | 25-MAY-95 | 299.53 |
| | 25-MAY-95 | 1,491.50 |
| | 25-JUL-95 | 18.00 |
| BURNCO | 04-APR-94 | 42,481.80 |
| | 03-MAY-94 | 12,234.60 |
| | 05-JUN-94 | 2,833.65 |
| | 06-SEP-94 | 1,450.35 |
| | 06-SEP-94 | 10,053.00 |
| | 12-MAR-95 | 31,476.60 |
| BUSINESSPROD | 10-APR-95 | 23.13 |
| CAMERASHOP | 26-MAR-94 | 4.17 |
| | 01-MAY-94 | 25.66 |
| CASHWAY | 27-JAN-94 | 35.01 |
| | 26-FEB-94 | 49.43 |
| | 29-APR-94 | 51.38 |
| | 27-MAY-94 | 64.40 |
| | 01-JUL-94 | 61.03 |
| | 01-JUL-94 | 13.11 |
| | 29-JUL-94 | 47.76 |
| | 26-AUG-94 | 7.54 |

*The printout of a single group report without control breaks.*

## The BREAKON Clause

A control break may be typed into the layout file in the following fashion:

```
D,COMPANY,1,NR;DATE,23;AMOUNT,-42
BREAKON COMPANY
   F,"---------",-42
   F,AMOUNT,-42,SUM
   F
ENDBREAK
F
```

The **BREAKON** clause is triggered by a change in the dataitem specified. The process is somewhat similar in effect to the non-repeating modifier (**NR**) that was discussed just a few pages ago.

```
QCREPRTF
   10   GROUP 1
   20     H,"-------------------------------------------------------",1
   30     H,"company",1;"date",23;"amount",-42
   40     H,"-------------------------------------------------------",1
   50     H
   60     D,UNIFY 8;COMPANY,1,NR;DATE,23;AMOUNT,-42
   70     BREAKON COMPANY
   80       F,"---------",-42
   90       F,"company total:",-30;AMOUNT,-42,SUM
  100       F,"company average:",-30;AMOUNT,-42,AVG
  110       F
  120       F,"-------------------------------------------------------",1
  130       F
  140     ENDBREAK
  150     F,"---------",-42
  160     F,"job total:",-30;AMOUNT,-42,SUM
  170     F
  180     ONNEWPAGE
  190       H,"-------------------------------------------------------",1
  200       H,"company",1;"date",23;"amount",-42
  210       H,"-------------------------------------------------------",1
  220       H
  230     ENDPAGE
  240   ENDGROUP
```

*The layout file has now been modified to have one control break.* The resulting output is shown on the opposite page. The /REPORT page was not recalculated to generate this new printout. Only the layout file was modified.

| company | date | amount |
|---|---|---|
| BOTSFORD | 30-APR-94 | 256.94 |
| | 01-JUN-95 | 73.33 |
| | company total: | 330.27 |
| | company average: | 165.14 |
| BUILDERSBLOCK | 25-APR-94 | 158.06 |
| | 31-MAY-94 | 16.68 |
| | 25-JUN-94 | 2,487.22 |
| | 25-AUG-94 | 15,372.90 |
| | 25-AUG-94 | 6,441.00 |
| | 25-SEP-94 | 2,660.00 |
| | 25-SEP-94 | 11.18 |
| | 25-OCT-94 | 28,311.90 |
| | 25-OCT-94 | 12,001.35 |
| | 25-NOV-94 | 24,852.00 |
| | 25-DEC-94 | 5.50 |
| | 25-DEC-94 | 11,207.15 |
| | 25-JAN-95 | 11,044.70 |
| | 25-JAN-95 | 21.24 |
| | 25-FEB-95 | 22.36 |
| | 25-FEB-95 | 13,841.50 |
| | 25-FEB-95 | 1,500.32 |
| | 25-FEB-95 | 550.00 |
| | 24-MAY-95 | 9,424.23 |
| | 25-MAY-95 | 21.08 |
| | 25-MAY-95 | 299.53 |
| | 25-MAY-95 | 1,491.50 |
| | 25-JUL-95 | 18.00 |
| | company total: | 141,759.40 |
| | company average: | 6,163.45 |
| BURNCO | 04-APR-94 | 42,481.80 |
| | 03-MAY-94 | 12,234.60 |
| | 05-JUN-94 | 2,833.65 |
| | 06-SEP-94 | 1,450.35 |
| | 06-SEP-94 | 10,053.00 |
| | 12-MAR-95 | 31,476.60 |
| | company total: | 100,530.00 |

*The report with one control break.*

## Multiple Control Breaks

Multiple control breaks may specified for a single report. The breaks are typed in a nested order such that the item that changes most often is specified first and the most major summarization appears last. A 2-break example is shown here:

```
QCREPRTF
   10   GROUP 1
   20     H,"-------------------------------------------------------",1
   30     H,"company",1;"date",23;"amount",-42
   40     H,"-------------------------------------------------------",1
   50     H
   60     D,UNIFY 8;COMPANY,1,NR;DATE,23;AMOUNT,-42
   70     BREAKON YEAR
   80       F,"---------",-52
   90       F,AMOUNT,-52,SUM
  100       F
  110       BREAKON COMPANY
  120         F,"company total:",-40;AMOUNT,-52,SUM
  130         F,"company average:",-40;AMOUNT,-52,AVG
  140         F
  150         F,"-------------------------------------------------------",1
  160         F
  170       ENDBREAK
  180     ENDBREAK
  190     F,"---------",-52
  200     F,"job total:",-40;AMOUNT,-52,SUM
  210     F
  220     ONNEWPAGE
  230       H,"-------------------------------------------------------",1
  240       H,"company",1;"date",23;"amount",-42
  250       H,"-------------------------------------------------------",1
  260       H
  270     ENDPAGE
  280   ENDGROUP
```

