HP Eloquence SORT Manual
A.06.00

HP Part No. B1368-90016
Printed in Federal Republic of Germany
October 1997

Fourth Edition

Notice

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Printing History

New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The dates on the title page change only when a new edition or a new update is published. No information is incorporated into a reprinting unless it appears as a prior update; the edition does not change when an update is incorporated.

The software code printed alongside the date indicates the version level of the software product at the time the manual or update was issued. Many product updates and fixes do not require manual changes and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one to one correspondence between product updates and manual updates.

First Edition	July 1991	A.03.00
Second Edition	February 1992	A.03.10
Third Edition	January 1997	A.06.00
Fourth Edition	October 1997	A.06.00

Table of Contents

Contents

1

Introduction

Overview of chapters

Chapter 1 presents a brief overview of SORT terms and concepts. Chapter 2 describes the syntax of the various statements and functions. Chapter 3 lists sample programs using SORT. Chapter 4 covers optimization techniques.

This manual is intended for the programmer who is familiar with both the Eloquence Language Programming Manual and the DBMS Programming Manual.

What is SORT ?

SORT is a collection of HP Eloquence statements and functions to facilitate the retrieval of information from an HP Eloquence database. There are statements available which allow you to access data in sorted order, and to select subsets of the total information available.

In addition, SORT enables you to set up simulationed structures more complex than the two-level networking supported by HP Eloquence databases. SORT enables the program to access a database in a hierarchical fashion. Simple data sets can also be handled, as can certain non-hierarchical structures.

Specifying database Structure

Before you begin any actual database access via SORT, you have to specify the structure of that portion of the database you want to use. You specify this structure as a list of set names. If you wish you can separate them using information concerning their inter-relationship. This list is called the *thread*. The thread specification describes the hierarchical (or other) structure on which SORT statements operate.

SORT operations are used to extract information according to the thread specification. This information is in the form of record pointers which the program uses in direct-mode DBGETs to obtain the actual information from the database. The thread may contain from one to ten sets, depending on the particular application.

The diagram below shows the example Sales Analysis database. Among reports which you could obtain are:

1 A list of all orders.

2 A list of products plus the orders placed for that product.

3 A list of products and orders, as above, but including the options contained with each order.

To produce report 1, only the CUSTOMER data set is involved. The thread list for such a structure would consist only of CUSTOMER. Report 2 involves the data sets PRODUCT and CUSTOMERS, while 3 involves the sets PRODUCT, CUSTOMER and OPTION.

Details of how to generate these are contained later in this chapter, along with the program to generate report 1. In chapter 3 you will find complete sample programs for reports 2 and 3.

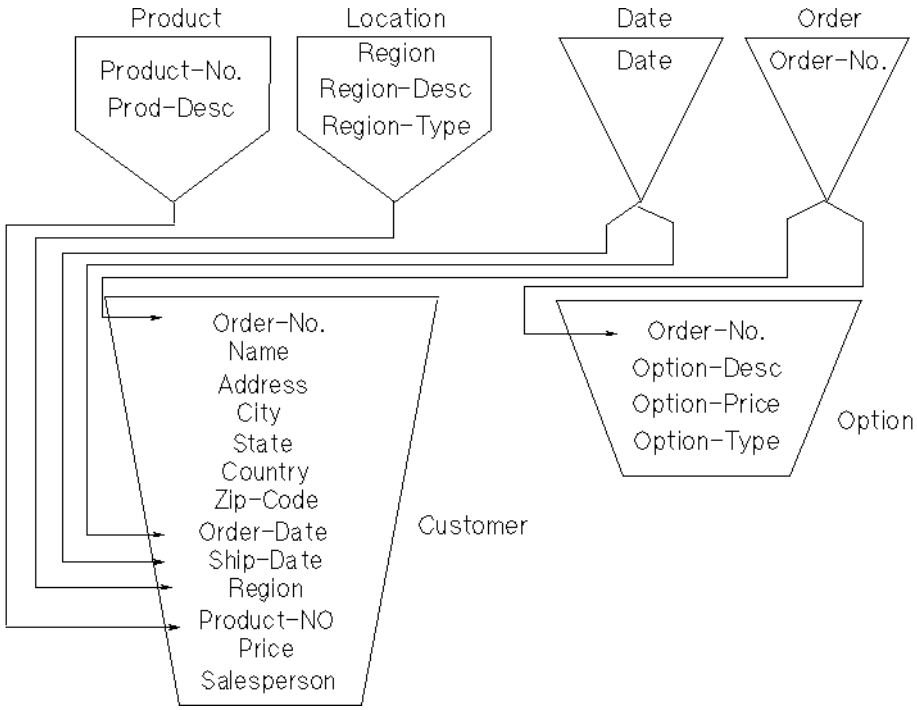

Figure 1 **Sales Analysis Data Base**

The Workfile

With SORT you create a specific access sequence to the database. You do not actually change the sequence of the data in the data base itself. What you do is to build a series of *pointers* to the various records of each set in the thread. These pointers are stored in a special file called the *workfile* and can be used with direct-mode DBGETs to extract information from the database. If, for example, you want to produce a report listing all orders plus the company placing each order (see the next report), you could use a program like the one shown next. This program opens the Sales Analysis database, sets up a workfile, sorts the data by order number and prints the results. (Note that the order shown in the sample run below is, in fact, correct since the items being stored are strings, *not* numerics).

```
10    DIM Buf$[170],B$[5],Order_no$[10],Names$[30]
20    INTEGER  S(9)                 ! Ten-element status array.
30    B$="SAD"
40    DBOPEN (B$,"SECRET",3,S(*))   !   Open the database.
50    DBASE IS B$
60    IN DATA SET "CUSTOMERS" USE Order_no$,Name$
70    !  Now set up a workfile with CUSTOMER as the thread.
         .
         .
         .
110   !  Sort the orders by order number.
         .
         .
140   PRINT " ORDER NUMBER";TAB(30);"CUSTOMER NAME";LIN(2)
150   FOR I=1 TO Entry_count
160     ! Read the record pointer into Rec_no.
         .
180     DBGET (B$,"CUSTOMER",4,S(*),"@",Buf$,Rec_no)
190     PRINT Order_no$;TAB(30);Name$
200   NEXT I
210   DBCLOSE (B$," ",1,S(*))
         .
230   END


RUN
ORDER NUMBER                        CUSTOMER NAME

10                                  ABC Company
100                                 Colorado Feed and Grain
12.6                                Bruce's Bar & Grill
17.2                                Timmy's Pet Store
20                                  ABC Company
999                                 Internal Revenue Service
```

In many respects, the workfile is just like a regular data file. It must be CREATEd and ASSIGNed just like a DATA file. Only its use in the program distinguishes it as a workfile. After the program defines a file as a workfile, it remains that way

until either it is de-ASSIGNed or the program stops. Since the workfile is so similar to a normal data file, most of the standard file operations work on it. Record pointers are read from the workfile with the READ # statement, and pointers may be added to it using PRINT #. The next figure shows how the contents of the workfile are related to the base entries used in the previous example.

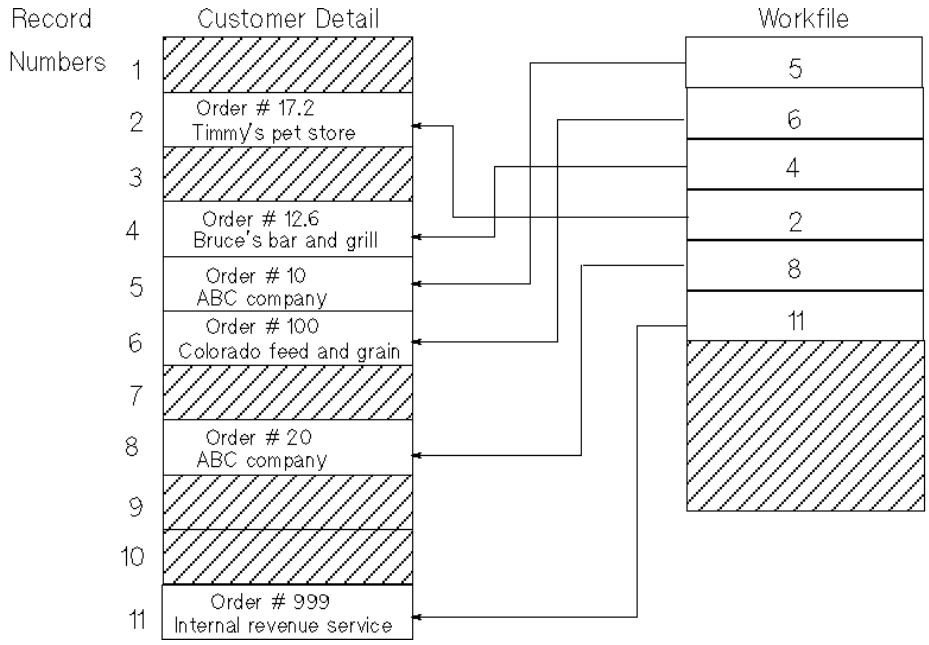

Figure 2 **Data Base/Workfile Relationship**

Putting Data into Sorted Order

The SORT BY statement allows you to specify a sort using up to ten data items from any data set in the thread. If a sequence of two elements cannot be determined on the basis of the first field, the second, the third, and so on, will be compared until a sequence can be found. If no sequence is found, the pointers into the database are compared in order to determine sequence. Additionally, sort direction is specifiable on each sort field on an individual basis. Any field may be suffixed by the keyword DES to cause the sort to be in descending order rather than ascending.

Here is the same program shown earlier, but with some additional statements filled in:

In this example, lines 80 thru 100 are used to create a file, ASSIGN it to a file position and convert it into a workfile. (Note that the file is still of type DATA.) Line 120 produces pointers so the data can be accessed in sorted order. Line 170 reads the pointer into an HP Eloquence variable so it can be used in the direct mode DBGET in line 180.