*The layout file modified a second time.* Two control breaks are now used, the first (minor) for the year and the second (major) for the company. Notice that YEAR is a dataitem that was calculated on the /REPORT page but is not otherwise used in the layout. This is perfectly legitimate.

| company | date | amount | |
|---|---|---|---|
| BUILDERSBLOCK | 25-APR-94 | 158.06 | |
| | 31-MAY-94 | 16.68 | |
| | 25-JUN-94 | 2,487.22 | |
| | 25-AUG-94 | 15,372.90 | |
| | 25-AUG-94 | 6,441.00 | |
| | 25-SEP-94 | 2,660.00 | |
| | 25-SEP-94 | 11.18 | |
| | 25-OCT-94 | 28,311.90 | |
| | 25-OCT-94 | 12,001.35 | |
| | 25-NOV-94 | 24,852.00 | |
| | 25-DEC-94 | 5.50 | |
| | 25-DEC-94 | 11,207.15 | |
| | | | 103,524.94 |
| | 25-JAN-95 | 11,044.70 | |
| | 25-JAN-95 | 21.24 | |
| | 25-FEB-95 | 22.36 | |
| | 25-FEB-95 | 13,841.50 | |
| | 25-FEB-95 | 1,500.32 | |
| | 25-FEB-95 | 550.00 | |
| | 24-MAY-95 | 9,424.23 | |
| | 25-MAY-95 | 21.08 | |
| | 25-MAY-95 | 299.53 | |
| | 25-MAY-95 | 1,491.50 | |
| | 25-JUL-95 | 18.00 | |
| | | | 38,234.46 |
| | | company total: | 141,759.40 |
| | | company average: | 6,163.45 |
| BURNCO | 04-APR-94 | 42,481.80 | |
| | 03-MAY-94 | 12,234.60 | |
| | 05-JUN-94 | 2,833.65 | |
| | 06-SEP-94 | 1,450.35 | |
| | 06-SEP-94 | 10,053.00 | |
| | | | 69,053.40 |
| | 12-MAR-95 | 31,476.60 | |
| | | | 31,476.60 |

*The report with control breaks for year and company.*

## Suppressing Detail Lines

In a "flat", one-group report, you may often wish to not only generate control break summaries, but also suppress the detail entries that contribute to those summary values. The supression of this unwanted information can be accomplished with the use of the "print-once" (**PO**) modifer as shown in line 60 of the /LAYOUT file below. The PO modifier works similarly to the NR modifier, but with PO, all further detail lines will be suppressed from printing until the specified item changes in value.

```
QCREPRTF
    10   GROUP 1
    20     H,"-------------------------------------------------",1
    30     H,"company",1;"year",-40;"amount",-52
    40     H,"-------------------------------------------------",1
    50     H
    60     D,UNIFY 8;COMPANY,1,PO
    70     D,ADDRESS,1
    80     BREAKON YEAR
    90       F,YEAR,-40;AMOUNT,-52,SUM
   100       BREAKON COMPANY
   110         F
   120         F,"total:",-40;AMOUNT,-52,SUM
   130         F,"average invoice:",-40;AMOUNT,-52,AVG
   140         F
   150         F,"-------------------------------------------------",1
   160         F
   170       ENDBREAK
   180     ENDBREAK
   190     F,"---------",-52
   200     F,"job total:",-40;AMOUNT,-52,SUM
   210     F
   220     ONNEWPAGE
   230       H,"-------------------------------------------------",1
   240       H,"company",1;"year",-40;"amount",-52
   250       H,"-------------------------------------------------",1
   260       H
   270     ENDPAGE
   280   ENDGROUP
```

*The layout file modified a third time*. The control breaks in this example are only very slightly modified from the previous example, but the detail lines have been print-suppressed. The detail lines now print only once, when the company name changes, the same condition that the major control break operates on.

```
company                                    year      amount
-------------------------------------------------------------

BORDERMACHINE
4545 E. AMADOR, LAS CRUCES 88001
                                           1995       46.09

                                         total:       46.09
                                 average invoice:     46.09

-------------------------------------------------------------

BOTSFORD
1326 FENTON, UNIVERSITY PARK 88003
                                           1994      256.94
                                           1995       73.33

                                         total:      330.27
                                 average invoice:     165.14

-------------------------------------------------------------

BUILDERSBLOCK
1200 LOHMAN, LAS CRUCES 88005
                                           1994  103,524.94
                                           1995   38,234.46

                                         total:  141,759.40
                                 average invoice:   6,163.45

-------------------------------------------------------------

BURNCO
HWY 70 AT EVANS, CHAMBERINO 88127
                                           1994   69,053.40
                                           1995   31,476.60

                                         total:  100,530.00
                                 average invoice:  16,755.00

-------------------------------------------------------------

BUSINESSPROD
12901 ESPINA, UNIVERSITY PARK 88003
                                           1995       23.13

                                         total:       23.13
                                 average invoice:     23.13
```

*The report modified so that the detail line prints only once.*

## Freedom of Suppress

If you wish to completely suppress the printing of detail lines, use the suppress (**SUPPRESS**) modifier. If the suppress modifier appears anywhere in a detail line, that line will not print.