One additional function has been introduced in this example, the WFLEN function used in line 130. This function returns the number of pointers in the workfile. It has as an argument, the file number of the workfile, since more than one workfile may be in use at a given time. Notice that the program creates and purges the workfile each time the program is run. If disk space is available, program execution time can be decreased by deleting lines 80 and 220, which allows the file to remain on the disk.

```
 10    DIM Buf$[170],B$[5],Order_no$[10],Name$[30]
 20    INTEGER S(9)                  !  Ten-element status array.
 30    B$="  SAD"
 40    DBOPEN (B$,"MANAGER",3,S(*))  !  Open the database.
 50    DBASE IS B$
 60    IN DATA SET "CUSTOMER" USE Order_no$,Name$
 70    !  Now set up a workfile with CUSTOMER as the thread.
 80    FCREATE "XYZ" ,0
 90    ASSIGN "XYZ" TO #1
 100   WORKFILE IS #1;THREAD IS "CUSTOMER"
 110   !  Sort the orders by order number.
 120   SORT BY Order_no$
 130  Entry_count=WFLEN(1)   !WFLEN returns no of pointers in file.
 140   PRINT " ORDER NUMBER";TAB(30);"CUSTOMER NAME";LIN(2)
 150   FOR I=1 TO Entry_count
 160   !  Read the record pointer into Rec_no.
 170   READ #1;Rec_no
 180   DBGET (B$,"CUSTOMER",4,S(*),"@",Buf$,Rec_no)
 190   PRINT Order_no$:TAB(30);Name$
 200   NEXT I
```

```
210   DBCLOSE (B$," ",1,S(*))
220   PURGE "XYZ"
230   END
```

Selecting Data

Frequently only a small portion of the total available space is of interest for processing purposes. SORT provides the FIND statement to select only those entries in the hierarchy which are relevant. This selection can involve data available at any level of the hierarchy and may use an arbitarily-complex selection criterion involving any function available in an HP Eloquence expression.

When a FIND is executed, pointers to some subset of the records in the hierarchy are put in the workfile. Only the pointers of records which meet the selection criteria are put in the workfile. If there are already pointers in the workfile from executing previous FINDs (or SORTs), the subset described by these pointers is used in successive FINDs and SORTs, rather than all the information present in the database.

Suppose, in the above example, you wanted to list only the orders for ABC Company. You could do this by inserting a FIND statement somewhere between line 100 and line 130 to select only those customers. Thus you could produce a report for just ABC Company by adding:

```
115   FIND TRIM$(Name$)="ABC Company"
```

This line could also have gone after the SORT BY in line 120, since executing a FIND does not change the sequence produced by the last SORT BY. Note the use of TRIM$. This is necessary because FIND works like a direct-mode DBGET. The unpacking procedure performed by IN DATA SET will leave any trailing blanks on the string.

Suppose, now, that you want to put an additional restriction on the set of orders in the report. The report should contain only orders from ABC Company and those with a “2” somewhere in the order number. You can do this in either of two ways. You can add another FIND statement specifying the additional restriction between lines 100 and 130. Or you can change line 115. The first method might produce a line like:

```
125   FIND POS(Order_no$,"2")<>0
```

NOTE: Now one of the FINDs is before the SORT BY and one is after it. Both could also appear before or both after the SORT BY.

The second method is a more efficient way. The fewer FIND statements executed the better, since then each data entry need be examined only once. (This is the usual case. More details on the best way to optimize FINDs are presented in Chapter 4.) This method might have produced a replacement for line 115 such as:

```
115   FIND (TRIM$(Name$)="ABC Company") AND (POS(Order_no$,"2")<>0)
```

Specifying Complex database Structures

As indicated earlier, it is sometimes useful to sort or find records spread over several data sets when those data sets logically represent a hierarchy. The thread parameter on the workfile statement allows you to do this. The thread is basically a list of the sets in the order they occur in the hierarchy.

The following figure shows one master with three detail sets linked to it.

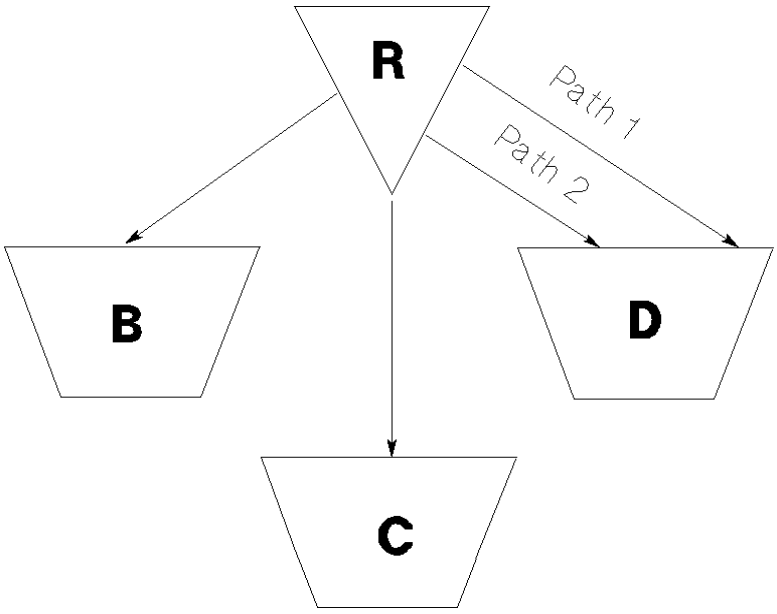

Figure 3 **Multiple Two-Level IMAGE Structure**

Threads defined in the above IMAGE Structure

{A} or {B} or {C} or {D}

{A,B} or {B,A} {A,C} or {C,A}

{B,A,C} or {C,A,B}

Notice that detail data set D has two data paths to the same master. In this case, linking set A to set D is ambiguous. To resolve this ambiguity, it is necessary to specify which path is involved. Adding this capability to the thread specification allows the description of the following additional threads:

Additional Threads

{A (via path 1) D (via path 2) A}
{A (via path 2) D (via path 1) A}
{D (via path 1) A (via path 2) D}
{D (via path 2) A (via path 1) D}
{C,A (via path 1) D (via path 2) A,B}
{B,A (via path 2) D (via path 1) A,C}
etc.

Remember that although all these threads can be defined, they may not make any sense! It is the programmer's responsibility to determine the sense of a thread.

For another example, see the three reports on page 1-2. Generating report 2 involves using two sets. The thread that describes this hierarchy is specified as a list of PRODUCT and CUSTOMER. Report 3 involves three sets (PRODUCT, CUSTOMER and OPTION). The structures involved in all these reports are hierarchical in nature. In report 2, for example, the PRODUCT data set is higher in the hierarchy than CUSTOMER. Report 3 is an example of a three-level hierarchy. The next figure shows how the hierarchy for report 3 is organized.

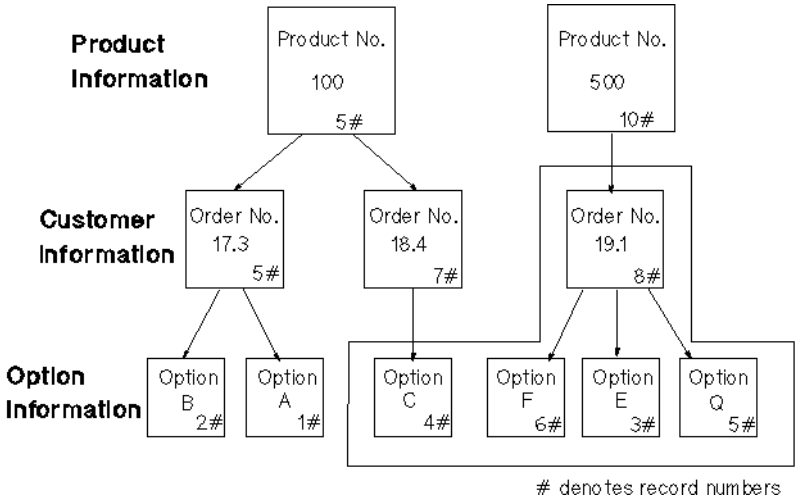

Figure 4 **Sample Three-level Hierarchy**

Unlike report 2, where there is a direct connection between PRODUCT and CUSTOMER, there is no connection between CUSTOMER and OPTION. This is why the ORDER master data set exists. The thread necessary for accessing this three-level hierarchy consists of four sets which are specified in the order PRODUCT, CUSTOMER, ORDER and OPTION. See the next figure.

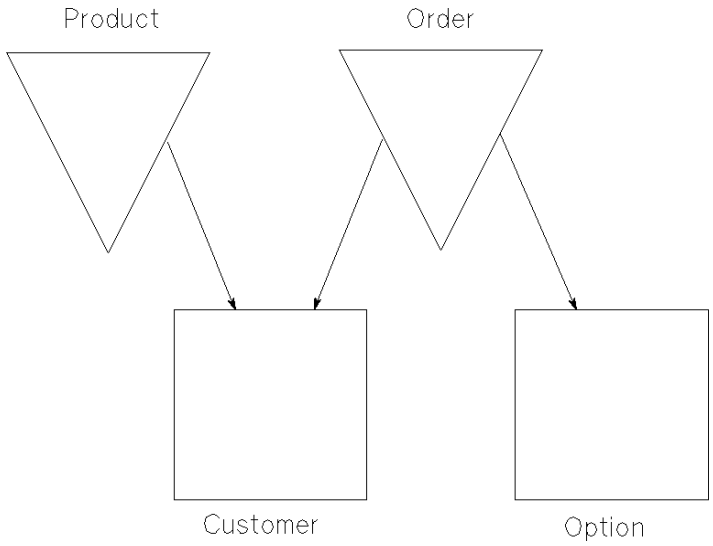

Figure 5 **Simulation of a Three-level Hierarchy**

A sample output for report 3 is shown next. Notice that information is obtained from the product data set (product number and description), as well as from each of the other sets. Graphically, this information is organized as shown on page 1-8. The numbers in the corner of the boxes correspond to the records where the information is stored in the database. Entries for the ORDER detail are not shown, since the ORDER set contains no information relevant to producing the report.

```
                              OUTSTANDING ORDERS LIST

PRODUCT NO.       ORDER NO.    CUSTOMER NAME              OPTIONS        PRICE

100(STD BICYCLE)    17,3       XYZ Company                A              10,25
                                                          B              20,31

                                                                         30,56

                    18,4       XYZ Company                C              30,97

                                                                         30,97

                                                 TOTAL 100 ORDERS:               61,53

500(5-SPEED)        19,1       ABC Company                E             132,05
                                                          F             100,10
                                                          Q               1,23

                                                                        224,38

                                                 TOTAL 500 ORDERS:       224,38

                                                 TOTAL ORDERS:           285,91
```

To produce report 3, it is necessary to extract this information from the database (record numbers from the figure titled “Sample Three-level Hierarchy” .)

Table 1 **Information to extract to get report 3**

Set Name	Record to Read	Action to Take
Product	5	Print header product.
Customer	5	Print header for order.

Table 1 **Information to extract to get report 3**

Set Name	Record to Read	Action to Take
Option	1	Print first option.
Option	2	Print last option and total.
Customer	7	Print header for new order.
Option	4	Print option and totals.
Product	10	Print header for new product.
Customer	8	Print header for order.
Option	3	Print first option.
Option	6	Print second option.
Option	5	Print last option and totals.