When using the SUPPRESS modifier, you will almost certainly want the variable(s) that set the control break(s) to appear in one of the control break footer lines. In this example, YEAR appears in footer line 80 and COMPANY appears in line 110.

```
QCREPRTF
    10   GROUP 1
    20     H,"-------------------------------------------------",1
    30     H,"company",1;"year",-40;"amount",-52
    40     H,"-------------------------------------------------",1
    50     H
    60     D,UNIFY 8;SUPPRESS
    70     BREAKON YEAR
    80       F,YEAR,-40;AMOUNT,-52,SUM
    90       BREAKON COMPANY
   100         F
   110         F,COMPANY,1;"total:",-40;AMOUNT,-52,SUM
   120         F,"average invoice:",-40;AMOUNT,-52,AVG
   130         F
   140         F,"-------------------------------------------------",1
   150         F
   160       ENDBREAK
   170     ENDBREAK
   180     F,"---------",-52
   190     F,"job total:",-40;AMOUNT,-52,SUM
   200     F
   210     ONNEWPAGE
   220       H,"-------------------------------------------------",1
   230       H,"company",1;"year",-40;"amount",-52
   240       H,"-------------------------------------------------",1
   250       H
   260     ENDPAGE
   270   ENDGROUP
```

*The layout file modified a fourth time.* In this fourth example, the detail line has been completely suppressed from printing. However, the detail lines cannot simply be dropped. At least one detail line must still appear in the /LAYOUT file. No report variables need to appear on the detail line. Indeed, should they be present, they will merely represent wasted processing time.

| company | | year | amount |
|---|---|---|---|
| | | 1995 | 46.09 |
| BORDERMACHINE | | total: | 46.09 |
| | | average invoice: | 46.09 |
| | | 1994 | 256.94 |
| | | 1995 | 73.33 |
| BOTSFORD | | total: | 330.27 |
| | | average invoice: | 165.14 |
| | | 1994 | 103,524.94 |
| | | 1995 | 38,234.46 |
| BUILDERSBLOCK | | total: | 141,759.40 |
| | | average invoice: | 6,163.45 |
| | | 1994 | 69,053.40 |
| | | 1995 | 31,476.60 |
| BURNCO | | total: | 100,530.00 |
| | | average invoice: | 16,755.00 |
| | | 1995 | 23.13 |
| BUSINESSPROD | | total: | 23.13 |
| | | average invoice: | 23.13 |
| | | 1994 | 29.83 |
| CAMERASHOP | | total: | 29.83 |
| | | average invoice: | 14.91 |

*The report modified so that detail lines have been completely suppressed.*

## Rules for Control Breaks

There are only a very few rules for the use of control breaks in a layout file. You do not need to memorize them. QueryCalc will, as it does elsewhere, warn you if you violate them.

- The BREAKON clause may appear only after the lowest group's last detail line and just before the lowest group's first footer line, as shown in the previous examples.
- The items summed in the BREAKON clause may only reference items in the lowest level group (referencing higher level groups makes no logical sense).
- The number of groups and control breaks cannot exceed a total of 5 levels.
- Only footer lines may appear in a BREAKON clause.
- Summations in a BREAKON clause are identical to standard footers, thus you may ask for either the SUM, AVG, MAX, MIN, DEV or VAR of the item (or any combination of these items in one or more footer lines).
- The BREAKON item does not need to be a sorted item on the /REPORT page, but a report would rarely have a control break specified for a non-sorted item.

## Final Thoughts

Information for use in a report page may be extracted from any other cell in any spreadsheet or graphics definition page in the current file. However, the reverse is not true. In general, standard spreadsheet or graphic definition pages cannot extract information from a report page. The information in the cells of the report page is transitory and exists for only a short time. If you wish to use summarizations which were calculated during the course of calculation of a report page, no easy alternative exists but to recalculate those summaries in individual cells on a standard spreadsheet.

# Concepts Introduced in Chapter 13

| | |
|---|---|
| REPORT PAGE | the third page type in QueryCalc. The definition of a report page allows for the construction of a single, independent detail list report. Up to 26 such reports may be created in one QueryCalc file. |
| GROUP | a defined level of detail (one "indention" in an outline). Sorting occurs within a group. A group is defined on a /REPORT page by a new @FIND statement. |
| "LINKING" ITEM | a dataitem value which is common to two or more datasets (in the same or different databases). The "linking" item value logically "relates" the different datasets. |
| LAYOUT FILE | the format description file |
| UNIFY | a command which demands that a specified number of print lines exist before the bottom of page is to be encountered, otherwise it commands a skip to the top of the next page and resumes the print there. |
| ONNEWPAGE | a conditional print clause which activates a secondary header whenever printing "walks" into a new page. |

# 14

*Modifying detail list report layouts and macroprograms*

# QueryCalc's Editor

## A Brief Introduction to the Editor

QueryCalc's internal editor, which is used for both modifying report formats for detail list reports and for programming macros, possesses two modes of operation: *layout* and *macroprogramming*. The two modes are virtually identical, except for slight differences in allowed commands. The editor is invoked by typing any of the following at the *Command* line:

**/LAYOUT**
**/LAYOUT M**
**/NEWLAYOUT**
**/NEWLAYOUT B**
**/PROG**
**/PROG macrofilename**

The mode of the editor is determined by the form of the command which invoked the editor.

QueryCalc's editor was designed to look and operate much like the other common editors on the HP3000 (EDIT/3000, TDP, QUAD, etc.), being programmed to look most like EDIT/3000. However the editor has also been programmed to simultaneously operate as a line-number-oriented editor, similar to the internal editors found in most BASICs.

The editor holds a maximum of 500 lines of text. Each line is limited to 240 characters in length. Lines longer than 72 characters will be broken into appropriately sized line segments on the screen, terminated with "&" continuation characters. When entering a long line of text, you may either terminate your line segments with "&" continuation characters, or you may simply keep typing, letting the terminal wrap the line for you.

Line numbers may be manually or automatically assigned. Line numbers may range from 1 to 10,000 as integers.