The numbers stored in the workfile, however, always contain one record from each set. Thus, the first record will contain the three order number pointers and the pointer to the ORDER set.

The subsequent record is the same except that the pointer for the option set is changed to 2. The next figure shows the pointers as they are stored in the workfile.

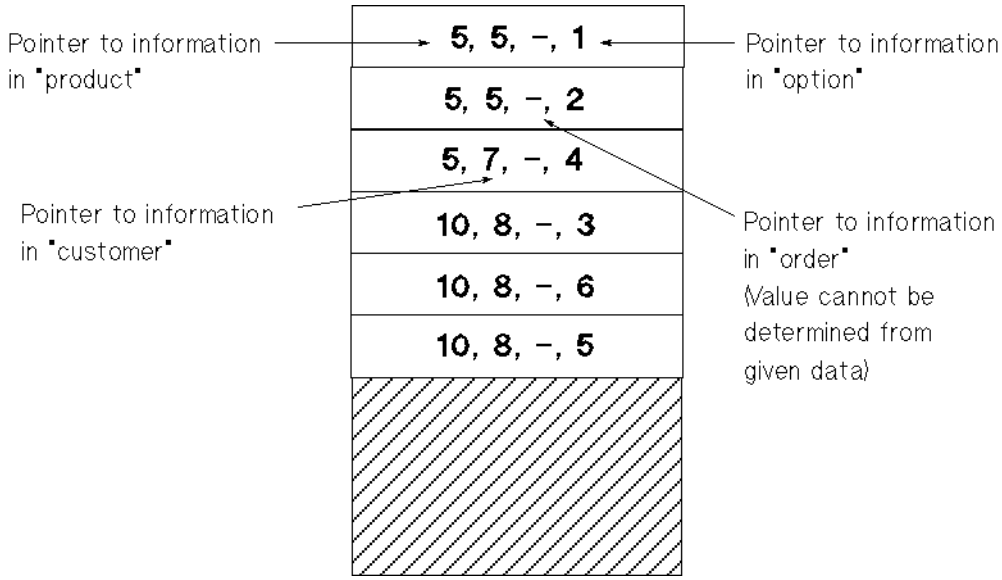

Figure 6 **Contents of Workfile after Sorting**

Note that one pointer for each set is always stored. If a record at one level of the hierarchy has no records associated with it at the next lower level, there is no way to store a record of pointers in the workfile relevant to that record. In particular, if the records surrounded by a box in the figure titled, “Sample Three-level Hierarchy” are deleted, product 500 has no order associated with it and order 18,4 has no associated options. The workfile would then have only two records corresponding to the bracketed records in the next figure. Further, if the options on order number 17,3 were deleted, FIND or SORT would return an empty workfile.

The program to produce the outstanding order list is fairly complex, as shown in Chapter 3. However, the skeleton for the program is shown next. This skeleton reads four pointers from the workfile even though the third pointer (to the automatic master set ORDER) is not used. Also, note that this skeleton repeatedly reads records from the PRODUCT and CUSTOMER data set even though it may be reading the same record as on the previous pass through the loop. For clarity’s sake, the code to optimize out the extra reads is not shown.

```
ASSIGN "XYZ" TO #1
WORKFILE IS #1;THREAD IS "PRODUCT","CUSTOMER","ORDER","OPTION"
    .
    .
    .
IN DATA SET "CUSTOMER" USE ALL
IN DATA SET "OPTION" USE SKP 1,Option_desc$,PO
```

```
    .
    .
    .
SORT BY Product_no, Order_no$,Option_desc$
    .
    .
    .
FOR L=1 TO WFLEN(1)
READ #1;R1,R2,R3,R4
DBGET (Base$,"PRODUCT",4,S(*),"@",Buf$,R1)
DBGET (Base$,"CUSTOMER",4,S(*),"@",Buf$,R2)
DBGET (Base$,"OPTION",4,S(*),"@",Buf$,R4)
    .
    .
    .
NEXT L
```

2

Sort Statements and Functions

Introduction

This chapter describes the syntax needed to use SORT software. The statements and functions provided with SORT are:

WORKFILE IS # A statement specifying the hierarchical structure (thread) of the data sets to be sorted, the work space for sorting, and the workfile itself.

SORT BY A statement specifying the order in which data is to be sorted.

FIND A statement used to select a subset of record pointers from the data base or the current workfile.

QFIND A statement used to select a subset of record pointers from the data base.

WFLEN A function returning the number of logical records in the workfile.

Two IMAGE statements, DBASE IS and IN DATA SET, are used to define the data base and data sets before unpacking data entries with SORT.

In addition, many HP Eloquence file storage operations (PRINT #, READ #, REC, etc.) are used in conjunction with SORT workfiles. Because of the workfile structure, these operations may work differently with SORT than as described in the HP Eloquence Manual. These differences are covered near the end of the chapter.

Conventions

The following conventions are used:

- **Bold type** is used when a new term is introduced.
- **`Computer font`** indicates text to be input exactly as shown or text that is output from the system.
- *Italic type* is used for emphasis and titles of publications. It is also used to indicate parameters that are user defined.
- KEYCAP represents a key on the keyboard.
- Shading represents the softkeys displayed on the computer scren.
- …indicates that the previous variable can be repeated.
- [] indicates that information inside the brackets is optional. If there are brackets within brackets, the parameter within the inner bracket may only be specified if the parameter in the outer bracket is specified. Parameters may also be stacked in brackets. For example A or B or neither may be selected when the following is shown:

$$\left[\begin{matrix} A \\ B \end{matrix}\right]$$

- {} indicates that one of the choices stacked within the braces must be selected from those stacked within braces. For example A or B or C must be selected when the following is shown:

$$\left\{\begin{matrix} A \\ B \\ C \end{matrix}\right\}$$

The WORKFILE IS # Statement

The WORKFILE IS # statement describes the hierarchical structure on which FIND, QFIND and SORT will operate, where the scratch area is for SORT, and where the results of executing a FIND, QFIND or SORT are stored.

WORKFILE IS # *file number* [**;THREAD IS**

[*set id* [LINKlink / :path id] , ...] *set id*] **thread list**

(up to 10 sets allowed)

The parameters are:

file number A numeric expression having an integer value from 1 though 10, and used to identify a file previously defined by an ASSIGN statement.

set id A numeric or string expression used to identify a data set. If numeric, this parameter references a data set number for the current data base (specified in the last DBASE IS statement). If a string, this parameter references a data set name for the current data base.

path id A numeric expression having an integer value from 1 through 8. This expression selects which data path to use between the first data set specified (set id) and the next in the thread list. It is needed only when more than one path exists between two sets being linked in the thread. If only one path exists for the data set specified, it is not necessary to list the path id parameter.

link An HP Eloquence variable which is currently linked via an IN DATA SET statement to an item found in the detail data set to which it is attached. The variable must match in type and length the search item in the master data set which follows in the thread list. If the variable refers to a sub-item, it may only be the first sub-item.

NOTE: The path or link parameters cannot be specified on the last data set in the thread list, since these operations specify a relationship between the set to which it is attached and the next set listed in the thread.

Some examples of the WORKFILE IS # statement are:

```
WORKFILE IS#1; THREAD IS"CUSTOMER"
WORKFILE IS#X+3; THREAD IS "CUSTOMER":2, "DATE"
WORKFILE IS#8; THREAD IS "CUSTOMER":2, "ORDER"
```

Up to 10 data sets can be specified for any thread list. The number of sets in the list is referred to as the **thread length**. Each set must be related to the sets on either side of it (or one side if it is at the end of the thread) by a path in the data base (or a **synthetic path** using the LINK option). This defines the hierarchical structure, with the leftmost set in the thread list usually being the highest (usually the least commonly occuring) in the hierarchy. Successful execution of WORK-FILE IS # converts the file into a workfile. To convert a file to a workfile, the file must be ASSIGNed in exclusive mode. The file remains a workfile until either another file is assigned in its place (same file number) or it is de-assigned. Closing the data base to which the workfile pertains automatically de-assigns the workfile.

The workfile is used to store all pointers generated by FIND, QFIND and SORT BY operations. Initially, the workfile contains no pointers, so any attempt to access them (via READ #) will result in an error. The REC function returns 0 to indicate this null state. Pointers can be put in the workfile by executing SORT BY, FIND, QFIND, FIND ALL or PRINT #.

The workfile is composed of logical records whose lengths in bytes are 4 times the thread length. Thus, a 4-byte pointer is stored for each set in the thread in any given logical record. Pointers may range in the value from 1 to the capacity of the set to which they pertain. (The first pointer in the record corresponds to the first set in the thread, the second pointer corresponds to the second set, and so on.)

In the case where more than one path connects two adjacent sets in the thread, it is necessary to specify which path is to be used. This is done by suffixing the first of the sets with a ":" and following that with a path id. The path id for a particular path is determined by using the schema listing. To find the path with path id *n*, for example, scan the detail for the *n*th occurrence of the master set name. If the path id is not specified, 1 is assumed.

A method exists for defining data set relationships independent of the data base structure. This method is used to link a detail data set to a master data set in the thread list. This is done by using the LINK option, which specifies an item in the detail data set and is used to perform a calculated access into the specified master data set. This item must match the type and length of the search item in the master data set (which is then the set id following the LINK in the thread list).

All SORT BY, FIND and QFIND operations work with the current workfile. Executing another WORKFILE IS # deactivates the current workfile and defines a new one. All subsequent SORTs, QFINDS and FINDs then work on the new file. The information in the old workfile is still intact, however, and can be accessed via READ # and PRINT # statements.

Since it may by desirable to return to do additional FINDs and SORTs on the previous workfile, a method is provided for saving and reactivating a workfile. This is done by executing another WORKFILE IS # which does not include the thread list. This will deactivate (but not erase) the current workfile and allow you to activate an old workfile. Do not attempt to reactivate the workfile by respecifying the thread list, since this loses all information currently in the file by resetting WFLEN to 0.

Expressions are allowed in all WORKFILE IS # parameters. When invoking multiple-line function subprograms, however, these subprograms cannot execute SORT BY, FIND, WORKFILE IS #, IN DATA SET or DBASE IS statements.

The SORT BY Statement

The SORT BY statement generates pointers accessing data in a specified order.

SORT BY *variable name* [**DES**] [**,…,** *variable name* [**DES**]]

The parameter is:

variable name An HP Eloquence variable linked via the IN DATA SET statement to an item appearing in one of the data sets in the thread. Substrings are not allowed.