## Commands Common to Both Edit Modes

*The standard editing commands are these:*

| | |
|---|---|
| **T[ext] filename** | loads the specified file |
| **A[dd]** | auto adds line(s) to current end of file. |
| **A[dd] 17** | auto adds line(s) at line 17 or next available line. Type "//" to quit line add. |
| **M[odify] 4** | modifies line 4 |
| **M[odify] 7/10** | modifies lines 7 thru 10 |
| **D[elete] 16** | deletes line 16 |
| **D[elete] 5/10** | deletes lines 5 thru 10 |
| **D[elete] ALL** | clears the program editor |
| **R[eplace] 5** | replaces line 5 |
| **R[eplace] 16/20** | replaces lines 16 thru 20 |
| **COPY 62 to 190** | copies line 62 to line 190 |
| **COPY 5/20 to 145** | copies lines 5 thru 20 to 145 |
| **S&R "old" "new" ALL** | replaces "old" with "new" everywhere |
| **S&R "in" "out" 5/20** | replace "in" with "out" in lines 5 thru 20 |
| **RENUM** | renumbers, starting at 10, step of 10 |
| **RENUM 1000** | renumbers, starting at 1000, step of 10 |
| **RENUM 500,100** | renumbers, starting at 500, step of 100 |
| **L[ist]** | lists the entire program |
| **L[ist] 5** | lists line 5 |
| **L[ist] 30/200** | lists lines 30 thru 200 |
| **P[rint]** | prints entire file to the system printer |
| **P[rint] 45/68** | prints lines 45 thru 68 |
| **C[at]** | displays the group's directory |
| **DIR** | displays the group's directory |
| **E[xit]** | exits the editor |

## Commands Unique to the Macro Mode

*These commands operate only in the editor's macro mode:*

| | |
|---|---|
| **K[eep] macroname**<br>**S[ave] macroname** | saves the specified macro in the group directory |
| **X{execute}** | executes the macro file currently loaded in the editor, whether it is saved or not, whether it is named or not |

## Commands Unique to the Layout Mode

*These commands operate only in the report layout mode:*

| | |
|---|---|
| **K[eep]**<br>**S[ave]** | saves the layout file. User specified names are disallowed. The *temporary* file created is named QCREPRT*x*, where *x* is the current report page. |
| **LAYOUT B** | loads the current layout file for page B. If the file does not exist, a default layout file is built based on the named items on Page B. |
| **NEWLAYOUT M** | clears the current layout file for Page M, if one exists, and builds a default layout file based on the named items on Page M. |

## Switching Modes While in the Editor

Texting in a macro file automatically converts the editor to the macro mode. Similarly, "layouting" a layout file converts the editor to the layout mode.

The editor can also be converted to the macro mode in "midstream", without texting in a macro file. Type: "D ALL" (delete and clear the editor file). The macro mode is the default mode for an "empty" editor. Should you wish to convert the editor to layout mode from macro mode, type "D ALL", followed by either "LAYOUT *x*" or "NEWLAYOUT *x*".

The principal difference between the two modes is that macro mode files are saved in QueryCalc's directory. In contrast, layout mode files are attached

to the spreadsheet pages themselves and never appear as independent files, although they are given temporary names (QCREPRT*x*) while the spreadsheet is loaded. Otherwise, layout and macro files are fundamentally the same.

## Modifying a Line in the Editor

The only editor command which is not self-explanatory is M[odify]. The modify command operates indentically to QueryCalc's cell editor (see Chap. 4, "Spreadsheet Operations") or MPE's :REDO line editor. An example of the modify command's use is shown on the opposite page for Line 100.

When a line has been selected with the modify command, the line will be written onto the terminal's screen and the cursor will be placed at the beginning of the line. Use the SPACE bar (*not the arrowed keys*) to space underneath the first character of the section of text you wish to modify.

The first letter you type at this point will determine the type of modification to be performed. "**I**" will cause any text following the "**I**" to be inserted at the position of the "**I**". "**R**" will replace a section of the text, character-for-character with any text you type following the "**R**". A string of "**D**"'s will delete text from the line, one character for each "**D**", for the length of the "**D**"'s. A string of "**D**"'s followed by an "**I**" will delete the marked text and then insert any text following the "**I**". "**A**" will append any text following the "**A**" to the end of the line. If the first letter you type is not one of these specific characters, the modify editor routine will simply replace the text, character-for-character, beginning at the present cursor location.

Only one modification can be made per spacing "pass". Press RETURN to cause your current modification to be inserted into the line. If other modifications are to be made to this line, repeat the procedure by spacing over to the next text segment to be modified.

Should the text line be longer than 72 characters and it "wraps" on the terminal screen, continue spacing until you also wrap and space underneath the text of interest.

To signal the editor that you find your modifications acceptable, press the RETURN key. This will cause the modified line to be written back into the edit file and return you to the ">" prompt.