Sorting can occur on up to ten data items. If an order cannot be determined from the first data item, subsequent data items can be specified to determine the order. If no order can be found, the order for those records will be determined by their record pointer value(s) in the data set(s). The specified data is sorted in reverse order by specifying **DES**. Each data item listed can be sorted in either order.

Data items used for sorting can come from any data set belonging to the thread of the current workfile. When listing the data items in the SORT BY statement, you must place them in order of their significance to the sort, not in their original set order. If an item occurs in two data sets in the thread, the item will be assumed to come from the leftmost set.

Since SORT BY and FIND handle record pointers in the data base, and other users may be modifying the data base, care should be taken when using FIND and SORT BY while the data base is opened in mode 1.

There are a couple of miscellaneous items concerning SORT BY. The first is that executing a SORT BY resets the workfile pointer (as determined by REC) to 1. The second is that if SORT BY is reading the data base via pointers in the workfile (rather than accessing the data base directly) and records in the data base have been deleted since the FIND, SORT or PRINT # that put the pointers there, then any logical workfile record which contains a pointer to a deleted data set record will be deleted. This is true only when SORT BY accesses the set in which the deletion occurred. If there is no sort item needed from that set, SORT BY will not perform the read to determine if a deletion has occurred.

Some example sequences using SORT BY are:

```
SORT BY Order_no$

SORT BY Product_no$,Name$ DES
```

The FIND Statement

The FIND statement selects a subset of records from the data base thread or the current workfile if the workfile is non-empty.

FIND { ALL / *condition* }

The parameter is:

condition Any numeric expression used to test variables (or any attribute) for certain conditions. If these conditions are met, the expression has a non-zero (true) result and the record pointers are stored in the workfile. Otherwise, the result is 0 and the record pointers are not stored.

If the workfile has not been used with any previous FIND or SORT BY operation, FIND examines the data base associated with the current workfile. The condition parameter is evaluated to determine whether the group of data entries just read should have their pointers put in the workfile. If the condition is met, the pointers are stored and the next group of entries are processed. Otherwise, the pointers are not stored and processing continued. Note that FIND must actually read each record and trigger the IN DATA SET for each set in the thread to establish the variable values it needs to evaluate the condition expression.

If the workfile already contains pointers (indicated by REC greater than 0), only the data entries specified by the pointers in the workfile are checked by the condition parameter. Pointers to data entries that meet the condition criteria are retained in the workfile; all other pointers are deleted.

Since FIND handles record pointers in the data base, and since other users may be modifying the data base, care should be taken when using FIND while the data base is opened in mode 1.

Specifying FIND ALL is the same as FIND 1=1, and is useful to get all records in unsorted order. If a subsequent FIND or SORT BY is used, however, the FIND ALL is not needed and only wastes time. If a FIND, SORT BY, or PRINT # has previously been done, FIND ALL has no effect except to reset the record pointer to record 1.

There are two miscellaneous items concerning FIND. The first is that executing a FIND resets the workfile pointer (as determined by REC) to 1. The second is that if FIND is reading the data base via pointers in the workfile and deletions have occurred in sets involved in the FIND, then FIND will delete the logical workfile records containing pointers to empty data set records.

NOTE:

If the condition parameter does not use values from a particular set in the thread (via an IN DATA SET statement), execution time can be improved by deacticating the IN DATA SET statement using the FREE option.

Some examples sequences using FIND are:

```
FIND TRIM$(Order_no$)>"1000"
FIND (Vendor_no>250) AND (Invoice_no>10000)
FIND ALL
```

The QFIND statement

The QFIND statement selects a subset of records from the data base thread.

QFIND *item, relation, value* [;*expr*]

or:

QFIND *item*,"**IN**", *value1, value2* [;*expr*]

or:

QFIND *iitem*,"**MATCHES**",*regular expression*

The parameters are:

item A numeric expression specifying an (index) item number or string expression specifying (index) item name. The specified (index) item must be in the first dataset of the THREAD list.

relation A string expression specifying the test to be performed on the given index item.

">" or **"GT"** - greater than
">=" or **"GE"** - greater or equal
"=" or **"EQ"** - equal
"<=" or **"LE"** - less or equal
"<" or **"LT"** - less than
MATCHES - matches regular expression

The **"IN"** relation will check for a value range starting at value1 and including up to value2.

If *item* specifies a search item, *relation* must be **=** or **EQ**.

value Any string or numeric expression. See DBFIND for how to specify (index) item lookup values.

expr Optional expression evaluated for each group of records in the thread. If the expression evaluates non-zero, the records are transfered into a workfile. See FIND statement for details.

regular expression See next page.

Using a FIND statement, the first data set in THREAD will be read in sequential order. This may take a long time, depending on the number of entries in the dataset. QFIND allows quick access using either index items or search items.

QFIND will always add pointers to the workfile but not process pointers already in the workfile, which means it is possible to add the pointers of multiple subsets into a workfile using QFIND.

Specifying the conditional expression results in the same workfile as using a QFIND/FIND sequence but reduces overhead.

Here you will find some sample sequences using QFIND.

```
QFIND "ORDER-NO","=","1234" QFIND "TEST", ">=",15 QFIND
"ACCOUNT","IN",10000,12000;Account_type=15 QFIND "CUS-
MC","MATCHES","M[a-f]*"
```

Regular Expressions

Elements:

[	starting delimiter of character class expression
[	ending delimiter of character class expression
!	negation expression (only as 1st character of character class)
-	range expression (only inside a character class)
?	any character
*****	any string (including the empty string)
#	numeric character (same as [0-9])

The backslash character (\) loses its special meaning within the delimiters, except in the following combinations:

\b - becomes **backspace**

\t - becomes **tab**

\r - becomes **cr**

\n - becomes **lf**

\f - becomes **ff**

\s - becomes **space**

The above combinations conform to the HP-UX standard, and are extremely practical.

Evaluation

An evaluation is only possible with index items, and then only for leading string segments. Index items without leading string segments cannot be accessed.

A regular expression must exactly describe the contents of the leading string segments. There is no implicit "*" at the end (as in DBFIND 2/3). For example, the value "AAA "(trailing space) does not match the search expression "AAA".

Examples of regular expressions:

A[BCD]	Index value starts with A, followed by either a B, C or D.
BOB?*	Index value starts with BOB, followed by at least one character.

The WFLEN Function

The WFLEN function returns the number of logical record pointers contained in the specified workfile.

WFLEN(*file number*)

The parameter is:

file number A numeric expression specifying the file number of the workfile.

WFLEN returns a value from 0 through 2^{31}. If a FIND, QFIND or SORT BY has not been executed on the workfile, 0 is returned. 0 also indicates no entries in the workfile. -1 is returned when the contents of the workfiles are invalid (caused by pressing HALT ALL or CTRL Y or getting a disk error during a SORT BY or FIND statement). Executing WFLEN on a file other than a workfile causes an error.

The READ # and PRINT # Statements

The READ # and PRINT # statements operate on workfiles in much the same way as they operate on DATA files. Although their syntax is identical, certain restrictions apply when operating on workfiles.

The first restriction is that only an integral number of logical records can be read or written. If a partial logical record is read, an error is issued and the record pointer is left at word one of the incompletely read record. If a partial logical record is written, the incompletely written record is not changed; instead, the record pointer at the beginning of that record and an error is issued. Strings cannot be read or written on workfiles. Arrays can be written or read by using the array notation (i.e. A(*), or via MAT PRINT # and MAT READ #.

Note that a pointer value is a value between 1 and the capacity of the set to which it pertains.

If a non-integral value is PRINTed on a workfile, it is rounded to an integer. If the rounded value is less than 1 or greater than the set capacity, an error occurs.

The record pointer for READ # and PRINT # can be positioned at any record from 1 through WFLEN + 1. Attempting to position past record number WFLEN + 1 results in an end-of-file error (which is trappable by ON END #). When printing to records greater than WFLEN, the value of WFLEN is adjusted appropriately. However, actually trying to read values in records beyond WFLEN causes end-of-file error.

PRINTing an END on a workfile resets WFLEN to a value corresponding to the record where END was printed -1. This effectively erases all information from the record where END was printed to the end of the workfile.

Other Statements and Functions

Certain statements and functions behave differently when applied to workfiles. Briefly, these are:

TYP Returns 0 since the data stored in a workfile is not a standard data type. (Each pointer may have a value from 1 through 2^{31} and takes 4 bytes of storage.)

WRD Always returns 1. Since only complete logical records can be read from or written to a workfile, the word pointer of the file will always be the first word of the record.

REC	Works the same as DATA files. The value returned will be between 1 and WFLEN + 1. If no pointers are put into the workfile, however, 0 is returned.
SIZE	Returns file size in sectors for a positive argument, and thread length for a negative argument.
RESET #	Erases the workfile. Result is same as WORKFILE IS #; THREAD.

SORT Order of Execution

SORT statement execution is strongly interrelated to DBMS and HP Eloquence file storage operations (see the previous section). In order to execute a statement, all statements which have pointers into the statement to be executed must have previously been executed (e.g., to execute a FIND, a WORKFILE IS # and at least one IN DATA SET must have been previously executed).

3

Program Examples

This chapter shows several programs using SORT operations with the Sales Analysis Data base (SAD). The two programs introduced in Chapter 1 are described here: a program to list products along with their associated orders and a program which also lists the options for each order. Whenever possible, the line numbering for logically-equivalent statements remains the same for each program.

Order List Programs

Each of the order list programs produces a report as shown on page 3-2. This report lists the orders in the data base, broken down by product and sorted according to order number. The products themselves are listed in sorted order. Also, totals are maintained for all orders on each product as well as a total of all orders.

Example program 1 uses a two-set thread (see line 1320). This means that two pointers must be read in line 1480. The R1 pointer refers to a record in the PRODUCT set and the R2 pointer refers to a record in the CUSTOMER set.

Every time the product changes, the value of R1 also changes. S1 represents the value of R1 at the previous pass through the FOR loop. It is used to detect when it is necessary to print a trailer for the current product (consisting primarily of the total of the orders for the product) and a heading for the new product. Note however, that printing a trailer at the first pass through the loop is undesirable. A special test for S1=0 is made to stop this from occurring.

Note that the sort performed in line 1360 has Prod_no as its primary sort field. This variable comes from the PRODUCT data set (see line 1190). Because the schema item "PRODUCT-NO" is a search item, however, the value of the variable *Product_no$* from the CUSTOMER detail set could just as well have been used.