To cancel a modification attempt without effect, type "//", followed by RETURN. This aborts the modify command, leaves the editor's text line unmodified, and returns you to the ">" prompt.

```
PHILMAC
    10  /clr all
    20  @opendb qcdemo/FRONT
    30  za1:@using employees, store in a socsecnum when numde&
        ductions ib 4,5!
    40  /j ab5
    50  listfile #1=!a
    60  on end of list #1 exit
    70  loop:
    80    read listfile #1
    90    *:'%listitem1
   100    *+1c:@using employees, val of lname when socsecnum=&
          [$*-1c]!
   110    *+1c:@using payrecord, sum of gross when socsecnum &
          is [$*-2c]!
   120    %col=2
   130    %row=%row+1
   140    if %row>20 then do
   150      %row=5
   160      %pag=%pag+1
   170    doend
   180    goto loop
   190  exit:

>m 100
   100  *+1c:@using employees, val of lname when socsecnum=&
        [$*-1c]!
                                        dddddilast-name
   100  *+1c:@using employees, val of last-name when socsec&
        num=[$*-1c]!

>l 70/120
    70  loop:
    80    read listfile #1
    90    *:'%listitem1
   100    *+1c:@using employees, val of last-name when socsec&
          num=[$*-1c]!
   110    *+1c:@using payrecord, sum of gross when socsecnum &
          is [$*-2c]!
   120    %col=2
```

*An example of the modification of Line 100.*

## Using Line Numbers

QueryCalc's internal editor not only operates using a command structure, but also simultaneously operates as a line-number-oriented editor. A line of text, when preceeded by a line number, commands the editor to operate on that line of text. Examples of such entries are:

```
143 %row=%row+5
250 "---line separator---",c
17
```

Line-numbered text may be typed into the file in any order. The line number will cause the text to be properly ordered. Typing a nonexistent line number will cause the entered text to be *inserted* between the appropriate line numbers. Typing a line number which already exists will *replace* the old line with the newly entered text. Typing simply a line number (as in the third example above) will cause the line with that number to be *deleted*, should it exist.

## Using the Terminal's Local Edit Functions

Line-number-oriented editors allow the use of the screen editing features which are common to all HP terminals (as well as all third-party workalike terminals and PC-based terminal emulation packages). *When the editor screen is brought up in QueryCalc, the function keys are automatically set to the local screen editing commands of the terminal.* To use these local editing features, press the [f1] function key ("LINE MODIFY"). You will see an asterisk appear in function key block on the screen. Use the *arrowed* keys to move the terminal's cursor to the line to be modified. Make any modifications to any line on the screen you wish. When you are finished, press RETURN. This action transmits the current cursor's line to the HP3000, exits the terminal's local edit mode, and returns you to QueryCalc's editing prompt ( a ">").

If the transmitted line should end in an "&" continuation character, a ">>" prompt will appear. This second prompt requests more text. You may either: (1) complete the line manually by typing in the remaining text, (2) move the cursor to the remaining line segment visible on the screen and press the ENTER key, or (3) you may press [f1], move the cursor to the remaining line segment, modify the segment, and then press RETURN to transmit it.

# Concepts Introduced in Chapter 14

| | |
|---|---|
| LAYOUT MODE | the editor mode which has been optimized to edit detail list report layout files. |
| MACRO MODE | the editor mode which has been optimized to edit macroprograms. |
| COMMAND MODE | editing a file through the use of EDIT/3000-like commands. |
| LINE NUMBER MODE | using line-numbered text to insert, replace and delete lines from the edit file. |

# 15

*Reading in data from files, programs, & databases*

# External File Reads

## Reading Bulk Data into the Spreadsheet

Three numeric functions, which may appear in any cell, allow you to read external data into a QueryCalc spreadsheet. They are:

| | |
|---|---|
| **READ** | reads data from a specified flat ASCII file into the spreadsheet. |
| **CALL** | executes an external, user-defined routine and imports its data into the spreadsheet. |
| **REPORT** | reads data created on a /REPORT page onto the spreadsheet |

All three functions are similarly constructed. The READ function is the model for the other two. READ reads data onto the spreadsheet from a user-specified external ASCII (flat) file. Two formats are available: comma-delimited format (CDF), used for both text and numbers, and restricted text (RTX) for text-only files.

The CALL function calls an external, user-defined program or subroutine (SL-resident). The user-defined routine is expected to write data into a pre-defined ASCII flat file, QCOUTPUT, which is then read into the spreadsheet in the same manner as READ.

The REPORT function reads data from the intermediate flat file that is created during the execution of a /REPORT page (see pages 13-4, 13-5). The data values to be read onto the spreadsheet are those associated with the names you have provided in Column B of the /REPORT page. All or some of these named data values may be used in the REPORT function, and they may be specified in any order.

# The READ Function

The form of the READ function is:

*te* *cr* *v* *v*

**READ(textfilename,cellrange,direction,format)**

where

*te* = text in quotes or a text equation
*cr* = a specified cell range
*v* = a simple value, not an equation

*Example 1:*

```
read("myfile",bb2:dm50,c,cdf)
```

In Example 1, data from a file named MYFILE will be read into the range of cells B2 to M50 on pages B to D, in column-wise order (C). The data in MYFILE is expected to be *comma-delimited format* (CDF). An example of this common data exchange format is:

```
"expenses",12367.56,45.67
"income",10456.15,123.98
"misc.",568.34,0
```

Items that are to be read as text are surrounded by quotes. Items that are to be read as numbers are absent of the quotes. Note that commas cannot be used in numeric entries; commas take on the role of entry delimiters.

After the READ function has executed, the spreadsheet will look like this:

| | a | b | c | d |
|---|---|---|---|---|
| 1 | 9.00 | | | |
| 2 | | expenses | 12,367.56 | 45.67 |
| 3 | | income | 10,456.15 | 123.98 |
| 4 | | misc. | 568.34 | 0.00 |

In this instance, the file's data records are read into the spreadsheet column-wise. Each record's subentry, as defined by a separating comma, is entered row-wise, one cell to the right. If the specified direction of entry were instead row-wise, these subsequent entries would be entered into the spread-

sheet down the columns. Text is left-justified by default. Numbers are right-justified. The format for the numeric entries is that which is currently defined by the global format setting.

If the READ function were placed in cell A1, as shown, the value returned is the number of cells for which data was entered. If the specified area was too small to receive all of the data in MYFILE, an error will appear in cell A1 stating that fact, saying that some data was lost.

*Example 2:*

```
read(c13+".pub.aics",mc5:rc50,c,rtx)
```

In this second example, the file name will be taken off of the spreadsheet. The group and account information is concatenated to file name. The concatenation of text is a normal function of a text equation. Referencing a cell on the spreadsheet for the file name allows the file to be accessed to be dynamically changed, either by an operator or by a macro program, or possibly even by another READ function.

The data in this second example will also be read in columnar order. But this time, the data is expected to be solely text. Thus the format specified is *restricted text* (RTX). The restriction placed on such text is that only printable ASCII characters will be read into the spreadsheet. Any ASCII control characters ($0_{10}$ - $31_{10}$, $127_{10}$) which might reside in the file's data will be stripped from the incoming text stream.

In the restricted text format, all printable characters will be entered. Text does not need to be surrounded by quotes. Indeed, quotes and commas are treated just as all other text characters are.

Because the data in the file is text-only, each record (line of data) will take up no more than one cell (although the text's length may be longer than the cell's width and therefore appear to be written across multiple cell widths on the spreadsheet). Thus, the cell range specified in this second example has the data being written only in Column C, rows 5 to 50, on pages M to R.

The READ, CALL, and REPORT functions are constructed as standard numeric functions. They recalculate on the issuance of a single "!".

# The CALL Function

The form of the CALL function is:

*te* *te* *cr* *v* *v*

**CALL(routinename,controltext,cellrange,direction,format)**

where

*te* = text in quotes or a text equation
*cr* = a specified cell range
*v* = a simple value, not an equation

*Examples:*

```
call("myprog","control text",ac5:mg50,c,cdf)
     call(bg13,bg14+bg15,ac5:mg50,c,cdf)
```

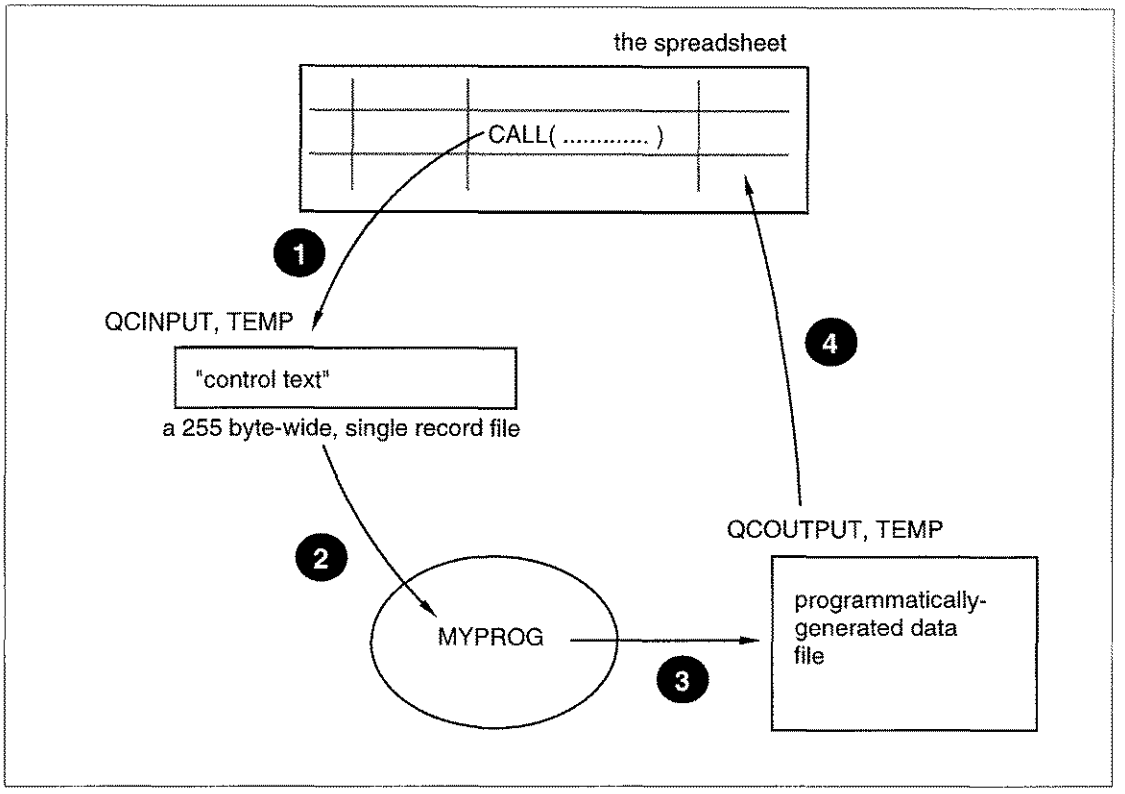
The CALL function executes either an external program or an SL-resident subroutine (XL's are not currently supported) and then reads the user-processed data back into the spreadsheet. The most reliable way to pass information between programs on the HP3000 is in files. Two file names are reserved for use by the CALL function: QCINPUT and QCOUTPUT.

*The execution sequence is as follows:*

*Step 1.* The CALL function creates the file QCINPUT as a temporary flat ASCII file, 255 bytes wide. The control text in the CALL function is then written into QCINPUT (see diagram, opposite page).

*Step 2.* The file directory on the HP3000 is then searched for an executable file named MYPROG. If found, MYPROG is activated (run). If not, the group's segmented library (SL) is searched. If MYPROG is not found there, the pub and system SL's are searched. If found, QueryCalc is suspended and MYPROG is activated. If MYPROG cannot be found, the CALL function fails with an announced error.