This program shows many poor programming practices which are corrected by example program 2:

- The status array is never tested at any point in the program. The data base may not have been opened; this will ultimately result in error 211 being issued in line 1180.
- As pointed out earlier, the PRODUCT data set need not be involved in the sort. As discussed in Chapter 4, having the PRODUCT data set in the thread greatly reduces efficiency of the SORT BY statement. The description field, however, must be accessed to get the description field for printing. (This is done by a calculated-access DBGET in line 1690 of example program 2.)
- After deleting PRODUCT from the thread there is only one pointer per record in the workfile (see line 1480). This points into the CUSTOMER set, so there is no way to wait for change in record number to indicate a change in product. Thus, the actual product numbers mut be compared. Note that the update of the old product number is accomplished by the IN DATA SET which is triggered when the DBGET in line 1690 is executed. This means that a line analogous to line 1710 in the first example is not needed.

Order List Report

```
                    OUTSTANDING ORDERS LIST

PRODUCT        ORDER NUMBER          CUSTOMER NAME              PRICE

50  (Tricycle)
               110                   Gissing, Malcom             45,00
                                                               =========

   TOTAL ORDERS FOR 50                                          45,00

100 (Standard Bicycle)
               101                   Noname, Joseph              77,50
               103                   Hernandes, Jose            109,75
               108                   Arauja, Luciano A.          80,00
                                                               =========
   TOTAL ORDERS FOR 100                                        267,25

300  (3-Speed Bicycle)
               104                   Houseman, Sean             133,00
                                                               =========
                                                               133,00
    TOTAL ORDERS FOR 300

500  (5-Speed Bicycle)
               100                   Smith, Thomas A.           175,50
               105                    Sono, Jomo A.             135,00
               109                    Bekker, Bart              125,00
                                                               =========
   TOTAL ORDERS FOR 500                                        435,50

1000  (10-Speed Bicycle)
               102                    Johnson, Sam              162,50
               106                   Heining, Heinz             175,00
               107                   Dalling, Jimmy             150,50
                                                               =========
   TOTAL ORDERS FOR 1000                                       488,00


                         TOTAL ORDERS                         $1368,25
                                                               =========
```

Example Program 1: A Two-set Thread

```
1000  !
1010  !   OUTSTANDING ORDERS REPORT  (NOT INCLUDING ALL DETAIL)
1020  !
1030      INTEGER S(9) , Prod_no
1040      DIM B$[12],P$[10],Buf$[170]
1050      DIM Desc$[30],Order_no$[30],Name$[30]
1060      DISP "~~"                        !   CLEAR SCREEN
1090      B$="  SAD, SALES"
1100      P$="MANAGER"
1110      DBOPEN  (B$,P$,1,S(*) )          !   OPEN DATA BASE
1150  !
1160  !   SET UP ALL APPROPRIATE RELATIONSHIPS
1170  !
1180      DBASE IS B$
```

```
1190      IN DATA SET “PRODUCT” USE Prod_no,Desc$
1200      IN DATA SET “CUSTOMER” USE ALL
1220  !
1230  !   SET UP THE WORKFILE
1240  !
1310      ASSIGN “XYZ” TO #1
1320      WORKFILE IS #1; THREAD IS “PRODUCT”,”CUSTOMER”
1330  !
1340  !   SORT THE STRUCTURE
1350  !
1360      SORT BY Prod_no,Order_no$
1400  !
1410  !   INITIALIZE VARIABLES & PRINT REPORT HEADER
1420  !
1430  Rep:Total=Master_total=0
1440      S1=0
1450      PRINT TAB(20);”OUTSTANDING ORDERS LIST”;LIN(1)
1460      PRINT “PRODUCT”;SPA(8);”ORDER NUMBER”;SPA(4);”CUSTOMER
          NAME”;SPA(14);”PRICE”;LIN(1);RPT$(”-”,63);LIN(1)
1461  !
1462  !   PRODUCE THE REPORT
1463  !
1470      FOR Z=1 TO WFLEN(1)
1480        READ #1;R1,R2
1570  !
1580  !     PRINT TRAILER FOR PRODUCT (IF NEEDED)
1590  !
1600  !   (SKIP IF SAME PRODUCT AS BEFORE, OR FIRST TIME THRU LOOP)
1610  !
1620        IF (R1=S1) OR NOT S1 THEN Notot
1630          PRINT USING Tot_image;VAL$(Prod_no),Total
1640          Total=0
1650  !
1660  !     PRINT HEADER FOR PRODUCT (IF NEEDED)
1670  !
1680  Notot:IF R1=S1 THEN Skip1
1690          DBGET (B$,”PRODUCT”,4,S(*),”@”,Buf$,R1)
1710          S1=R1
1720          PRINT VAL$(Prod_no);” (”;TRIM$(Desc$);”)”
1810  !
1820  !     PRINT ORDERS
1830  !
1840  Skip1:DBGET (B$,”CUSTOMER”,4,S(*),”@”,Buf$,R2)
1860        PRINT TAB(16);
1870        PRINT USING Itm_image;Order_no$,Name$,Price
1880  Itm_image:IMAGE 16A,22A,2X,5DRDD
1890  !
1900  !     ACCUMULATE TOTALS
1910  !
1920        Total=Total+Price
1940        Master_total=Master_total+Price
1950      NEXT Z
1960  !
1970  !   PRINT FINAL TOTALS
1980  !
2000      PRINT USING Tot_image;VAL$(Prod_no),Total
2010      PRINT USING Mstr_image;Master_total
2040  Tot_image:IMAGE 54X,9(”=”) / 3X,
      ”TOTAL ORDERS FOR “,10A,24X,6DRDD /
2050  Mstr_image:IMAGE // 25X,”TOTAL ORDERS”,14X,
      ”$”8DRDD /   54X,9(”=”)
```

```
2130      END
```

Example Program 2: Using Only One Set Instead of Two

```
1000  !
1010  !   OUTSTANDING ORDERS REPORT  (NOT INCLUDING ALL DETAIL)
1020  !
1030      INTEGER S(9),Product_no,Prod_no
1040      DIM B$[12],P$[10],Buf$[170]
1050      DIM Desc$[30],Order_no$[30],Name$[30]
1060      DISP "~~"                    !  CLEAR SCREEN
1090      B$="  SAD,SALES"
1100      P$="MANAGER"
1110      DBOPEN (B$,P$,1,S(*))        !  OPEN DATA BASE
1120      IF S(0) THEN Dberr
1150  !
1160  !   SET UP ALL APPROPRIATE RELATIONSHIPS
1170  !
1180      DBASE IS B$
1190      IN DATA SET "PRODUCT" USE Prod_no,Desc$
1200      IN DATA SET "CUSTOMER" USE ALL
1220  !
1230  !   SET UP THE WORKFILE
1240  !
1310      ASSIGN "XYZ" TO #1
1320      WORKFILE IS #1; THREAD IS "CUSTOMER"
1330  !
1340  !   SORT THE STRUCTURE
1350  !
1360      SORT BY Product_no,Order_no$
1400  !
1410  !   INITIALIZE VARIABLES & PRINT REPORT HEADER
1420  !
1430  Rep:Total=Master_total=0
1440      Prod_no=-1
1450      PRINT TAB(20);"OUTSTANDING ORDERS LIST";LIN(1)
1460      PRINT "PRODUCT";SPA(8);"ORDER NUMBER";SPA(4);"CUSTOMER
          NAME";SPA(14);"PRICE";LIN(1);RPT$("-",63);LIN(1)
1461  !
1462  !   PRODUCE THE REPORT
1463  !
1470      FOR Z=1 TO WFLEN(1)
1480        READ #1;R1
1490        DBGET (B$,"CUSTOMER",4,S(*),"@",Buf$,R1)
1500        IF S(0) THEN Dberr
1570  !
1580  !     PRINT TRAILER FOR PRODUCT (IF NEEDED)
1590  !
1600  !   (SKIP IF SAME PRODUCT AS BEFORE, OR FIRST TIME THRU LOOP)
1610  !
1620        IF (Prod_no=Product_no) OR (Prod_no%<0) THEN Notot
1630          PRINT USING Tot_image;VAL$(Product_no),Total
1640          Total=0
1650  !
1660  !     PRINT HEADER FOR PRODUCT (IF NEEDED)
1670  !
1680  Notot:IF Prod_no=Product_no THEN Skip1
1690          DBGET (B$,"PRODUCT",7,S(*),"@",Buf$,Product_no)
1700          IF S(0) THEN Dberr
```

```
1720          PRINT VAL$(Prod_no);" (";TRIM$(Desc$);")"
1810  !
1820  !     PRINT ORDERS
1830  !
1860  Skip1:PRINT TAB(16);
1870        PRINT USING Itm_image;Order_no$,Name$,Price
1880  Itm_image:IMAGE 16A,22A,2X,5DRDD
1890  !
1900  !     ACCUMULATE TOTALS
1910  !
1920        Total=Total+Price
1940        Master_total=Master_total+Price
1950      NEXT Z
1960  !
1970  !   PRINT FINAL TOTALS
1980  !
2000      PRINT USING Tot_image;VAL$(Prod_no),Total
2010      PRINT USING Mstr_image;Master_total
2040  Tot_image:IMAGE 54X,9("=") / 3X,
      "TOTAL ORDERS FOR ",10A,24X,6DRDD /
2050  Mstr_image:IMAGE // 25X,"TOTAL ORDERS",14X,
      "$"8DRDD / 54X,9("=")
2060      STOP
2070  !
2080  !     ERROR TERMINATION ROUTINE
2090  !
2100  Dberr:DISP LIN(2);"STATUS ERROR ";VAL$(S(0));
      " IN LINE ";S(6)
2170      END
```

Itemized Order List Programs

The remaining three programs are all extensions to the previous programs, in that the report is essentially the same, but each order has its option listed along with it. In example programs 3 and 4 the options are listed in sorted order. A report that could be printed by these programs is shown on page 3-7/8. Example program 5 lists the options in the order they occur along the chain in the OPTION detail. The report produced this program is shown on pages 3-14/15.

Note that there is a blank option following the customer name. There is actually an entry with a blank option number field in ORDER for each order placed. This record contains the price of the product, and the all-blank field is used to force this entry to occur before any of the options to guarantee that it will be the first in the chain.

The blank entry also serves another function. If it were not included, then any order sold with no options would have no record in the OPTION set. This would generate an incomplete hierarchy for such orders, so they would not occur in the workfile generated by programs 3 and 4, though program 5 could be modified to handle such orders.

Example program 3 uses a four-set thread (see line 1320). The construction of this thread is discussed in Chapter 1. Note, that although four pointers must be read from the workfile (see line 1480), the third pointer, R3, is never used. This third pointer is just the place holder to skip over the information in the automatic set, ORDER. Again, the change in record number pertaining to the PRODUCT set is used to trigger the headers and trailers for new products (via variables R1 and S1). A similar technique is used to detect the change in order number (via variables R2 and S2).