*Step 3.* Once QueryCalc has suspended, the user routine MYPROG performs whatever function it has been programmed to do. MYPROG is expected to write its results in the file QCOUTPUT. The user is responsible for the proper creation of QCOUTPUT as a flat ASCII file. If multiple us-




*The program sequence for the CALL function.*

ers are likely to be in the same group at the same time using the CALL function, the file QCOUTPUT must be created as a temporary file so that it will be identified with only the calling user's current file domain.

It is generally necessary to pass information to a user routine. Although the controlling text must be written all in one record, the use of delimiters such as comma (",") or back-slash (" \ ") allow you the capacity to provide multiple input parameters. MYPROG will have to be, of course, programmed to parse this inputted text into its separate components. You may occasionally find the normal QueryCalc cell equation limitation of 187 characters to be too small. If so, reference two or more cells as shown in the second example. Up to 255 characters may be transfered in this manner.

*Step 4.* At the termination of MYPROG, control returns to QueryCalc. The newly-created file QCOUTPUT is then read into the spreadsheet within the cell range specified. The reading of QCOUTPUT from this point on is identical to the READ function (see pages 15-2, 15-3).

MYPROG does not necessarily need to be a small program. It may perform elaborate power spectra statistical analyses or be a gateway into an IBM mainframe, downloading current information into the HP3000's databases.

# The REPORT Function

The form of the REPORT function is:

*v* *te* *cr* *ov*

**REPORT(reportpage,detailitemlist,cellrange,numcolumns)**

where

*te* = text in quotes or a text equation
*cr* = a specified cell range
*v* = a simple value, not an equation
*ov* = an optional value

*Example:*

```
report(A,"ssn,name,gross",mb2:te30,1)
```

The REPORT function reads data from the intermediate flat file created by a /REPORT page into a specified cell range on the spreadsheet. In the example shown, the /REPORT page is to be found on Page A. Among the data values that have been defined on the /REPORT page are *name*, *ssn*, and *gross*. When Page A was recalculated, these values—among possibly many others—were written into Page A's intermediate flat file, QCRPLSTA (see pages 13-4, 13-5). The values on this flat file are the values that will be read back onto the spreadsheet.

```
 A |_____________equation (a)_____________|____name (b)_____|_sort (c)__|_width____
 1|_____________________________find when zip<>0_____________________________________
 2|                  (Aa2)                 ssn                                  10
 3|                  (Aa3)                 name                  1a             15
 4|                  (Aa4)                 gross                                15
 5|

Equations for Page A
____________________

   (Aa1): @Using qcdemo.employees, find when zip<>0
   (Aa2): @rereading, val of socsecnum
   (Aa3): @rereading, val of lname
   (Aa4): @Using qcdemo.payrecord, sum of gross when
          socsecnum is [$a2]
```

*The equations for the Page A /REPORT*

| | a | b | c | d | e |
|---|---|---|---|---|---|
| 1 | | | | | |
| 2 | | ssn | name | gross | |
| 3 | | | | | |
| 4 | | 450134894 | ALCALA | 4,864.89 | |
| 5 | | 573277699 | AMBRIZ | 231.21 | |
| 6 | | 552458722 | ANDRADE | 1,716.75 | |
| 7 | | 526474725 | ARRATIA | 730.00 | |
| 8 | | 585079720 | AVALOS | 17,025.87 | |
| 9 | | 525066916 | BALDERRAMA | 915.00 | |
| 10 | | 585295348 | BANEGAS | 309.68 | |
| 11 | | 585662308 | BARRAGAN | 5,299.51 | |
| 12 | | 466986450 | BARRAZA | 512.00 | |
| 13 | | 051605462 | BAST | 294.00 | |
| 14 | | 096625991 | BEASLEY | 773.50 | |
| 15 | | 466087356 | BELTRAN | 769.04 | |
| 16 | | 527320816 | BIGGINS | 83,950.00 | |
| 17 | | 585884806 | BIGGINS | 25,135.49 | |
| 18 | | 585801899 | BLACK | 19,830.43 | |
| 19 | | 525068122 | BLACK | 380.00 | |
| 20 | | 361322407 | BLASING | 876.16 | |
| 21 | | 525963550 | BROWN | 42,700.00 | |
| 22 | | 450398360 | BURCIAGA | 2,841.00 | |
| 23 | | 457984236 | BURCIAGA | 4,000.50 | |
| 24 | | 465342237 | BURCIAGA | 3,916.00 | |
| 25 | | 526511186 | CABRALES | 1,536.00 | |
| 26 | | 585274817 | CALDERON | 1,401.75 | |
| 27 | | 449805738 | CALDERON, JR. | 2,432.50 | |
| 28 | | 585302203 | CARPENTER | 4,712.85 | |
| 29 | | 585078222 | CARSON | 42,700.00 | |
| 30 | | 585380764 | CARSON | 2,450.00 | |

*The resulting list on Page M*

When the data is read from the QCRPLSTA flat file, it is read into the cell range specified. In this instance, the data is read into the cell range B2 to D30, on pages M to T.

The first two header rows are automatically synthesized by the REPORT function. The selected data value names which appear in Column B of the /REPORT are used as column headings. Text item names are left justified; numeric item names are right justified. The widths of the columns are based on the width values specified in Column D of the /REPORT page. If no width values are specified, default column width values of 36 characters are used for text items, 12 characters for numeric items. A second row of header is filled with repeated hyphens in each affected column.

Once the two header rows have been built, data is read in the remaining cell

range. When page M's range fills with data, the process continues on to the next page.