Example program 3 is another case of bad programming. Example program 4 cleans up these problems. It adds status checks for data base calls, error trapping (see line 1070) and HALT key trapping (see line 1080). Also, all the previous examples have assumed that the data file “XYZ:” exists for use as a workfile. Example program 4 now checks to see if the workfile exists and creates it if it does not. It stops if the file is protected or is of the wrong type.

For reasons detailed in Chapter 4, long threads are undesirable and should be avoided when possible. As in example program 2, the PRODUCT set can be eliminated from the thread by use of a calculated-access DBGET. This reduces the thread length to three. Also, if it is not particularly important to have the options listed in sorted order, a DBFIND on the OPTION set using the order number from

the CUSTOMER set may be done. This allows chained mode DBGETs to be used to get the options. Listing will thus be in the chain order (the order the options appeared in on the original order). This reduces the thread length to only one set, the CUSTOMER set. Program example 5 shows how this could be done.

In example program 5, as in example 2, the actual product number is used to determine when headers and trailers are required. However, since each record in the workfile corresponds to a new order, no special logic is needed to detect change in order number; The header and trailer each occur every time through the loop. A special imbedded FOR loop is added, however, to print out the options (see lines 1835 through 1945).

Itemized Options List Report (sorted order)

```
                    OUTSTANDING ORDERS LIST

PRODUCT        ORDER NUMBER          CUSTOMER NAME              PRICE

50  (Tricycle)
               110                   Gissing, Malcom             45,00
                                                               --------
                                                                  45,00
                                                              ---------
   TOTAL ORDERS FOR 50                                            45,00

100 (Standard Bicycle)
               101                   Noname, Joseph              75,00
                                                    Horn          2,50
                                                               --------
                                                                  77,50

               103                   Hernandes, Jose             75,00
                                                    Fan          10,00
                                                    Horn         10,00
                                                    Light         5,00
                                                    Mud Flaps     7,25
                                                    Stripes       2,50
                                                               --------
                                                                 109,75

               108                   Arauja, Luciano A.          75,00
                                                    Horn          5,00
                                                               --------
                                                                  80,00
                                                              ---------
   TOTAL ORDERS FOR 100                                           267,25

300  (3-Speed Bicycle)
               104                   Houseman, Sean             110,00
                                                    Light         5,00
                                                    Super Tire   18,00
                                                               --------
                                                                 133,00
                                                              ---------
   TOTAL ORDERS FOR 300                                           133,00
```

```
500  (5-Speed Bicycle)
             100                  Smith, Thomas A.          125,00
                                                   Basketle  45,00
                                                   Light      5,00
                                                           ---------
                                                             175,50

             105                  Sono, Jomo A.             125,00
                                                   Horn       2,50
                                                   Reflector  7,50
                                                           --------
                                                             135,00

             109                  Bekker, Bart                125,00
                                                           ---------
                                                             125,00

                                                           ---------
   TOTAL ORDERS FOR 500                                      435,50

1000  (10-Speed Bicycle)
             102                  Johnson, Sam              150,00
                                                   Chrome     12,50
                                                           --------
                                                             162,50

             106                  Heining, Heinz            150,00
                                                   Basket     15,00
                                                   Light      10,00
                                                           --------
                                                             175,00

             107                  Dalling, Jimmy              150,00
                                                           --------
                                                             150,00
                                                           ---------
   TOTAL ORDERS FOR 1000                                     487,50

                      TOTAL ORDERS                          $1368,25
                                                           ---------
```

Example Program 3: A Four-set Thread

```
1000  !
1010  !   OUTSTANDING ORDERS REPORT  (INCLUDING ALL DETAIL)
1020  !
1030      INTEGER S(9),Prod_no
1040      DIM B$[12],P$[10],Buf$[170]
1050     DIM Desc$[30],Order_no$[30],Name$[30],Option_desc$[10]
1060      DISP  “   “;                  !  CLEAR SCREEN
1090      B$=”  SAD,SALES”
1100      P$=”MANAGER”
1110      DBOPEN (B$,P$,1,S(*))         !  OPEN DATA BASE
1120      IF S(0) THEN Dberr
1150  !
1160  !   SET UP ALL APPROPRIATE RELATIONSHIPS
1170  !
1180      DBASE IS B$
1190      IN DATA SET “PRODUCT” USE Prod_no,Desc$
```

```
1200      IN DATA SET “CUSTOMER” USE ALL
1210      IN DATA SET “OPTION” USE SKP 1,Option_desc$,PO
1220  !
1230  !   SET UP THE WORKFILE
1240  !
1310      ASSIGN “XYZ” TO #1
1320      WORKFILE IS #1;THREAD IS “PRODUCT”,”CUSTOMER”,
          ”ORDER”,”OPTION”
1330  !
1340  !   SORT THE STRUCTURE
1350  !
1360      SORT BY Prod_no,Order_no$,Option_desc$
1400  !
1410  !   INITIALIZE VARIABLES & PRINT REPORT HEADER
1420  !
1430  Rep:Sub_total=Total=Master_total=0
1440      S1=S2=0
1450      PRINT TAB(30);”OUTSTANDING ORDERS LIST”;LIN(1)
1460     PRINT “PRODUCT”;SPA(8);”ORDER NUMBER”;SPA(10);”CUSTOMER
          NAME”;SPA(9);”OPTIONS”;SPA(8);”PRICE”;LIN(1);
          RPT$(”-”,79);LIN(1)
1461  !
1462  !   PRODUCE THE REPORT
1463  !
1470      FOR Z=1 TO WFLEN(1)
1480        READ #1;R1,R2,R3,R4
1490  !
1500  !     PRINT TRAILER FOR ORDER (IF NEEDED)
1510  !
1520  !    (SKIP IF SAME ORDER AS BEFORE, OR FIRST TIME THRU LOOP)
1530  !
1540        IF (R2=S2) OR NOT S2 THEN Nosub
1550          PRINT USING Sub_image;Sub_total
1560          Sub_total=0
1570  !
1580  !     PRINT TRAIILER FOR PRODUCT (IF NEEDED)
1590  !
1600  !   (SKIP IF SAME PRODUCT AS BEFORE, OR FIRST TIME THRU LOOP)
1610  !
1620  Nosub:IF (R1=S1) OR NOT S1 THEN Notot
1630          PRINT USING Tot_image;VAL$(Prod_no),Total
1640          Total=0
1650  !
1660  !     PRINT HEADER FOR PRODUCT (IF NEEDED)
1670  !
1680  Notot:IF R1=S1 THEN Skip1
1690          DBGET (B$,”PRODUCT”,4,S(*),”@”,Buf$,R1)
1710          S1=R1
1720          PRINT VAL$(Prod_no);”  (”;TRIM$(Desc$);”)”
1730  !
1740  !     PRINT HEADER FOR ORDER (IF NEEDED)
1750  !
1760  Skip1:IF R2=S2 THEN Skip2
1770          DBGET (B$,”CUSTOMER”,4,S(*),”@”,Buf$,R2)
1790          PRINT TAB(20);Order_no$;TAB(38);Name$[1,21];
1800          S2=R2
1810  !
1820  !     PRINT OPTIONS
1830  !
1840  Skip2:DBGET (B$,”OPTION”,4,S(*),”@”,Buf$,R4)
1860        PRINT TAB(60);
```

```
1870        PRINT USING Itm_image;Option_desc$,PO
1880  Itm_image:IMAGE 10A,2X,5DRDD
1890  !
1900  !     ACCUMULATE TOTALS
1910  !
1920        Total=Total+PO
1930        Sub_total=Sub_total+PO
1940        Master_total=Master_total+PO
1950      NEXT Z
1960  !
1970  !   PRINT FINAL TOTALS
1980  !
1990      PRINT USING Sub_image;Sub_total
2000      PRINT USING Tot_image;VAL$(Prod_no),Total
2010      PRINT USING Mstr_image;Master_total
2030  Sub_image:IMAGE 71X,8("-") / 71X,5DRDD /
2040  Tot_image:IMAGE 70X,9("-") / 11X,"TOTAL ORDERS FOR ",10A,
      32 X,6DRDD /
2050  Mstr_image:IMAGE // 31X,"TOTAL ORDERS",24X,"$"8DRDD / 70X,
      9("-")
2160      END
```

Example Program 4: Using Only One Set Instead of Four

```
1000  !
1010  !   OUTSTANDING ORDERS REPORT (INCLUDING ALL DETAIL)
1020  !
1030      INTEGER S(9),Prod_no
1040      DIM B$[12],P$[10],Buf$[170]
1050      DIM Desc$[30],Order_no$[30],Name[30],Option_desc$[10]
1060      DISP "    ";                    ! CLEAR SCREEN
1070     ON ERROR GOTO Error        ! SET UP ERROR AND HALT TRAPS
1080      ON HALT GOTO Halt
1090      B$="  SAD,SALES"
1100      P$="MANAGER"
1110      DBOPEN (B$,P$,1,S(*))       ! OPEN DATA BASE
1120      IF S(0) THEN Dberr
1150  !
1160  !   SET UP ALL APPROPRIATE RELATIONSHIPS
1170  !
1180      DBASE IS B$
1190      IN DATA SET "PRODUCT" USE Prod_no,Desc$
1200      IN DATA SET "CUSTOMER" USE ALL
1210      IN DATA SET "OPTION" USE SKP 1, Option_desc$,PO
1220  !
1230  !   SET UP THE WORKFILE
1240  !
1250      ASSIGN "XYZ" TO #1,Z
1260      IF Z%<2 THEN Ok
1270      DISP "CAN'T ASSIGN THE WORKFILE!"
1280      STOP
1290  Ok: IF NOT Z THEN Aok         ! CREATE  WORKFILE IF NECESSARY
1300      FCREATE "XYZ" ,0
1310      ASSIGN "XYZ" TO #1
1320   Aok:WORKFILE IS #1;THREAD IS "PRODUCT","CUSTOMER","ORDER",
      "OPTION"
1330  !
1340  !   SORT THE STRUCTURE
```

```
 1350  !
 1360       SORT BY Prod_no,Order_no$,Option_desc$
 1370       IF WFLEN(1) THEN Rep
1380      DISP "THERE ARE NO ENTRIES IN THE STRUCTURE TO REPORT ON."
 1390       STOP
 1400  !
 1410  !    INITIALIZE VARIABLES & PRINT REPORT HEADER
 1420  !
 1430  Rep:Sub_total=Total=Master_total=0
 1440       S1=S2=0
 1450       PRINT TAB(30);"OUTSTANDING ORDERS LIST";LIN(1)
1460      PRINT "PRODUCT";SPA(8);"ORDER NUMBER";SPA(10);"CUSTOMER
           NAME";SPA(9);"OPTIONS";SPA(8);"PRICE";LIN(1);
           RPT$("-",79);LIN(1)
 1461  !
 1462  !    PRODUCE THE REPORT
 1463  !
 1470       FOR Z=1 TO WFLEN(1)
 1480         READ #1;R1,R2,R3,R4
 1490  !
 1500  !      PRINT TRAILER FOR ORDER (IF NEEDED)
 1510  !
1520  !       (SKIP IF SAE ORDER AS BEFORE, OR FIRST TIME THRU LOOP)
 1530  !
 1540         IF (R2=S2) OR NOT S2 THEN Nosub
 1550           PRINT USING Sub_image;Sub_total
 1560           Sub_total=0
 1570  !
 1580  !      PRINT TRAILER FOR PRODUCT (IF NEEDED)
 1590  !
1600  !     (SKIP IF SAME PRODUCT AS BEFORE,OR FIRST TIME THRU LOOP)
 1610  !
 1620  Nosub:IF (R1=S1) OR NOT S1 THEN Notot
 1630           PRINT USING Tot_image;VAL$(Prod_no),Total
 1640           Total=0
 1650  !
 1660  !      PRINT HEADER FOR PRODUCT (IF NEEDED)
 1670  !
 1680  Notot:IF R1=S1 THEN Skip1
 1690           DBGET (B$,"PRODUCT",4,S(*),"@",Buf$,R1)
 1700           IF S(0) THEN Dberr
 1710           S1=R1
 1720           PRINT VAL$(Prod_no);" (";TRIM$(Desc$);")"
 1730  !
 1740  !      PRINT HEADER FOR ORDER (IF NEEDED)
 1750  !
 1760  Skip1:IF R2=S2 THEN Skip2
 1770           DBGET (B$,"CUSTOMER",4,S(*),"@",Buf$,R2)
 1780           IF S(0) THEN Dberr
 1790           PRINT TAB(20);Order_desc$;TAB(38);Name$[1,21];
 1800           S2=R2
 1810  !
 1820  !      PRINT OPTIONS
 1830  !
 1840  Skip2:DBGET (B$,"OPTION",4,S(*),"@",Buf$,R4)
 1850         IF S(0) THEN Dberr
 1860         PRINT TAB(60);
 1870         PRINT USING Itm_image;Option_no$,PO
 1880  Itm_image:IMAGE 10A,2X,5DRDD
 1890  !
 1900  !      ACCUMULATE TOTALS
```

```
 1910  !
 1920        Total=Total+PO
 1930        Sub_total=Sub_total+PO
 1940        Master_total=Master_total+PO
 1950      NEXT Z
 1960  !
 1970  !   PRINT FINAL TOTALS
 1980  !
 1990      PRINT USING Sub_image;Sub_total
 2000      PRINT USING Tot_image;VAL$(Prod_no),Total
 2010      PRINT USING Mstr_image;Master_total
 2020      DISP "REPORT COMPLETE."
 2030  Sub_image:IMAGE 71X,8("-") / 71X,5DRDD /
 2040  Tot_image:IMAGE 70X,9("=") / 11X,"TOTAL ORDERS FOR ",10A,
       32 X,6DRDD /
 2050  Mstr_image:IMAGE // 31X,"TOTAL ORDERS",24X,"$"8DRDD /
       70X,9("=")
 2060      STOP
 2070  !
 2080  !     ERROR AND HALT TERMINATION ROUTINES
 2090  !
 2100 Dberr:DISP LIN(2);"STATUS ERROR ";VAL(S(0));" IN LINE";S(6)
 2110      STOP
 2120  Error:DISP LIN(2);"UNEXPECTED ";ERRM$
 2130      STOP
 2140  Halt:PRINT LIN(2)
 2150      DISP LIN(2);"PROGRAM TERMINATED."
 2160      END
```

Itemized Options List Report (unsorted order)

```
                    OUTSTANDING ORDERS LIST

PRODUCT        ORDER NUMBER          CUSTOMER NAME              PRICE

50  (Tricycle)
               110                   Gissing, Malcom             45,00
                                                               --------
                                                                45,00
                                                               ========
   TOTAL ORDERS FOR 50                                          45,00

100 (Standard Bicycle)
               101                   Noname, Joseph              75,00
                                                      Horn          2,50
                                                               --------
                                                                77,50

               103                   Hernandes, Jose             75,00
                                                      Light         5,00
                                                      Mud Flaps     7,25
                                                      Horn         10,00
                                                      Stripes       2,50
                                                      Fan          10,00
                                                               --------
                                                                109,75

               108                   Araujo, Luciano A.          75,00
                                                      Horn          5,00
```

```
                                                                --------
                                                                   80,00
                                                                =========
   TOTAL ORDERS FOR 100                                           267,25

300  (3-Speed Bicycle)
             104                  Houseman, Sean                   110,00
                                                          SuperTire 18,00
                                                          Light      5,00
                                                                --------
                                                                  133,00
                                                                =========
   TOTAL ORDERS FOR 300                                           133,00

500  (5-Speed Bicycle)
             100                  Smith, Thomas A.                 125,00
                                                          Light      5,00
                                                          Basketle  45,50
                                                                --------
                                                                  175,50

             105                  Sono, Jomo A.                    125,00
                                                          Horn       2,50
                                                          Reflector  7,50
                                                                --------
                                                                  135,00

             109                  Bekker, Bart                     125,00
                                                                --------
                                                                  125,00

                                                                =========
   TOTAL ORDERS FOR 500                                           435,50

1000  (10-Speed Bicycle)
             102                  Johnson, Sam                     150,00
                                                          Chrome    12,50
                                                                --------
                                                                  162,50

             106                  Heining, Heinz                   150,00
                                                          Light     10,00
                                                          Basket    15,00
                                                                --------
                                                                  175,00
                                                                --------
             107                  Dalling, Jimmy                   150,00
                                                                --------
                                                                  150,00
                                                                =========
   TOTAL ORDERS FOR 1000                                          487,50

                       TOTAL ORDERS                              $1368,25
                                                                =========
```

Example 5: Listing Options in Unsorted Order

```
1000  !
1010  !   OUTSTANDING ORDERS REPORT (INCLUDING ALL DETAIL)
1020  !
```

```
   1030      INTEGER S(9),Product_no,Prod_no
   1040      DIM B$[12],P$[10],Buf$[170]
   1050     DIM Desc$[30],Order_no$[30],Name$[30],Option_desc$[10]
   1060      DISP "   ";                    ! CLEAR SCREEN
   1070    ON ERROR GOTO Error         ! SET UP ERROR AND HALT TRAPS
   1080      ON HALT GOTO Halt
   1090      B$="  SAD,SALES"
   1100      P$="MANAGER"
   1110      DBOPEN (B$,P$,1,S(*))        ! OPEN DATA BASE
   1120      IF S(0) THEN Dberr
   1150  !
   1160  !   SET UP ALL APPROPRIATE RELATIONSHIPS
   1170  !
   1180      DBASE IS B$
   1190      IN DATA SET "PRODUCT" USE Prod_no,Desc$
   1200      IN DATA SET "CUSTOMER" USE ALL
   1210      IN DATA SET "OPTION" USE SKIP 1,Option_desc$,PO
   1220  !
   1230  !   SET UP THE WORKFILE
   1240  !
   1250      ASSIGN "XYZ" TO #1,Z
   1260      IF Z%<2 THEN Ok
   1270      DISP "CAN'T ASSIGN THE WORKFILE!"
   1280      STOP
   1290  Ok: IF NOT Z THEN Aok          ! CREATE WORK FILE IF NECESSARY
   1300      FCREATE "XYZ",0
   1310      ASSIGN "XYZ" TO #1
   1320   Aok:WORKFILE IS #1;THREAD IS "CUSTOMER"
   1330   !
   1340   !   SORT THE STRUCTURE
   1350   !
   1360      SORT BY Product_no,Order_no$
   1370      IF WFLEN(1) THEN Rep
   1380    DISP "THERE ARE NO ENTRIES IN THE STRUCTURE TO REPORT ON."
   1390      STOP
   1400   !
   1410   !   INITIALIZE VARIABLES & PRINT REPORT HEADER
   1420   !
   1430   Rep:Total=Master_total=0
   1440      Prod_no=-1
   1450      PRINT TAB(30);"OUTSTANDING ORDERS LIST";LIN(1)
   1460     PRINT "PRODUCT";SPA(8);"ORDER NUMBER";SPA(10);"CUSTOMER
   NAME";SPA(9);"OPTIONS";SPA(8);"PRICE";LIN(1);
RPT$("-",79);LIN(1)
   1461  !
   1462  !   PRODUCE THE REPORT
   1463  !
   1470      FOR Z=1 TO WFLEN(1)
   1480        READ #1;R1
   1490        DBGET (B$,"CUSTOMER",4,S(*),"@",Buf$,R1)
   1500        IF S(0) THEN Dberr
   1520  !        (SKIP IF SAME ORDER AS BEFORE, OR FIRST TIME THRU
LOOP)
   1530  !
   1570  !
   1580  !     PRINT TRAILER FOR PRODUCT (IF NEEDED)
   1590  !
   1600  !       (SKIP IF SAME PRODUCT AS BEFORE, OR FIRST TIME THRU
LOOP)
   1610  !
   1620  Nosub:IF (Prod_no=Product_no) OR (Prod_no%<0) THEN Notot
```

```
1630          PRINT USING Tot_image;VAL$(Prod_no),Total
1640          Total=0
1650  !
1660  !     PRINT HEADER FOR PRODUCT (IF NEEDED)
1670  !
1680  Notot:IF Prod_no=Product_no THEN Skip1
1690          DBGET (B$,"PRODUCT",7,S(*),"@",Buf$,Product_no)
1700          IF S(0) THEN Dberr
1720          PRINT VAL$(Prod_no);" (";TRIM$(Desc$);")"
1730  !
1740  !     PRINT HEADER FOR ORDER
1750  !
1790  Skip1:PRINT TAB(20);Order_no$;TAB(38);Name$[1,21];
1810  !
1820  !     PRINT OPTIONS
1830  !
1835        DBFIND (B$,"OPTION",1,S(*),"ORDER-NO",Order_no$)
1836        IF S(0) THEN Dberr
1840        FOR C=1 TO S(5)
1845          DBGET (B$,"OPTION",5,S(*),"@",Buf$,0)
1850          IF S(0) THEN Dberr
1860          PRINT TAB(60);
1870          PRINT USING Itm_image;Option_desc$,PO
1880  Itm_image:IMAGE 10A,2X,5DRDD
1890  !
1900  !     ACCUMULATE TOTALS
1910  !
1920          Total=Total+PO
1930          Sub_total=Sub_total+PO
1940          Master_total=Master_total+PO
1945        NEXT C
1946        PRINT USING Sub_image;Sub_total
1947        Sub_total=0
1950      NEXT Z
1960  !
1970  !   PRINT FINAL TOTALS
1980  !
2000      PRINT USING Tot_image;VAL$(Prod_no),Total
2010      PRINT USING Mstr_image;Master_total
2020      DISP "REPORT COMPLETE."
2030  Sub_image:IMAGE 71X,8("-") / 71X,5DRDD /
2040  Tot_image:IMAGE 70X,9("=") / 11X,"TOTAL ORDERS FOR ",10A,
          32X,6DRDD /
2050  Mstr_image:IMAGE // 31X,"TOTAL ORDERS",24X,
          "$"8DRDD / 70X,9("=")
2060      STOP
2070  !
2080  !     ERROR AND HALT TERMINATION ROUTINES
2090  !
2100  Dberr:DISP LIN(2);"STATUS ERROR ";VAL$(S(0));" IN LINE";S(6)
2110      STOP
2120  Error:DISP LIN(2);"UNEXPECTED ";ERRM$
2130      STOP
2140  Halt:PRINT LIN(2)
2150      DISP LIN(2);"PROGRAM TERMINATED."
2160      END
```