An optional parameter in the REPORT function is the number of major coumns the data is to be written in. If the parameter is left unspecified, one major column is assumed. This is the manner in which the example shown was executed. If more than one major column is desired, the REPORT function will first write the list of data down the lefthand edge of the cell range, as shown, and then step over an equal number of columns to the right, skipping a "gutter" column. The width of the gutter column will always be set to 6 characters. The widths of the new columns past the gutter column will be set identically to the first set. This columnar process can be repeated as many times as will fit in the 26 columns of a QueryCalc spreadsheet page, *but note that the cell range specified specifed as a parameter in the* REPORT *function must account for a gutter column for each major column (even for the rightmost column).*


Although in this particular example data is read directly from the database without modification (that is, the concatenation of text, downshifting text, or numeric items multiplied together), such calculations are easily performed on the /REPORT page, and indeed is one of the strong points of the manner in which the /REPORT page is constructed.

When data is read from the intermediate flat file, numeric items retain the format structure that they were given on the /REPORT page. If this format is undesirable, or the automatically synthesized headers are unwanted, they may be changed or erased using standard QueryCalc commands. It is expected that data read into the spreadsheet from a /REPORT page will be further manipulated by hand.

One obvious use for REPORT-created listings is as a check reconciliation register, where all checks written in the previous month are listed on the spreadsheet, in sorted order. Cells to the right of checknumber, amount, and payee could be used to manually identify which checks had been cashed and which are still outstanding, carrying the actual and anticipated bank balances forward. Another such use might be *pro forma* ("what if...") projections of future project expenses based on expenses-to-date. A number of uses for half-manually/half-automatically maintained data can be imagined.

# USER EXERCISES

## The REPORT Function

1. Sign onto the practice account by typing **HELLO USER.AICS** at the colon prompt. Supply any necessary passwords. Passwords, if they are present, were put there by your system manager and he or she will know them if you do not.

2. Type **QC** to run QueryCalc. If this does not work, type **RUN QC.QCPROGS**. Press **RETURN** to bypass the instructions screen. You are now in QueryCalc proper.

3. Type **@opendb qcdemo**. Type **FRONT** (in all caps) in response to the password question. IMAGE databases are the only place in the HP3000 where passwords are case sensitive. You now have the database open.

## First Create a /REPORT Page

4. Now that you are on Page A, type **/REPORT** to convert the standard spreadsheet page into a detail list report page.

5. In cell A1, type **@using employees, find when zip<>0!** to select all of the employees who have a permanent address.

6. Move to cell A2 by pressing the **[f6]** function key. Type **@rereading, val of socsecnum!**. Press **[f6]** again to move to cell A3. Type **@rereading, val of lname!**. Press **[f6]** one final time to move cell A4. Type **@using payrecord, sum of gross when socsecnum is [$a2]!**. The first two @REREADING cells extract information from the EMPLOYEES dataset, the same set as the initial @FIND. The third cell sums all of the records for the person in question from the dataset PAYRECORD, using the person's social security number as the "linking" item.

## Name the Dataitems

7. Move the cursor to cell B2. Type **'ssn** as a dataitem label. Move to cell B3 and type **'name**. Move to cell B4 and type **'gross**. These labeled cells indicate which of the retrieved values are to be written into the report. In this case, because all three cells are labeled, all of the retrieved information will be written into the report.

8. Move to cell C3 and type **'1A**. This entry specifies the sort order. For this report, the retrieved entries will be sorted by last name.

# USER EXERCISES

9. Move the cursor to cell D2 and type **10**. Move the cursor to D3 and type **15**. Move the cursor to D4 and type **15** again. These numeric values set in the desired column widths for the labeled dataitems.

**The REPORT Function**

10. Now jump to Page B by either pressing the **[f4]** key or typing **/J B**. In cell A1, type **REPORT(A,"ssn,name,gross",mb2:pr30,3)**. Four hyphens ("----") will appear in the cell indicating the cell function has not yet been calculated.

11. Type **!!** to recalculate the entire report. Doing this will first cause the report specified in Page A to be calculated, sorted and written into a temporary flat file. Once Page A has recalculated, the REPORT function in Page B will read the data written in /REPORT Page A's temporary file. This data will then be placed it into the specified locations on the spreadsheet.

**Examine the Results**

12. Jump to Page M by typing **/J M** to examine the results. Notice that the column headers are the names we assigned in Column B of the /REPORT page. Notice also that the column widths are the widths assigned in Column D. We specified that three columns be written into the spreadsheet. If you move the cursor to the right you will see that three columns have been written. Now jump to Page N. You'll find that the data that could not fit in the rectangle specified in the REPORT function (B2:R30) is automatically continued into the next page.

**Comparing the REPORT & PRINT Functions**

13. To emphasize the difference—and the similarities—between the REPORT function and a standard report print, type **/PRINT A L**. The same data will now be "printed" onto your terminal's screen, as defined by an automatically generated default layout. The difference between the two functions is simple: /PRINT prints the report to paper. REPORT reads the same data back into the spreadsheet. The format of the printed data can be changed by typing **/LAYOUT** and modifying the format file (Please see Chapter 13). If the "L" in the command above were changed to an "S", the list report would be printed to your system printer.

# Concepts Introduced in Chapter 15

| | |
|---|---|
| BULK DATA READ | A process where data is read into the spreadsheet from an external file or database. Three numeric functions, READ, CALL, & REPORT are defined for separate data sources. The process is often called "downloading". |
| COMMA-DELIMITED | A data exchange format where commas separate subentries within a data record. Text entries are surrounded by quotes. Numeric entries are absent of quotes. |
| RESTRICTED TEXT | A second data exchange format for text-only entries. ASCII control characters are stripped out of the incoming text before being entered into spreadsheet cells. |
| ENTRY DIRECTION | The direction of data entry for the imported bulk data. The choices are either row-wise (R) or column-wise (C). |