4

Programming Considerations

Introduction

A great deal can be done toward speeding up SORT operations by following certain programming guidelines. This chapter presents factors which should be considered in program and data base design to optimize sorting speed. Use of some factors will always result in optimum sort speed. Use of other factors may increase or decrease speed, depending on how they are implemented; trial and error will determine the optimum combination for a given application.

Software Optimization

The most significant gains in terms of speed improvement can be made by following some simple rules in designing programs using SORT operations. There are essentially three classes of rules which will be covered:

- Generally true rules.
- Rules which are to be used if no FINDs, QFINDS, SORTs or PRINT #s have been done on the workfile (REC=0).
- Rules which are to be used if pointers have been put in the workfile (REC \\neq0).

General Rules

The most important rule is to keep thread length minimal. Also, if either the first or the last set in the thread is a master and the only item that will ever used out of it for FINDing, QFINDing or SORTing is the search item, it can be eliminated. This is possible since that item also exists in the associated detail thus enabling a calculated access DBGET to be used to get the additional information out of the master.

Turn off all possible IN DATA SETs (via the FREE option) before doing a FIND or QFIND. If a particular IN DATA SET is active for some data set in the thread and no values from that set are needed to evaluate the selection expression, FREEing that IN DATA SET stops FIND/QFIND from reading information from that set.

Rules When REC = 0

When there are no pointers in the workfile, SORT BY and FIND statements must do a serial read of the first set in the thread. If you need only a few entries from a large data set, use QFIND to locate requested entries before executing FIND or SORT BY.

Specifying the conditional expression in a QFIND statement reduces overhead against QFIND/FIND sequence, because records are only processed once.

QFIND works by appending, i.e. adding entries. This is particularly useful if data conditions are diferent.

If an item occurs in more than one set in the thread (normally because it is a search item), it should be selected to come from the set closest to the start of the thread. For FIND it is very important to notice all appropriate sets (ones with IN DATA SETs active) have been read. Thus, if the needed set can be restricted to those near the head of the thread, the expression can be evaluated sooner.

If the FIND condition is a series of conditions separated by ANDs, it may be beneficial to break them up into separate FINDs. In general, if some of the clauses pertain only to the first set in the thread and they will select significantly less than all the data available, then it is best to construct two FIND statements (the first one pertaining only to the set at the head of the thread). Remember when doing this to deactivate and reactivate the IN DATA SET relations (via FREE) to maximize effect.

Rules When REC ≠ 0

Here again it is a good idea to deactivate all unused IN DATA SET relations pertaining to sets in the thread. In the case of SORT BY, the fewer sets involved the better. Remember that if one of the sort items is a search item it may be possible to select it from one to several sets. Select it from the set which allows you to deactivate the most IN DATA SETs.

In the case of FIND the same things as mentioned for SORT BY also apply. However, breaking up a complex FIND separated by ANDs into several FINDs may increase speed if (and only if) some of the clauses separated by the ANDs do not involve the same sets or involve fewer sets than the other clauses. If this is the case, the clauses which have the fewest sets involved and the lowest probability of being true should be executed first. Remember, again, that the only way FIND knows which sets are involved is by which IN DATA SETs are active. Clearly, most of these rules assume the programmer has a good understanding of the form the data will take (in terms of probable events). When in doubt, perform tests.

Table 2 **Overview**

	REC =0	REC ≠ 0
	(no previous QFIND, FIND, SORT BY or PRINT)	(previous FIND, SORT BY or PRINT #)
FIND	Keep thread lenght short. Make sure the last set with an IN DATA SET active on it is as close to the start of the thread as possible	Make sure IN DATA SET are active on only those sets from which information must be retrieved
SORT BY	Keep thread lenght short.	Make sure sort keys come from as few sets as possible.

ALWAYS:

1) Minimize thread length.
2) Minimize complexity of the FIND selection expression.
3) Minimize total sort key.

A

Schema Listing for the SAD Data Base

```
$CONTROL              LIST, TABLE, ROOT
$TITLE  “Sales Analysis Data Base”

BEGIN DATA BASE    SAD;  <<CUSTOMER SALES ANALYSIS DATA BASE>>

PASSWORDS:                 <<Password definition>>
                   10      SALESMAN;
                   15      MANAGER;
                    3      SECRTARY; <<WILL HAVE READ ACCESS ONLY>>

ITEMS                      <<Item definition>>

                   ADDRESS,           2 X30;
                           <<2 LINES OF ADDRESS ALLOWED>>
                   CITY,                X16;
                   COUNTRY,             X12;
                           <<PATH FOR ORDER-DATE, SHIP-DATE>>
                   DATE,                I;
                   NAME,                X30;
                   OPTION-DESC,         X10;
                   OPTION-PRICE,        L;
                   OPTION-TYPE,         I;
                   ORDER-DATE,          I;   <<MUST BE YYMM>>
                   ORDER-NO,            X10;
                   PRICE,               L;
                   PRODUCT-NO,          I;
                   PROD-DESC,           X30;
                   REGION,              X6;
                   REGION-DESC,         X30;
                   REGION,TYPE,         I;
                   SALESPERSON,         X4;
                   SHIP-DATE,           I;    <<MUST BE YYMM>>
                   STATE,               X6;
                   ZIP-CODE,            X8

   SETS:                          << Set defintion >>


   NAME:         DATE,  A (3/10,15);
   ENTRY:        DATE (2);
   CAPACITY:     51;

   NAME:         ORDER, A (3/10,15);
   ENTRY:        ORDER-NO (2);
   CAPACITY:     101;

   NAME:         PRODUCT, M (3,10/15);
   ENTRY:        PRODUCT-NO (1)
                 PROD-DESC; (1)
   CAPACITY:     11;

   NAME:         LOCATION, M(3,10/15);
   ENTRY:        REGION (1),
                 REGION-DESC,
                 REGION-TYPE;
   CAPACITY:     17;

   NAME:         OPTION, D (3/10,15);
   ENTRY:        ORDER-NO (ORDER),
                 OPTION-DESC,
                 OPTION-PRICE,
                 OPTION-TYPE;
   CAPACITY:     300;

   NAME:         CUSTOMER, DETAIL (3/10,15);
   ENTRY:        ORDER-NO (ORDER),
                 NAME,
```

```
            ADDRESS,
            CITY,
            STATE,
            COUNTRY,
            ZIP-CODE,
            ORDER-DATE (DATE),
            SHIP-DATE (DATE),
            REGION (LOCATION),
            PRODUCT-NO (PRODUCT),
            PRICE,
            SALESPERSON;
CAPACITY:   100;

END
```

Appendix B SORT Error Codes

211 **`No DBASE IS statement active or bad data base specifier.`** Attempt to execute an IN DATA SET or WORKFILE IS # without previously executing a DBASE IS or the data base that the DBASE IS was executed for has been closed. Or bad data base specified in DBASE IS.

212 **`Specified data set not found.`** An improper set name or number was specified.

230 **`Improper nesting of SORT statements.`** An attempt was made to execute a SORT BY, QFIND, FIND, IN DATA SET, DBASE IS, etc. while nested inside one of these statements. This can only happen if an expression uses a multi-line function subprogram.

231 **`Cannot reactivate workfile, or file is not a workfile.`** An attempt is made to reactivate a workfile by using the WORKFILE IS # statement with no thread list, but the specified file is not a workfile.

233 **`No read access to specified data set.`** One of the data sets in the thread is not accessible with the current password.

234 **`Missing or improper data set linkage.`** For WORKFILE #, two adjacent sets in the thread list have no path between them, or the chain id specified does not refer to an existing chain.

235 **`No WORKFILE IS# statement active.`** Attempt to execute a SORT BY or FIND when no workfile has been declared or the workfile was closed (either by de-assigning it or by DBCLOSE).

236 **`Improper data/index item or data/index item not found.`** The item specified in the LINK parameter of WORKFILE IS # does not refer to an item for the specified set or the given item in the SORT BY list is not linked via IN DATA SET to an item in one of the sets in the thread. Improper or non-existent data/index specified in QFIND statement.

238 **`Improper synthetic linkage.`** The item in the LINK parameter of WORKFILE IS # either does not match the type of the search item in the master set following the LINK or it is not the first sub-item. Also, LINK is applied to a master set, or the set following the LINK is not of type master.

239 **`Insufficient space in workfile.`** Write error on workfile.

241 **`Improper operation attempt on workfile.`** Attempt to position the word pointer of a workfile to someplace other than word 1. Also, attempt to print an array on a workfile.

242 **`Improper READ # or PRINT # on workfile.`** A complete logical record was not read or written. The word pointer is reset to word 1.

243 **`Workfile contains invalid information.`** Attempt to access the workfile via SORT BY, QFIND, FIND, READ # or PRINT # after its contents have been destroyed by a disk error or CTRL Y stopping a FIND, QFIND or SORT.

247 **`Unexpected error accessing data base`**

248 **`Improper QFIND relation`**

249 **`Improper value type or improper number of value parameter`**

Index